Solid Model Databases: Techniques and Empirical Results

This paper presents techniques for managing solid models in relational database management systems. Our goal is to enable support for traditional database operations (sorting, distance metrics, range queries, nearest neighbors, etc.) on large databases of solid models. We introduce an approach to compare models based on shape using information extracted from the model boundary representation into Model Signature Graphs. We show how the Model Signature Graphs can be used to compute topological distances among models and how to use these measures to create metric spaces for indexing and clustering of solid models. We believe this work will begin to bridge the solid modeling and database communities, enabling new paradigms for interrogation of CAD datasets based on the engineering content of solid models. [DOI: 10.1115/1.1430233]

1 Introduction

Users of CAD/CAM software accumulate large collections of models that they create or obtain in the course of their design and manufacturing activities. As the number of models grows in such an archive, data and knowledge management becomes extremely difficult. Many CAD database systems currently do not use content and shape-based analysis of the models that they contain. Frequently these systems only index models based on simple properties that have little or nothing to do with the structure or shape of the part, i.e., part name, designer’s name, bills of materials or various text files associated with the model.

Databases of solid models that do make use of high-level, content-based analysis of the models typically rely on traditionally manual classification techniques, such as group technology coding [1]. This requires human observers to examine each model in the system and relate them to other models in the system. Automation of these processes has proven difficult, in large part because they were designed “by humans, for humans” prior to the ubiquity of computing technologies and fully digital CAD/CAM. The task of manual indexing is error prone, dependent on the personal judgment of the observer, and expensive. As a result, incorrectly or inconsistently indexed models may be ‘‘lost’’ in a large database. In order to accurately and efficiently maintain a large database of CAD models, it is clear that automated algorithmic techniques to index models based on their semantic and structural properties are essential.

Past research and current commercial systems for engineering information management and Product Data Management (PDM) have predominantly taken annotation and document-based approaches—where the solid modeling data itself is simply stored as a related file to other project documents. Research in CAD and engineering databases has produced great advances, such as representation schemas for STEP-based data elements, however existing technologies stop short of enabling content-based and semantic retrieval of solid modeling data of the types now available for other higher-dimensional media (images, audio and video).

We focus on developing efficient techniques to index and compare solid models based primarily on their shape and structural properties. To this end, we construct a mapping from the boundary representation (BRep) of the solid model to a graph-based data structure called a Model Signature Graph (MSG). We then project the graphs obtained into vector spaces, and examine distances between models based on the distances of their images in these spaces. These distances become central elements in indexing and clustering of the solid models. Distance-based indexing and clustering will enable efficient retrieval of models that are similar to a given query model. In addition, clustering a collection of models using our metrics will enable the understanding of distributions and patterns within that collection.

This paper is organized as follows: Section 2 provides a brief overview of related work on CAD and multi-media databases. Section 3 introduces our problem formulation and our approach; Section 4 presents the results of our empirical experiments based on this approach. Lastly, Section 5 discusses our research contributions as well as describes enhancements and modifications that we plan to experiment with in the future.

2 Background and Related Work

2.1 CAD Databases. Very little previous work has been done in the field of databases for CAD/CAM databases that make use of the structural or semantic information contained within solid models as a basis for organization, identification, and efficient retrieval of the models.

Hardwick et al. [2–5] provide a mapping between the software component architecture of Common Object Request Broker Architecture (CORBA) and the Standard for the Exchange of Product Model Data (STEP) to provide programmatic interfaces to parts and models stored persistently in a database. However, the focus of this work was not on the organization or retrieval based on semantic features, it was focused on the construction of programmatic interfaces to access the parameters and components of models and other STEP objects. From the mainstream database literature, there have been a couple approaches to index simple CAD models (in both cases fasteners) using only their 2D profiles [6,7].

A significant amount of work has been done in the field of vision and image recognition in shape similarity assessment and pattern recognition. Some of this research has been applied to the recognition of solid models using the techniques developed in the vision community.

Jain et al. [8] have developed techniques used to index multimedia image data by using feature vectors containing information such as the color, density, or intensity patterns of the image. Their work has been extended to CAD models by using 3D CAD data to create sets of 3D feature vectors. However, the features are not design or manufacturing features in a solid modeling sense—rather they refer to image-related properties of the CAD model.

Dartmouth’s 3D Base project [9–11] takes a different approach to performing similarity assessment on CAD models. Two solid models are compared by first rendering them using a voxel-based, or 3D-pixel based, representation. Invariant features of the shape are identified and used to narrow the model search space. Finally,
a symmetric-difference between selected regions of the voxel model is performed to evaluate the model similarity.

These vision-based techniques are highly dependent on pixels, color, texture, and other similar properties of the models. None of them are able to do topological comparisons of models under local deformations of scale, orientation, or other transformations. One technique that has been used to achieve transformation invariance is that of Aspect Graphs. This technique involves computing the set of topologically distinct projections of a solid model onto a plane, and constructing a graph by adding edges between projections whenever one can be directly transformed into another by rotating the model. While aspect graphs have been used for similarity assessment of solid models, it is extremely difficult to compute them for arbitrary solid models. Typically their use is restricted to simpler polyhedral models and surfaces of revolution, for which the aspect graph can be efficiently computed.

Previous work by Elinson, Nau and Regli [12,13] as well as by our group [14–17] has addressed the need for databases to be able to perform retrieval of models based on CAD/CAM-relevant semantics. We have previously developed methods for using design and machining features as to created graph-based part signatures. This work, however, did not develop indexing or clustering methods and part retrieval was performed with a linear search of the model dataset. Other research in this direction includes that of Wyss et al. [18], who developed techniques to compare boundary representations of polyhedral objects.

2.2 Database Indexing. Database indexing is the study of the data structures used to store collections of data in order to allow for efficient search and retrieval. The choice of indexing techniques for a database is dependent on many factors, including types and distributions of the queries expected to be performed on the database, the nature of the data, and the size of the database. On one extreme, no indexing requires the entire database to be scanned to determine whether an element is to be included in the result of a query, while at the other, a perfect index maps directly a query to the collection of items that match the query.

Spatial indexing techniques arose out of the common problem of efficiently finding closely related elements in a database. Frequently applications such as Geographic Information Systems (GIS) enable the user to perform queries to find neighboring items in a database, such as the set of music halls in a given radius of a street intersection. This class of queries primarily includes:

1. Range Queries: Given a database of models, \( D \), a query model \( m \) and a range \( R \), find the set of elements within range \( R \) of \( m \), i.e., \( \forall e \in D \):

\[
e \in \text{Result} \iff \text{distance}(m,e) \leq R
\]

\[
\text{distance}(m,e) \leq R \Rightarrow e \in \text{Result}
\]

2. K-Nearest-Neighbor Queries: Given a database of models, \( D \), a query model \( m \) and a constant \( K \), find the set of the closest \( K \) elements to \( m \). Specifically, \( \forall e,f \in D \):

\[
\text{Result} \leq K
\]

\[
(e \in \text{Result}) \land (f \in \text{Result}) \Rightarrow \text{distance}(M,e) \geq \text{distance}(M,f)
\]

2.3 Spatial Indexing of Vector Spaces. A popular spatial index used over vector spaces is the k-d Tree, parameterized by \( k \), the number of dimensions in the vector space that it indexes. This index consists of a tree-based structure of nodes with two children, \( \text{left} \) and \( \text{right} \) and a vector data element \( v \), represented as a tuple \( \langle \text{left}, \text{right}, v \rangle \). Insertion of a vector \( v \) into the tree involves traversing the tree based on comparisons of the components of \( v \) with components in the node vectors. At each level \( l \) in the tree, \( node,v[l] \) is compared to \( \hat{v}[l] \), and the decision is made to follow the \( \text{left} \) or \( \text{right} \) child of \( node \) based on whether the result of the comparison is less than or greater than. Effectively, insertion is performed as would be expected in a standard binary tree, interleaving comparisons on each dimension of the vector space.

A common alternative to the k-d Tree for spatial indexing is the R-Tree. This index breaks the elements in the database into a set of rectangles at each level in the tree. Conceptually, these rectangles are described as 2-dimensional entities, however generalized R-Trees (gR-Trees) can be constructed using \( n \)-dimensional "rectangles." When inserting a vector into the tree, it is placed into the subtree associated with that rectangle. If no such rectangle exists at a level of the tree, either a new rectangle is created or an existing rectangle is enlarged to accommodate it.

Unfortunately, spatial indexes such as \( kd \)-Trees and R-Trees typically fail to scale and to handle datasets with a high dimensionality. The central problem is that the amount of space enclosed by partitioned regions of the indexes increases exponentially as the dimensionality increases, resulting in extremely sparse datasets. As a result, empirical studies indicate that these indexes, such as the R-Tree become extremely inefficient [19].

Another criticism of these spatial index methods is that they require data to fit into a vector space. Many forms of data, such as graphs or video-streams, do not have a natural mapping to a vector space, and as a result cannot be directly handled by these spatial indexes. Despite this, the concept of range queries over such data types frequently are meaningful.

2.4 Metric-Space Indexing. Metric spaces are collections of data along with a function, \( \delta(x,y) \), known as a distance metric, which computes an effective distance between any two objects in the set. This distance function must satisfy the following conditions:

\[
\delta(x,y) = 0 \iff x = y : \text{Identity}
\]

\[
\delta(x,y) \geq 0 : \text{Positivity}
\]

\[
\delta(x,y) = \delta(y,x) : \text{Symmetry}
\]

\[
\delta(x,y) + \delta(y,z) \geq \delta(x,z) : \text{Triangle Equality}
\]

Many data types that do not fit well conceptually into a vector space are easily representable in a metric space. Graphs, for instance, do not fit well into a vector space. It is extremely difficult to construct a vector representation that preserves the property that the two vectors are equal if and only if the two graphs are equal. On the other hand, a number of distance comparisons can be performed on these graphs, such as edit-distance.

Another significant application of metric spaces are for data sets that have a human-driven interpretation of similarity. Frequently, complex functions can be constructed to approximate the human interpretation of similarity between two or more objects. Consider for example, handwriting recognition or image recognition techniques. As long as the measurement satisfies the requirements for a distance metric, or almost achieves them, such a measure can be used to index the data in a way that may be intuitive for a human user.

Given indexing techniques to efficiently handle metric spaces, the high-dimensionality problem of traditional spatial indexes can be reduced. Vector spaces, in conjunction with a distance metric such as Euclidean distance, constitute a metric space. Additionally, if metric indexing techniques make use of the assumption that distance computations may be an extremely difficult task, they should perform well even when the dimensionality of the vector space is extremely high.

An early approach to indexing metric spaces was the VP-Tree. The VP-Tree takes representative “vantage point” objects, and constructs the median sphere centered at that point, dividing the remaining data into those inside the sphere, and those outside the sphere. The asymmetry in the size of regions inside and outside the sphere lead to a disproportionate number of elements being placed in the outer half of the sphere, resulting in an unbalanced tree, requiring costly rebalancing operations. Some attempts to
improve the behavior of this technique involve the use of multiple vantage point objects on each level of the tree, and exhibit some improvement [20].

In addition to the VP-Tree, work has been done on GH-Trees (Generalized Hyperplane Tree) which choose two (or perhaps more) elements at each level of the tree, and partition the remaining elements into two sets—those closer to one element or the other. This technique has the problem that it requires two or more potentially expensive distance calculations at each level in the tree. One of the most sophisticated approaches is the GNAT tree, which introduces a number of heuristics to ensure that the tree is relatively balanced [21], although this results in a lot of computational overhead.

The VP-Tree and the GNAT tree, however, fail to perform as well when data is being inserted into the tree in a dynamic way, without costly balancing operations being performed. A structure known as the Metric Tree (M-Tree) has been developed [22], which attempts to construct the index incrementally in a manner which scales under the dynamic insertion of data. Each node in the tree consists of a model data element e, and a range r, in a tuple (e, r). The tree has the property that every element e stored under the node node satisfies distance(e, node.e) ≤ node.r. An example of the M-Tree is shown in Fig. 1. Heuristic balancing techniques can be used to ensure that the height of the tree does not grow too large in practical applications. Empirical results indicate that this technique scales fairly well even under dynamic insertions for extremely high-dimensional datasets.

2.5 Clustering in Databases. Agrawal et al. [23] describes clustering as “...task that seeks to identify homogeneous groups of objects based on the values of their attributes (dimensions)...” More formally, a cluster of a set S of data entities is a subset T ⊆ S, such that the elements in T are similar to one another than to those elements in S not in T. In a clustering, C = {T₁, T₂, T₃, ..., Tₙ}, the elements in Tᵢ are more similar to each other than the elements in Tⱼ.

Algorithms that perform these clusterings have been studied extensively in several different fields such as biology, chemistry, and sociology. There is a large taxonomy of clustering techniques that are all suited for different purposes. Two of the most widely used techniques are hierarchical and partitional.

Fasulo [24] describes a popular method of partitioning called k-clustering using the optimization algorithm. k-clustering takes a set of objects, S, and an integer k and outputs a partition of S into subsets T₁, T₂, ..., Tₖ such that objects in a cluster are more similar to each other than to objects in different clusters. According to Jain et al. [25], the general problem is known to be NP-Hard. Some heuristics in use are the k-means algorithm by MacQueen [26], k-medoid by Kaufman et al. [27], and one algorithm by Gonzalez [28] which instead of minimizing the sum of the partitions, minimizes the maximum inter-cluster distance. According to Fasulo, there are a few weaknesses in these methods. They favor spherical clusters, not general shapes and do not deal with noise. These weaknesses have been addressed by Agrawal et al. [23] and Ester et al. [29]. These all belong to a class of clustering methods that rely on iterative partitional clustering. It is from this family of clustering algorithms that our research will be continued and for which a general outline of the steps is detailed below:

1. **Start with initial partition of K clusters:** Since K is supplied as a parameter to the algorithm, we can set up an arbitrary partition of the data into clusters. Generally, better results can be obtained if the initial partition is closer to the optimal partition.

2. **Updating the Partition:** A method for doing this is to find a random cluster with an outlying point, and attempt to swap it into a new cluster if the criterion function is moved closer to the optimal partition.

3. **Compute the new representative objects for each of the clusters:** A popular method is to compute the centroid, or mean vector, of the cluster.

4. **Repeat 2 and 3 until the optimum value for the criterion function is found.**

5. **Merge small clusters and split large clusters and remove small, outlying clusters:** This is done to take care of excessively large clusters, empty clusters, and bad data points that may influence the clustering in a negative fashion.

3 Problem Statement and Technical Approach

Our goal is to store a collection of CAD/CAM models in a database and perform efficient search and retrieval of these models based on their shape and engineering properties. Additionally, we wish to cluster the collection of these models in order to extract information regarding the structure and distribution of the models in the database.

To this end, as described in Fig. 2, we first construct a mapping from the boundary representation of solid models to a graph-based data structure that we call the Model Signature Graph. We develop two alternative projections from Model Signature Graphs to vector spaces based on the semantic and structural properties in this graph, as shown in Fig. 2. In these vector spaces, created from the signatures, we perform standard metric computations of distances between vectors. Lastly, we show how to make use of these distances to perform clustering and metric-based indexing of the models in the database.

3.1 Model Signature Graphs. We make use of specialized graph structures in order to represent solid models with the intention of performing similarity comparisons. These structures, by exploiting intrinsic graph properties and domain specific properties, we develop techniques for archiving and comparing solid models.

A Model Signature Graph (MSG), is constructed from the boundary representation of a solid model in a manner similar to that by Wyś et al. [18] and the Attributed Adjacency Graph (AAG) structures used to perform Feature Recognition from solid models [30]. The primary advantage that MSGs have over the alternative formats is the increased expressiveness of the vertex- and edge-labels available in the graphs. We believe that these
labels will allow graph-based model comparisons to be more semantically relevant than those using other types of graphs.

The boundary representation (BRep) [31] essentially consists of a set of edges and a set of faces used by the solid modeler to describe the shape of the model in 3-dimensional space. An MSG for a solid model \( P \) is defined as a labeled graph, \( G=(V,E) \), where each face \( f \in P \) is represented as a vertex \( v \in V \), with attributes that further describe the qualities of the face in the model. For instance, we record the type of the surface (e.g., flat or curved), the relative size, and other physical attributes of the face with the vertex, including:

1. topological identifier for the face (planar, conical, etc.);
2. underlying geometric representation of the surface, i.e., the type of function describing the surface;
3. surface area, \( f_{\text{area}} \), for the face;
4. set of surface normals or aspects for \( f \).

An MSG edge \( \{f_1,f_2\} \in E \) exists whenever two faces \( f_1,f_2 \in P \) are adjacent—i.e., when \( f_1 \) and \( f_2 \) meet at an edge \( e \) in the BRep. The MSG edge \( \{f_1,f_2\} \) is attributed with features representing the type of BRep edge, \( e \), connection that is being used, including:

1. topological identifier for the edge \( e \) in the model \( P \);
2. concavity/convexity of \( e \);
3. underlying geometric representation of the curve of \( e \), i.e., the type of function describing the curve;
4. the length of the curve of \( e \).

MSGs are representable as undirected graphs with labeled edges and vertices. MSGs are capable of representing either individual parts or collections of parts—the connected components within the MSG each represent one of the parts making up the collection. There is no significant restriction on the attributes stored on the vertices and edges in a MSG. The selection to be used in applied systems will most likely be tailored to meet the needs of the particular application. We assume that for the most part, little information aside from the BRep is available for the construction of MSGs.

These attributes represent a substantial amount of the structure contained within common BRep models, and will enhance model comparisons by aiding in the differentiation of edges and faces in the models. In addition, the attributes are easily computed from BRep models, which make them extremely practical.

We believe that some attributes, such as the modeling features that are used to generate faces and edges in the BRep, may be very important in the matching process during practical use. Unfortunately, deriving this information from pure BRep data is an extremely difficult process, and may not be worth the gains in performance of the similarity measures. Further work must be done before we incorporate such attributes in our MSGs for comparisons.

Examples. Figure 3 shows two models and their signature graphs. The first, Fig. 3(a) illustrates a transformation from the boundary representation of the solid model for a simple bracket; Fig. 3(b) shows a motor crankcase. The complexity of the relationships between the faces and edges in the BRep of this model gives a good idea of the difficulty of performing shape similarity based on BRep data. Good practical algorithms in the research literature for determining the similarity of adjacency structures of this form are scarce. Making use of MSGs for similarity assessment proves to be an extremely challenging problem to solve.

3.2 EigenDistance-Based Similarity. We primarily focus on techniques of spectral graph theory as a basis to approximate graph similarity among model signature graphs. Spectral graph theory is the study of the adjacency matrix of graphs using linear algebra—particularly the study of their eigenvalues and how they relate to other properties of a graph.

The spectrum of a graph—the sorted eigenvalues of its adjacency matrix, holds a tremendous amount of information related to the structure and the topology of a graph. It has been used in numerous applications for the purposes of graph partitioning [32] and geometric hashing in vision recognition [33].

There are a number of different adjacency matrices that are used in spectral graph theory to generate a graph’s spectrum. Biggs [34], for instance, uses a common form:
Chung [35], proposes the use of an alternative “normalized” form of the graph adjacency matrix, which takes into consideration the degrees of the vertices in the graph to determine the entries in the matrix. This form, known as the graph Laplacian, has eigenvalues that have well-known and powerful relationships with other graph properties, such as the graph diameter. The Laplacian is formally defined as follows:

\[
A_G(u,v) = \begin{cases} 
1 & \text{if } u \text{ is adjacent to } v \\
0 & \text{otherwise.}
\end{cases}
\]

\[
L_G(u,v) = \frac{d_u d_v}{d_u + d_v} - \delta_{u,v},
\]

In addition to the relationship between the Laplacian spectrum and other graph invariants, as well as the fact that the normalization helps to reduce the effect of irregular degrees on graph vertices, we chose to make use of the Laplacian matrix for our approach.

### 3.2.1 Spectral Hashing of Graphs.
A primary goal of the distance metric is to ensure that two identical graphs have no distance between them. In addition, two graphs with only minor differences should consistently be measured relatively closely. To achieve this requirement, we first compute the eigenvalue spectrum of the graph. This operation can be done during a preprocessing phase, and the results stored for later use, if a model of reference MSG is being used in an application. From this point on, we consider this spectrum to be a projection of the graph from the space of graphs to \( R^n \), and we use this space as a basis for most distance computations that make up the metric.

Distance computations between the image of graphs in \( R^n \) are done using any one of a number of vector-based metric norms. In particular, investigating the use of \( L_1, L_2, \) and other norms.

### 3.2.2 Substructure Comparison.
A reasonable measure of similarity between graphs relies on the concept of shared substructure [36]. Semantically, two graphs that have similar common substructures themselves will tend to be similar. We believe that substructure similarity is central to most practical applications of shape recognition.

We make use of substructure information in the following manner. After computing the eigenvalue spectrum for a graph, we partition the graph into two or more subgraphs, removing the edges that cross the partition(s). If this is done wisely, it will isolate highly connected component graphs at each level of the recursion. We generate the eigenvalue spectrum for each component, and recursively partition and index again.

Optimally partitioning a graph to minimize the number (or weight) of the edges that cross the partition is itself an NP-hard problem. In order to make the problem more tractable, we make use of an approximation to partitioning using the eigenvalues that were computed for the graph. The technique, known as Spectral Graph Partitioning, puts the \( n^{th} \) vertices into one of two subgraphs based on the sign of the corresponding entry in the eigenvector corresponding to the second smallest eigenvalue (also known as the Fiedler Vector) [37–39]. Thus, with little added asymptotic complexity, the graph can be decomposed into two halves, and the process may be repeated.

### 3.3 Invariant Topology Vectors (ITVs).
The eventual aim of our work is to enable indexing and quick retrieval of a large repository of CAD models. Although a measure such as graph edit distance provides a proven metric for graph similarity, the cost of running the algorithm is too high for a database that could potentially consist of thousands of designs. Using graph invariants, we can determine groups of potentially isomorphic graphs, or graphs that are close enough in similarity to satisfy a query. Since the goal of this paper is simply to show that clustering based on invariants is a useful means to an end, a simple clustering method has been chosen to illustrate feasibility. Runtime is not an issue at this point, the primary interest being whether or not the clustering provides suitable results for further research.

A set of graph invariants can be used to tell if two graphs have the potential to be isomorphic or not, but can not be used to give a definitive answer. If the invariants of two graphs are different, there is no way the graphs can be isomorphic; if the invariants are all equal, then the graphs may or may not be isomorphic.

We encode a fixed set of invariants as an Invariant Topology Vector (ITV). ITVs will give us the ability to determine a "position" for a graph in the metric space relative to other graphs. Graphs with similar values for invariants will lie in the same region of the metric space. This gives us an approximate method for comparing solid models.

The graph invariants that we focus on are directly related to the degree sequence of a graph. The degree sequence, while not the most powerful invariant known, is easily computable, and can be exploited to recover general structural properties of the graph. We attempt to construct these invariants such that they are normalized, and do not depend on differences of scale in the number of edges or vertices. At present, we are using the following graph invariants:

1. **Vertex and Edge Count:** Trivially computable graph invariants. Unfortunately, they reveal practically no information about the graph structure, and may be weighted too strongly in the comparisons we perform, skewing our similarity metrics.

2. **Minimum and Maximum Degree:** Provide a measurement of the maximum and minimum number of adjacencies between one face and the other faces in the model.

3. **Median and Mode Degree:** Provides a measurement that represents statistically the most common and “average” cardinalities of adjacencies between faces in a model.

4. **Diameter:** The longest path of edge/face crossings between any two faces in the model.

Using graph invariants helps prune out graphs which have no similarity to a given graph and does not depend on the application of the technique. However, we would like to be able to tailor it in some fashion to a particular domain to aid in more precise measurements. Because of the extensions to graphs present in the MSG, we can define other properties that will be invariant under isomorphism of two MSGs.

1. **Type Histogram:** Each model BRep contains a set of faces and edges which have various types and features. The analysis and statistical breakdown of these component types is used to further generate a representation of the model. In the ACIS system, there are 13 surface types and 8 edge types. The count of each type is contained in the histogram.

2. **Half Space and Half Edge Count:** The number of half spaces and half edges in the model.

3. **Surface Area Ratio:** This captures the ratio between the surface area of the entire model and the surface area of the bounding box.

4. **Aspect Count:** The number of aspects that exist within the model.

In generating the MSG from the model, we put labels on the vertices and edges. We use these values in our ITV to give even finer granularity in our assessment. However, since there are varying numbers of vertices and edges in the graphs, the labels can not be used directly.

The ACIS Solid Modeler from Spatial Technologies is the modeling system used in our experiments. ACIS has 13 different surface types, and 8 different curve types. We modify the definition of the ITV to include 21 additional values. In effect, we create a histogram of the number of times particular surface and edge
types occur in the model. Furthermore, ACIS can be used to compute the surface area of the model, the surface area of the bounding box, the number of half-spaces and half-edge, and the number of aspects that exist in the model.

The ITV is a point in 33 dimensions and can be clustered using any method available to work with high-dimensional points. The benefits of conducting experiments with clustering are two-fold. The first is that a qualitative assessment of the metric for model comparison is made possible. If the metric is valid or approximately valid, similar models will ideally be grouped in the same clusters. The second is that employing these clustering methods provides the basis for the development of large scale model databases.

3.4 Using the Distance Metrics: Examples. We apply the techniques discussed to construct the ITVs and eigenvalue spectra required to perform distance computations on a simple model available in the National Design Repository [40], to demonstrate the transformations we use to perform shape recognition and matching. We will employ the solid model depicted in Fig. 2: the spinner3 model, a simple practical model found in industry with an extremely simple BRep.

ITV. First, we compute the ITV associated with the model. The following table depicts the graph invariant attributes associated with the MSG for spinner3:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Count</td>
<td>24</td>
</tr>
<tr>
<td>Edge Count</td>
<td>63</td>
</tr>
<tr>
<td>Degree Minimum</td>
<td>2</td>
</tr>
<tr>
<td>Degree Maximum</td>
<td>17</td>
</tr>
<tr>
<td>Degree Mode</td>
<td>4</td>
</tr>
<tr>
<td>Degree Median</td>
<td>4.00</td>
</tr>
<tr>
<td>Degree Std. Deviation</td>
<td>3.27</td>
</tr>
<tr>
<td>Graph Diameter</td>
<td>3</td>
</tr>
</tbody>
</table>

The computation of these attributes is relatively straightforward. The only attribute that we are currently concerned with that is not computable in time linear in the number of edges is the graph diameter. We are able to compute graph diameter in two ways. In one implementation, the diameter is computed using an all-pairs shortest-path algorithm, such as the Floyd-Warshall algorithm. The only attribute that we are currently concerned with that is not computable in time linear in the number of edges is the graph diameter. We are able to compute graph diameter in two ways. In one implementation, the diameter is computed using an all-pairs shortest-path algorithm, such as the Floyd-Warshall algorithm.

\[
D(G) = \frac{\log(n - 1)}{\log(\frac{\lambda_{n-1} + \lambda_1}{\lambda_{n-1} - \lambda_1})}
\]

where \(\lambda_n\) is the \(n\)-th largest eigenvalue in the eigenvalue spectrum, and \(n\) is the number of eigenvalues in the spectrum (which is equal to the number of vertices in the graph). We investigate the use of this as an approximation of graph diameter as if both our eigenvalue metric and the invariance metrics are being used, the computation of graph diameter is done at effectively no added cost.

Eigenvalue Metric. In order to compute the eigenvalue similarity measure we must first compute the Laplacian matrix for the spinner3 model, and then compute its eigenvalue spectrum. Both the Laplacian matrix (\(L\)), and the eigenvalue spectrum (\(S\)) computed from the spinner3 model are depicted in Fig. 4. We compute the eigenvalues of the system with software packages that use a Householder-QL matrix diagonalization algorithm, which efficiently generates the eigenvalues in time cubic in the number of vertices of the graph.

The ITV and the eigenvalue vector can be compared using standard vector norms in \(\mathbb{R}^n\). In the case of comparing graph spectra, small provisions must be made for comparing graphs with differing number of vertices, as spectrum vectors of different lengths must be compared. Currently, we simply remove the smallest eigenvalues from the larger vector, and compute standard \(L_2\) (Euclidean) and \(L_p\) norms.

We investigate the application of the measures on models found in the National Design Repository [40]. We have computed the distances between each a collection of models using both the eigenvalue spectrum similarity measure (EigenDistance), and the graph invariant topology vector measure (ITVDistance). The EigenDistance computations that we have done make use of a non-recursive version of the method, which we expect limits its ability to do substructure similarity assessment. Figures 5 and 6 depict the similarity measures between triples of models that we have selected to highlight various properties of our approaches.

Example 1. The first triplet of models, shown in Fig. 5, is made up of parts known as part01, spinner3, and spinner4. These are fairly simple industrial models, and their relationship under both distance measures is illustrated in Fig. 5. Visually, one would expect that spinner3 and spinner4 would be more similar to one another, which is contradictory to the results attained by both of our similarity metrics. More interestingly, this is consistent for both similarity measures. Our interpretation is that the measures are picking up on a similarity in the model complexity in spinner3.
and part01, given that spinner4 is a noticeably more complex part with a greater number of geometric and topological entities.

**Example 2.** The second triple, shown in Fig. 6, consists of models from the US Department of Energy’s Technologies Enabling Agile Manufacturing (TEAM) Program. The Team Part-1, Team Part-2, and 3tmount. Team Part-1 and Team Part-2 are benchmark parts made to test process planning applications, exhibit striking similarity, although their scales and orientation in the Repository are different. 3tmount is an actual model from industry, and is considerably different from either Team Part-1 or Team Part-2. As is depicted in Fig. 6, the distance measures seem to agree with visual intuition. The Team Part-1 and Team Part-2 models are considered to be far more similar than 3tmount.

Although not readily apparent in the figures, each of these parts is stored in the Design Repository [40] with differing scales, and transformations. It is essential to note that our similarity measures have no difficulty performing the comparisons between the objects based purely on their shape and structure, regardless of these transformations.

The selected comparisons illustrate the metric properties of our similarity measures. Take, for example, the comparison between Team Part-1, Team Part-2, and 3tmount. Tables 1 and 2 provide verification that the triangle inequality is obeyed by both the EigenDistance and ITVDistance similarity measures.

The central conclusion is that the “length” of the path between two models that passes through an intermediate is longer than the direct path between the models. This is a realization of the triangle inequality, and is a key property of our distance measures. It should be noted that our similarity measures only approximate metric functions, and models may be found for which the triangle inequality does not hold.

### 3.5 Clustering of Invariant Topology Vectors (ITVs)

By considering an ITV as a point in multi-dimensional space it becomes possible to group points together via the partitional clustering method described above. As of now, our clustering algorithm implements the first step of selecting a fair partition. The exact mechanism in which this works is:
1. Allocate $k$ clusters.
2. Seed clusters with $k$ ITVs. The ITVs become the representative vector.
3. For each ITV not already in a cluster, we add it to the cluster whose representative vector is closest.
4. Recalculate the representative vector for the cluster just modified.
5. Repeat Steps 3 and 4 until no more unclustered ITVs remain.

As one can see, this algorithm will not do a very good and refined job of selecting the best partitioning. However, for testing and verification purposes, it is adequate, as we will show in our data and conclusions section.

It is also worth noting that any valid metric will work for clustering models. It would be possible to cluster the graphs according to the EigenDistance value of the graphs. Therefore, it is desirable to have a framework in place whereby a different metric function could be used in the algorithm. This could be useful if a better function is desired or if the domain changes such that the comparison criteria also changes.

4 Empirical Results

4.1 Experimental Testbed: The National Design Repository. The National Design Repository (http://www.designrepository.org, http://repos.mcs.drexel.edu) is a digital library of Computer-Aided Design (CAD) models and engineering designs from a variety of domains [40,42,43]. The Repository project started in 1994 and its 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 and solid models. The Repository provides benchmark designs in a variety of formats to serve as reference data to aide developers and students.

The Design Repository currently contains over 55,000 files maintained in multiple data file formats (including STEP AP 203, ACIS .sat, DXF, IGES, DGN, Parasolid .xmt). Individual object sizes in the Repository range from a few kilobytes for simple solid models to hundreds of megabytes for complex electro-mechanical assemblies. Contributions have been made by many major research laboratories and academic institutions. Currently there are 10 gigabytes of design, engineering and manufacturing information and it is growing by approximately 20% each year.

In the following experiments, we used the entire content of the Design Repository. In this paper, we highlight some of these results by selecting a realistic and varied subset of solid models from the Repository. In selecting the models, we chose representative range of complexity—from those with simple boundary representations to more complex mechanical artifacts. While the subsequent figures and tables show only 26 models, note that these 26 are chosen from over the 44,000 models that were in the Repository at the time the experiments were performed.

4.2 EigenDistance. Figures 9 and 10 depict the distance measurements between a varied selection of models taken from the National Design Repository. Figure 9 illustrates the distances computed through the use of the distance measure described in Sec. 3.2, using a non-recursive form of the distance metric. Figure 10 reflects distances computed between the objects based on Euclidean distance between the ITV vectors of the models, described in Sec. 3.3.

The distances computed for these models using both distance measures are fairly well correlated, having a statistical correlation coefficient of about 0.70. Corresponding the distance measures to actual models, however reveals the topology-oriented nature of these similarity measures.

Figure 7 illustrates the parts STAMP and X2Part, which were considered to be extremely close by both measures. The human eye depends much more on the scale and shape of the components of the objects than our measures do. The distances we compute reflect large similarity between the parts MSG representations,
deviation away from the query model. The query part and the parts returned in the results all show strong topological similarities.

4.4 Topological Clustering Based on ITVs. To test the ITV clustering technique, we ran several tests with \( k = 1000 \ldots 20000 \) clusters. As described in 3.5 the algorithm generates a rough initial partition based on the Euclidean distance between the clusters’ representative vectors and the vectors in the clusters. To empirically test our results we selected clusters with a small total error from \( k = 1000, k = 3000, \) and \( k = 20000, \) and visually compared the models selected from different clusters in the clustering. In each clustering run, over 40,000 models were grouped based on their topological similarity. The following are three examples pulled from this huge dataset which illustrate some of the patterns that can be found in the data.

**Example 1.** Figure 13(a) shows a very simple example where the models were selected from the same cluster. The reader can verify, both models have 10 faces and are topologically similar. The main difference between the two models is that the model on the left has a protrusion from the top while the right model has a depression.

**Example 2.** Figure 13(b) shows a slightly more complicated example where \( k = 3000. \) The method used was the same as in the previous example. Notice that the clustering does not, as of yet, do a very refined job of grouping models with differing features, but does a good job of grouping together models with a similar overall shape.

**Example 3.** Figure 13(c) shows two of the more topologically complicated parts in the repository, on a clustering where \( k = 20000 \) and the total number of models is 40,000, these two were grouped together. This is significant because at 20,000 clusters, the average cluster size is roughly 2.

5 Conclusions

This paper introduced our approach for managing solid models in modern relational database management systems. Our contributions are the development of a number of novel storage and retrieval strategies that extend the state-of-the-art in database research as well as change the way in which solid modeling software developers and design and manufacturing enterprises view CAD-centric data management problems. In particular, our *Model Signature Graph* approach is an effective formalism for topological similarity assessment of solid models. We showed how MSGs can be used to build M-Tree-based indices and cluster analyses for our testbed Design Repository. Our empirical results show that our two distance measurement techniques, Invariant Topology Vectors and EigenDistance measures, can be used to create semantically meaningful metric spaces in which solid models can be compared, indexed and clustered. We believe this work will begin to bridge the solid modeling and database communities, enabling new paradigms for interrogation of CAD datasets.
Future Work. We foresee two major aspects of the EigenDistance method that need further development. Further refining the technique of recursive subdivision of the measure to aid in identifying substructure similarity is essential to shape recognition. Additionally, means of comparing eigenvalue spectra with a differing number of elements may be done in a more intelligent manner, that does not throw away information, or artificially inflate the differences between graphs with a large difference in the number of vertices.

Additional work is needed in further developing the ITV distance metric. Some possible enhancements include refinement of the attribute lists to eliminate interdependence of attributes and the possible addition of other graph invariants, such as eigenvalue spectra. Another important refinement will be determining the relative importance of certain attributes in the vector and weight them accordingly so that no one value dominates the distance function as well as optimize similarity calculations over particular collections of models.

The standard K-clustering algorithm used is not adequate for large databases, as it exhibits prohibitive run time costs. The clusters created in the process are limited to hyperspherical shapes which limit the quality of the clusters formed. Furthermore, the algorithm is highly sensitive to outlying points.

Additionally, more thorough validation techniques are needed. In our current work, the claims for the efficacy of the techniques is largely based on examples and empirical evidence from our National Design Repository. While this has allowed us to create some interesting case-studies, it does not imply a general proof. To this end, we are creating sets of models with known properties and expected groupings as well as gathering additional data from industry and government partners. We believe that this will lead to systematic, large-scale, evaluation studies that show that the approach in this paper can provide the basis for retrieval of CAD models based on more specific engineering properties.

Finally, we plan to apply our similarity metrics to databases of solid models collected from the National Design Repository to enable real-time search and retrieval of models over the web. We believe that the metrics will prove to be extremely useful in performing database indexing and clustering of the models for knowledge-discovery and data mining (KDD). Effort will therefore be expended in developing the appropriate database machinery to perform experiments with the dataset.

Acknowledgments

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, Grant ENG/DMI-9713718 and Office of Naval Research Grant N00014-01-1-0618.

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, Office of Naval Research or the other supporting government and corporate organizations.

References


