An Approach to Indexing Databases of Graphs

David McWherter, Mitch Peabody, William C. Regli, Ali Shokoufandeh

Technical Report DU-MCS-01-03
Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104
June 2001

An Approach to Indexing Databases of Graphs

David McWherter Mitch Peabody William C. Regli Ali Shokoufandeh

Geometric and Intelligent Computing Laboratory
Drexel University
Dept. of Math and Computer Science
3141 Chestnut Street
Philadelphia, PA 19104
USA
{udmcwher,umpeabod,wregli,ashokouf}@mcs.drexel.edu

Abstract

The representational power of combinatorial graphs makes them the basis of a wide variety of data structures found in a large number of domains. As time progresses and many instances of these data structures are collected, the problem of managing and understanding the data within the graphs becomes exceedingly difficult. Techniques developed to index multimedia data such as images, video, and audio are inadequate in themselves for efficiently dealing with graph data types. Developing indexing techniques that make use of semantic and structural properties of graphs is essential in order to develop digital libraries of graph-based information.

This paper presents our approach and preliminary experimental results towards a general indexing scheme for graphs. To achieve this, we employ spectral graph theory to construct a homomorphism between graphs and high-dimensional vector spaces, where a metric distance can be used to compare graphs. We use these metric distance measures to construct an M-tree data structure to efficiently index the graphs. To validate our approach, we apply this methodology to the contents of the National Design Repository, a digital library of solid and CAD models from a variety of engineering design domains.

Proceedings of the 27th VLDB Conference, Roma, Italy, 2001

1 Introduction

Common problems in modern computing involve determining how to store and represent large quantities of information. In many cases, the expressivity of simple data structures fails to be sufficient to fully describe the information within a domain. Typically these cases require the description of a set of elements and arbitrary relations between elements within the set. This data can be represented as a graph (or a set of graphs) in which the elements become vertices within the graph, and the relation is stored as edges between the vertices.

Graphs form the basis of information representation in a large number of domains. They are used to model program control and data flows in the field of software engineering, chemical structures, database relations, electrical circuits in VLSI, relations among components in Computer-Aided Design (CAD), communications between sets of people. Recently, with the advent of the Internet, graphs have been used in the analysis of the connectivity structure of the World Wide Web.

As the use of graph-based data structures expands, and the amount of information needed to be stored in graph-based formats increases, the need to store and manage large collections of graphs becomes much more important. As is found in the management of other multimedia data types (e.g.,

images, video, audio), it is not sufficient to superficially store the data for future retrieval. Database support for graphs must exploit the inherent structural properties of the graphs themselves in order to maximize the utility of queries, data storage, and indexing.

This paper presents our approach and preliminary experimental results toward a generalized indexing scheme for graph-based data structures. In particular, we develop a technique for efficiently indexing a large number of small-to-moderate-sized graphs to facilitate range and nearest-neighbor queries. Our approach first involves constructing a *graph comparison space* by projecting graphs into a variable-dimensional vector space. A metric distance function, called *EigenDistance*, is defined between graph images in the graph comparison space. This function is then used to create a spatial index of the graph comparison space using an M-Tree data structure. We describe heuristics that exploit the properties of the graph comparison space to ensure that the M-Tree operations remain efficient.

In order to validate our approach, we present the results of applying our methodology to index graphs in the domain of Computer-Aided Design (CAD) and solid modeling. Our data set is derived from the National Design Repository[1], a large digital library of over 55,000 solid models collected from industrial and academic arenas. These graphs, called *Model Signature Graphs (MSG)*, represent the structural and adjacency information in the faces of a solid object. First, we assess the quality of the *EigenDistance* measure in the pairwise comparison of solid models. Second, we evaluate the performance of the M-Tree indices with a sampling of range queries performed over the MSG database. In order to perform this evaluation, we make use of an implementation testbed based on the PostgreSQL RDBMS.

We believe that this work is of significance to both the database and engineering communities: we provide a novel approach to manage a unique media type and answer queries of practical importance to engineering design and manufacturing enterprises. Our methods address a unique space of database and data management problems that are beyond the present scope of existing multimedia and spatial database technologies. The methodology we present can be used as a basis for future work in indexing databases of CAD and engineering data, and for building databases that handle 3D models as a media type. Our ultimate goal is to enable comprehensive and flexible indexing of engineering design data and meta-data to allow engineers to execute knowledge-rich queries about device structure (e.g., shape), behavior (e.g., physical properties and performance) and function (e.g., design rationale and intent).

This paper is organized as follows: Section 2 provides a brief overview of the relevant background literature from multimedia database, graph database, engineering design and modeling communities. Section 3 introduces our technical approach for assessing graph similarity using metric distance functions, and indexing graphs based on these distances. Section 4 presents some of our experimental results with the National Design Repository. Lastly, Section 5 gives our conclusions and discusses our research contributions and areas for future work.

2 Background and Related Work

Multimedia Databases. In the past decade, there has been extremely rapid growth in the availability and quantity of digitized multimedia and audio/visual content. In order to manage and provide access to this material, the database community has been developing techniques to make use of the semantic and internal structure inherent to popular data formats. For instance, a wide range of techniques have been developed to enable efficient search and retrieval of 2D photo-realistic images, real-time video, map data in GIS applications, audio, and other conventional multimedia data types. Multimedia database systems typically exploit regularities inherent in the data type, as well as the expected access characteristics of the model.

In the 2D shape matching and image retrieval literature, the indexing and shape matching process usually follows one of four common approaches: (1) textual query, based on keywords stored for each image in the database; (2) query by example, which uses similarity measures derived off of a set of query images provided as input; (3) query by sketch, which looks for image segments matching the sketched profile; (4) iconic queries, which use templates representing critical aspects of the desired image to identify images with similar features. Methods (2-4) employ image processing and computer vision techniques to identify relevant features (e.g., sunrises, people, trees, etc) in 2D images (i.e., GIF, JPEG, etc).

Graph Databases. Unfortunately, development of techniques to efficiently handle graph-based data elements has not proceeded as has those for dealing with other multimedia data types. One technique to handle graph-based information in a database

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

environment is to employ a conventional relational database system.

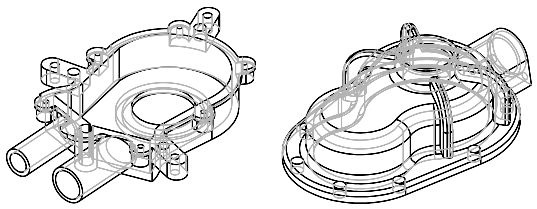
Using a RDBMS, the vertices of a graph are mapped to a set of unique entities (integers, for instance) in the database, and the edges are stored as tuples of these entities in a relation table. While all of the information about a graph may be stored within a database in this fashion, it becomes extremely difficult to extract data from the database. One problem with this approach is that it becomes extremely difficult to describe queries that rely on structural relationships between entities in the graph using typical query languages such as SQL. Another problem is that each graph structure isn't an independent element within the database. Constructing an index in order to optimize queries that involve comparing graphs and returning a collection of graphs from the database is extremely difficult.

A common approach to representing graph information within databases is to construct databases which deal explicitly with graph (also known as network) data. This solution, taken by such projects as PROGRESS/GRAS [17], makes use of a data model whose elements consist exclusively of graph vertices and relations the edges that connect them. Queries for the system involve finding substructures within the graph which satisfy specific connectivity constraints. In addition, to simplify the query description, queries may make use of graph rewriting rules to modify the graph to simplify the query description. These rules consist of a pair of two graphs, called the left-hand side and right-hand side. Evaluation of the rules involves replacing instances of the left-hand side within the graph with the right-hand side. rules involves replacing substructures in the graph with other structures.

These solutions also suffer from a number of problems. First, the introduction of a new database paradigm that is foreign to database users makes wide adoption and widespread use of the system unlikely. Another obstacle for these systems is that they rely on the computation of a large number of subgraph matching operations during the execution of a query. Not only is the primary query operation is based on subgraph matching, the application of graph rewriting rules involves subgraph matching as well. The detection of a subgraph within a larger graph is a fundamentally difficult problem, known to be NP-complete (the $CLIQUE$ problem reduces to it). As a result, use of these systems may be computationally intractable when used with non-trivial graphs.

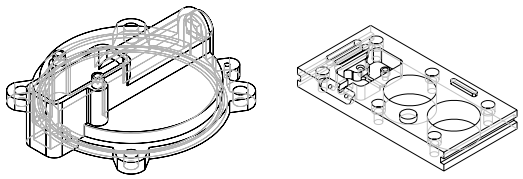
Given the computational complexity of using graph structures as a basis for a database storage and query system, it would seem as if relying on them directly would be impractical. A fundamental assumption of graph database projects is that regardless of the potential cost associated with graph operations, the systems make up for this cost in the representational power of graph structures.

An approach to deal with the problem of providing a smooth transition for traditional database systems to support graph structures is GraphDB by Güting et al. [8] The goal of the GraphDB is to develop techniques to support modeling and querying of network information. In the GraphDB model, entities in the database are represented as simple objects, links between objects, and sequences of links describing paths through the graph. Syntactically, queries are described within extensions to the common SQL *select ... from ... where* grammar. Conceptually, the execution of a query involves a sequence of query operations. These operations are capable of deriving new relations from simpler relations, rewriting operations, and selecting subgraphs of the graph.



(a) Machined Pump Cover (155KB)

(b) Cast Pump Housing (242KB)



(c) Machined Chamber Cover (153KB)

(d) DoE Team Benchmark Part (89KB)

Figure 1: Common Examples from the National Design Repository: 3D solid models of machined parts (some shown as wire-frames to reveal hidden topological details) as well as mechanical assemblies.

Databases for Solid Modeling and CAD. This work formulates a general mechanism efficiently handling large collections of graph data structures.

To validate the strength of the data model for representation and to generate experimental results, we focus on the application of graph databases to Computer-Aided Design and Solid Modeling data.

Solid models are geometrically and topologically precise descriptions of 3D shapes. Early work on solid modeling was driven by the design and manufacturing industry—hence, solid models are often used to describe the shape of machinable parts, such as engine blocks, gears, plastic casings for consumer electronics, etc. Figure 1 provides an example of solid models found in industrial applications.

We primarily focus on solid models described as Boundary Representations (BReps) [11]. BReps describe a model by specifying the individual surfaces found in a model, along with the edges via which they connect. For example, a cube would be represented as 6 planar surfaces for each side of the cube. The BRep will also specify the edges along which the faces touch, each face of the cube sharing an edge with its four neighbors. BReps have an advantage over other common model representation formats (such as Constructive Solid Geometry (CSG)) in that they create a unique and unambiguous representation of the exact shape of an artifact. Additionally, they have become the dominant model representation format used in engineering analysis, simulation, collision detection, animation, and manufacturing planning.

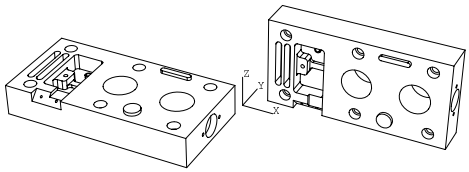


Figure 2: With origin coordinate axes shown in the center, two identical solid model BReps with differing locations and orientations. Determining that these two models exactly match is computationally intensive and prone to numeric errors.

Comparing solid models by simply comparing the contents of their BRep or other model representation may lead to problems. Transformations done to a model (scale, rotation, translation, etc) may change the appearance of the model as stored in a BRep. When comparing solid models, however, it is desirable for comparisons to be transformation invariant (semantically, a cup standing upright is identical to that cup lying on its side). As a result, determining whether two models are identical requires searching the complete configuration space of rotations, translations, and other transformations of the models. Figure 2 illustrates how a change in rotation and translation may make comparing two solid models difficult. Making the problem more difficult, floating point roundoff errors may accumulate in transformations that slightly deform the relationships and sizes of components within a model. These problems make comparing models directly by their descriptions infeasible.

Graphs can be used to represent the BRep of a solid model in a manner which is more easily dealt with than the BRep itself. One approach, used by Wysk et al. [20] is to map the faces of a BRep as vertices in a graph, and the edges that connect faces as edges within the graph. In addition, they develop a similarity function that evaluates the difference between two models based on their graph representations.

The approach taken by Wysk et al. [20] has a number of limitations that need to be addressed. First, the method applies only to polyhedral models, which limits its applicability. Second, the similarity measure that is constructed is unfit for use in a database system. The primary difficulty with the measure is that it is not symmetric. The similarity measured between solid models A and B is not the same as measuring the similarity between B and A. This results in undesirable properties when constructing an index over a collection of models using the similarity measure. Finally, this method fails to incorporate (or reflect) manufacturing considerations, such as approachability, fixturing, and operation interference, and we do not see any obvious way to add them.

Another approach to representing models as graphs is using Aspect graphs [15]. This approach computes all possible appearances that an object may take when viewed by an observer. Commonly, this is done by computing the image of a projection operator that maps the model from 3D to 2D space. The image of this operator consists of a set of differing views of the graph, and are represented in the graph as vertices. An edge is introduced between two vertices in the graph whenever a simple spatial transformation (ie: a rotation or translation) may directly transform one view into another.

Unfortunately, aspect graphs are extremely difficult to compute for arbitrary solid models, and are typically used only for objects whose boundaries can be described by simple polygonal structures, or surfaces of revolution. Practical models are much more complicated, having combinations of both smooth and polygonal surfaces. Known techniques for computing aspect graphs for these more complicated

models are computationally intractable [15].

We develop a custom graph representation for BRep models, that build on the work of Wysk et al. [20]. Our model includes vertex and edge labels in the graph that contain additional feature information. Much like in conventional image retrieval and indexing, however, there is no universally acceptable set of features on which model comparisons can be based. Evaluation measures for solid model feature attributes will depend on the application intent of the engineers using the database. For instance, process engineers may need to query based on manufacturing process features but an industrial designer may need to consider shape aspects of an object. The capricious and ill-defined semantics of the engineering design and manufacturing domain require that we create a more flexible methodology for model comparisons which can be customized to consider the criteria of required by different end users. It is our goal that our representation will enable a wide range of similarity measures which can be used when necessary.

Spatial Indices. Spatial indexing techniques are designed to cluster objects close to one another in a vector or metric space on nearby database pages, in order to improve the query processing time of range and nearest- neighbor queries. Conventional spatial indices such as *kd*-Trees and R-Trees, however, fail to scale to handle datasets that exhibit an extremely high dimensionality [14]. The central problem is that the amount of space enclosed by the partitioned regions of space by these indices increases exponentially with the dimensionality of the dataset. As a result, the index structures typically become extremely sparse, and the cost of maintaining the index dominates the query time.

Another criticism of conventional spatial indices is that they require the data to fit in a vector space. Many forms of data, such as video-streams or graphs don't have natural mappings to vector spaces. As a result, the data cannot be directly handled by these spatial indices.

Several effective indexing techniques exist to handle large metric spaces and high-dimensional vector spaces. Most of these techniques store data in a recursive tree-like structure based on distances computed between objects. Most of these systems attempt to minimize the number of distance computations that are done when executing a range query, or when trying to locate a given object. In general, each of these techniques achieve nearly logarithmic improvements in storage space, depending on how balanced the tree is.

The VP-Tree is an early approach to metric space indices. Each node in a VP-Tree contains a set of elements. A representative element (a "vantage-point") is chosen, and a sphere is constructed around it of the appropriate size to divide the elements into two evenly sized sets—those within the sphere, and those outside the sphere. While this constructs a balanced tree structure, the effects of dynamic insertion and deletion operations on the tree cause difficulties in maintaining the balanced nature of the tree. Without performing costly balancing operations, the tree suffers from the problem that the space is broken into asymmetric pieces—the area outside of each sphere is typically much larger than the area enclosed within. There has been some work done to improve the behavior of this scheme making use of multiple vantage point objects [2].

Another metric tree structure known as the GNAT [3] makes use of "generalized hyperplanes" to partition the space. At each node in the tree, two or more representative elements are chosen. The remaining elements are put into subtrees corresponding to each of the representatives, depending on which they are closest to. Each node therefore breaks up the space of elements into Dirichlet domains (Voronoi diagrams in planar spaces). Unfortunately maintaining a balanced GNAT can lead to poor performance.

Given that we expect that a practical database of solid models to undergo nearly consistent changes as models are added, removed, and updated, we require that an index structure perform well under dynamic insertions and deletions. We have chosen to make use of an indexing technique known as the Metric Tree (M-Tree) [5]. This structure has been developed to provide acceptable performance even in the face of standard updates to the tree's structure. As illustrated in Figure 6, each node in the M-Tree consists of a model data element e, and a range r in a tuple (e, r). The tree maintains the property that every element $\hat{e}$ stored under the node nd satisfies $distance(\hat{e}, nd.e) \leq nd.r$. Heuristic balancing techniques can be applied to ensure that the tree does not grow too deep in practical applications.

Metric Spaces. A **metric space** is a collection of objects along with a distance function, $\delta(x, y)$, known as the *metric*, which computes a distance between any two elements in the set. The only requirement of the distance function $\delta(x, y)$ is that it must satisfy the following conditions:

$$\begin{aligned} \delta(x,y) = 0 \iff x = y &: \text{Identity} \\ \delta(x,y) \geq 0 &: \text{Positivity} \\ \delta(x,y) = \delta(y,x) &: \text{Symmetry} \\ \delta(x,y) + \delta(y,z) \geq \delta(x,z) &: \text{Triangle Equality} \end{aligned}$$

It is likely that many acceptable distance metrics exist, leading to a number of different metric spaces that are able to be built over the same database of graphs. We believe that in most practical applications, the use of more than one distance metric will be beneficial, corresponding to different views of the dataset and depending on the particular questions being sought by the database user.

The constraints provided by the distance metric provide a significant amount of structure that can be exploited in a collection of objects for the purpose of organization. Recently, a large amount of research has been put forth to store data in metric spaces using various tree-structures [3, 5, 7] for efficient search and retrieval. These indexing techniques are tailored for efficient handling of spatial queries such as nearest-neighbor searches in high-dimensional vector spaces, or for datasets which do not conform to a vector space.

In addition, clustering and knowledge-discovery techniques such as k-means or k-median regularly make use of metric distance functions to organize and examine statistical distributions of data. Metric or near-metric distance functions for graphs therefore have the potential for use in a wide variety of applications. This property suggests that further research should be conducted to develop more powerful and meaningful metrics.

We believe that a number of metric distance functions can be used to perform similarity assessment of the structure and semantics of two or more graph objects. These functions constitute a "plug-in" module that can be tailored to perform a wide range of similarity measures based on the nature of the application. Primarily, we currently focus on similarity measures based on topological and structural properties of the graphs.

In order to make use of distance metrics in a practical system, they must be relatively easy to compute. The algorithmic complexity in computing distance metrics for arbitrarily-formed graph structures is unfortunately an open question. The computation of a distance metric between two graph data structures is Turing-reducible to the graph-isomorphism problem. That is, to decide whether two graphs are identical under some permutation of the vertices. Two graphs are isomorphic if and only if the two graphs have a metric distance of zero. The asymptotic computing time for an algorithm to compute a graph-based distance metric will therefore be related to that for an algorithm for computing graph isomorphism.

Graph isomorphism is a problem that has been studied for decades due to its wide range of applications in computer science and other fields of research. Despite the large amount of energy that has been expended on the problem, no algorithms have been developed that have reduced worst-case running time below exponential. It is still unknown whether the problem is even NP-hard, or whether polynomially-bounded algorithms are possible.

As a result of the difficulties involved in understanding and constructing efficient solutions to the graph isomorphism/graph distance metric problem dictates that practical approximation algorithms must be used to index a graph-based data structure. We investigate two approximate distance metrics over a collection of graphs: $VertexDistance$ and $EigenDistance$. These distance measures were constructed in order to approximate the behavior of a proper distance metric over the space of graphs that we plan to index.

While our distance measures are only approximate distance metrics, they appear to perform well in practice. In particular, they fail to ensure that $\delta(x,y) = 0 \implies x = y$ — examples can be constructed in which the distances between two objects is zero even though the objects are not identical. We have not yet found a pair of real-world graphs that exhibit this property.

3 Technical Approach

3.1 Problem Formulation

We define a graph G to be a tuple of sets (V, E). V is a set of elements called the vertices of the graph. E is a set of elements of $V \times V$, and add the further restriction that the graph be undirected — that $(v_1, v_2) \in E \implies (v_2, v_1) \in E$.

An isomorphism between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a bijection $\phi : V_1 \to V_2$ such that $(u, v) \in E_1 \implies (\phi(u), \phi(v) \in E_2)$. In other words, ϕ is a mapping between the vertex sets of G_1 and G_2 that preserves the edge relation. For notational simplicity, we may refer to an isomorphism between two graphs $\phi : G_1 \to G_2$ such that $\phi(G) = (\phi(V), E)$. It should be noted that an isomorphism between two graphs may or may not exist. If there exists an isomorphism between two graphs, we say that they are isomorphic, and describe this

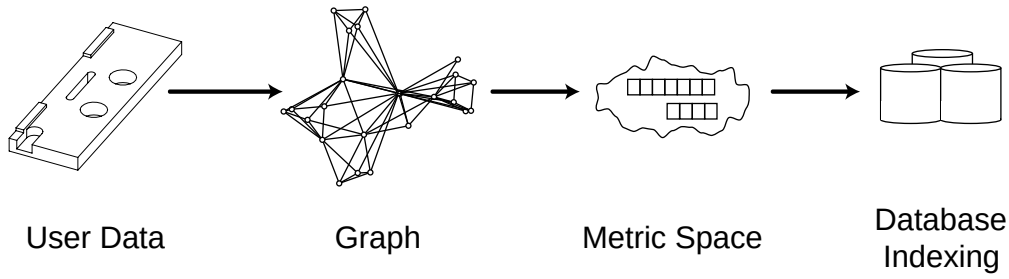


Figure 3: Overview of our approach for indexing graph databases: A collection of information (e.g., CAD data) is transformed into a graph (e.g., MSG) over which a metric space is constructed for database indexing.

notationally as $G_1 \sim G_2$. Furthermore, in our discussion the concept of graph equality and graph isomorphisms are equivalent – $G_1 \sim G_2 \implies G_1 = G_2$ and $G_1 \not\sim G_2 \implies G_1 \neq G_2$ unless otherwise noted.

Formally, a range query on a set of graphs is defined as a tuple (g^*, f, r). g^* is a graph, called the query graph. f is a distance function, which maps two graphs into a non-negative real number $(f : (G \times G) \rightarrow \Re^+)$. r is a non-negative real number $(r \in \Re^+)$, representing a range around the query graph. The results of executing a range query on a set of graphs Γ is the set of elements $\gamma \subset \Gamma$ such that:

$$\forall g \in \gamma : f(g^*, g) \leq r$$

A nearest-neighbor query is defined as a tuple (g^*, f, k). g^* and f are as before, and k is a non-negative integer indicating the number of neighbors to be returned by the query. The results of executing a range query on a set of graphs Γ is the set of elements $\gamma \subset \Gamma$ such that:

$$\forall g \in (\Gamma \setminus \gamma), \forall \hat{g} \in \gamma : f(\hat{g}, g^*) \leq f(g, g^*)$$

$(\Gamma \setminus \gamma)$ is the set of graphs in Γ, not in the set γ. In other words, the results of the query are the k closest neighbors to g^* in the set Γ.

Our goal is to provide a means through which a large database of graphs can be automatically maintained and queried, making use of the structural and semantic information in the graphs. In particular, we focus on the efficient execution of range and nearest-neighbor queries using solely the semantic and structural information contained within the graphs.

Our approach, depicted in Figure 3, consists of constructing a homomorphic projection from graphs into variable-dimensional vectors. We then use a set of metric distance (or inverse similarity) functions between the images of graphs under this mapping as a metric distance function between two graphs. Finally, we use spatial indexing techniques for metric spaces to construct an M-Tree data structure over the set of graphs.

3.1.1 VertexDistance

The first of our distance measures is extremely simplistic. The $VertexDistance$ between two solid models is defined as the difference in the number of faces of each model. This corresponds to the difference in the number of vertices in the graphs. Intuitively, the measure gives a rough estimate of the difference in complexity of two graphs—graphs with more vertices can hold much more information than those with fewer vertices. The complexity in computing this distance measure is trivial; it is linear in the number of vertices in the graph.

Given that $VertexDistance$ provides little or no structural information about the edge-structure of the graph, we suspect that this measure will perform poorly in providing human users with satisfying measures of similarity between graphs. We intend that our other distance measure, $EigenDistance$, to be superior in terms of meeting human expectations, as well as effectively indexing the space.

3.1.2 EigenDistance

We have developed another graph comparison technique rooted in the field of *spectral graph theory*. Spectral graph theory is the study of the adjacency matrices for graphs using techniques from linear algebra. In particular, the eigenvalues (characteristic values) corresponding to the adjacency matrix are examined, and correlated to other properties and structure of a graph.

The sorted eigenvalues for the adjacency matrix of a graph is referred to as the graph spectrum:

$$\text{Spectrum} = \{\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n\}$$

$$\mathcal{L} = \begin{bmatrix}
1 & 0 & -.2 & -.1 & -.1 & -.1 & -.1 & 0 & -.1 & 0 & -.1 & 0 & -.1 & -.1 & -.1 & -.1 & -.1 & 0 & -.1 & -.1 & -.1 & -.1 & 0 & -.2 \\
0 & 1 & 0 & 0 & -.2 & -.2 & -.2 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 0 & 0 & 0 & 0 & 0 \\
-.1 & -.2 & 0 & 0 & 1 & -.2 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & -.2 & 0 & 0 & -.2 & 1 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & -.2 & 0 & 0 & 0 & -.2 & 1 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -.2 & -.2 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 \\
-.1 & -.2 & 0 & -.2 & -.2 & 0 & -.2 & -.2 & 1 & -.2 & -.2 & 0 & 0 & 0 & 0 & -.1 & 0 & -.1 & 0 & 0 & 0 & 0 & -.2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 1 & -.2 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 1 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -.2 & -.2 & -.2 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 & -.2 & -.2 & -.2 & 1 & -.2 & 0 & -.1 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 1 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 & -.2 & 1 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.1 & 0 & 0 & -.2 & -.1 & 0 & -.2 & 1 & -.2 & -.1 & 0 & 0 & 0 & 0 & -.2 & 0 \\
-.1 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 1 & -.2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -.2 & -.2 & 0 & 0 & 0 & 0 & -.1 & 0 & 0 & 0 & 0 & 0 & 0 & -.1 & -.2 & 1 & -.2 & -.2 & -.2 & -.2 & 0 & -.2 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 1 & -.2 & 0 & -.2 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 1 & -.2 & 0 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 & -.2 & 1 & -.2 & 0 & 0 \\
-.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 0 & -.2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & -.2 & 0 & 0 & 0 & -.2 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
-.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.2 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

$$Spectrum = \begin{matrix}[0.30.30.50.60.80.91.01.01.01.01.01.0\ldots \\ \ldots 1.00.01.11.21.31.31.31.41.41.51.51.6]\end{matrix}$$

Figure 4: $\mathcal{L}$ is the Laplacian matric associated with the MSG for the *spinner3* model. *Spectrum* is the set of eigenvalues associated with $\mathcal{L}$.

The graph spectrum retains a tremendous amount of information related to the structure and topology of a graph, and is one of the strongest known polynomial-time computable graph invariants. Properties of the eigenvalues and the eigenvectors of a graph have been used to develop graph partitioning techniques that minimize edge-cuts [10], graph isomorphism tests, geometric hashing in vision recognition [19], and other applications. Numerous relations are now known relating components of the spectrum to graph properties such as the graph diameter or volume [4].

A variety of forms of the adjacency matrices are used in differing areas of spectral graph theory. Each of these forms lead to different relationships between the eigenvalue spectrum and graph properties, so care must be taken to ensure that the appropriate form is used. Biggs [1], for instance, bases his work on a common form of the adjacency matrix:

$$A_G(u,v) = \left\{ \begin{array}{rcl} 1 & : & \text{if } u \text{ is adjacent to } v \\ 0 & : & \text{otherwise.} \end{array} \right\}$$

Chung [4], proposes the use of an alternative "normalized" form of the graph adjacency matrix for spectral decomposition. This form takes into consideration the degrees of the vertices in the graph when determining the entries in the matrix. This form, known as the graph *Laplacian* helps eliminate scaling factors introduced in the eigenvalues as a result of unevenly distributed edges in a graph. All of the eigenvalues lie between 0.0 and 2.0, regardless of the size or number of edges in the graph. In addition, the Laplacian introduces an attractive relationship between the combinatorial nature of graphs and continuous mathematics, readers are referred to Chung's work for further information [4]. The definition of the Laplacian is formally defined as follows:

$$\mathcal{L}_G(u,v) = \left\{ \begin{array}{rcl} 1 & : & \text{if } u = v \text{ and } d_v \neq 0, \\ \frac{-1}{\sqrt{d_u d_v}} & : & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & : & \text{otherwise.} \end{array} \right\}$$

As a result of its normalizing properties, we primarily focus on the use of eigenvalue spectra derived from the Laplacian matrix of graphs to compute the distance between two graphs. The Laplacian is a symmetric, real-valued matrix which implies that all of its eigenvalues are positive and real-valued as well.

Currently, we compute all of the eigenvalues of the adjacency matrix using a Householder-QL matrix diagonalization algorithm, which efficiently generates the eigenvalues in time cubic in the number of vertices in the graph.

Given the eigenvalue spectra for two graphs, we can compare them for similarity using conventional distance metrics for vector spaces. The only caveat is that the number of eigenvalues generated will depend on the number of vertices within the graph. In order to compare graphs with differing number of vertices, a system must be developed for comparing sets of eigenvalues of different sizes. We have been investigating a number of techniques for dealing with this problem. In one case we simply throw away the largest elements of the longer vector, based on the assumption that the smallest eigenvalues are

more important in shape matching. This judgment is based on drawing an analogy to the computation of the modes of vibration in a spring system using eigenvalue computations. The smallest eigenvalues correspond to the lowest-frequency modes of vibration in the system, which tend to subdivide the nodes into larger subsets. In addition, the smallest eigenvalues have a correlation to a number of graph-partitioning techniques that minimize edge-crossings between partitions. In the future, we refer to this distance measure as *EigenDistance Truncate*

Throwing away the information contained in the eigenvalues, however, is an undesirable prospect. It is not hard to see that this has the potential to affect the triangle-inequality property of the distance function — that $\delta(x,z)$ becomes less closely related to $\delta(x,y)+\delta(y,z)$. As a result, the metric properties of the distance measure may be eliminated.

We have also been investigating ways to pad the shorter set of eigenvalues to make them longer. We have experimented with padding the set with a set of constant eigenvalues, of either 0.0, 1.0, and 2.0. When the set is padded with excessive zeros, the largest elements in the longer, leading to an inflated distance computation. Consider the Euclidean distance between the vectors S_1 and $Pad(S_2, x)$, for some positive integers n, k such that $k < n$:

$$S_1 = [\lambda_{1,1}, \lambda_{1,2} \ldots \lambda_{1,n}]$$

$$S_2 = [\lambda_{2,1}, \lambda_{2,2} \ldots \lambda_{2,n-k}]$$

$$Pad(S_2, x) = [\lambda_{2,1}, \lambda_{2,2} \ldots \lambda_{2,n-k}, x \ldots x]$$

The Euclidean distance of the eigenvalue vectors when padding with x then becomes:

$$\delta(S_1, S_2) = \sqrt{(\sum_{i=1}^{n-k}(S_1[i] - S_2[i])^2 + \sum_{i=n-k}^{n}(S_1[i] - x)^2)}$$

We have investigated using another padding technique, which limits the growth of the distance measure. In this case, we pad the eigenvalue vectors with 2.0 instead of 0.0. Given that the eigenvalues are bounded from above by 2.0 and the largest eigenvalues are more likely to be closer to 2.0 than they are to 0.0 it would seem as if this technique would make more intuitive sense. In the future, we refer to these measures as *EigenDistance Pad*(x) when padding eigenvalue vectors with the constant value x.

Given that the number of eigenvalues in a graph spectrum is equal to the number of vertices in the graph, the size of the vectors being compared is potentially very large. The graphs generated in our dataset, from the models of the National Design Repository vary from less to 6 vertices to nearly two thousand vertices. While the dimensionality of the vector space generated by the padded graph spectra depends on the actual data stored within the database, it can easily become much too large for conventional spatial database indexing techniques to handle well. Fortunately, at least in the computation of individual distance computations, the computation time is dominated only by the size of the spectra being compared (and not by the size of the largest spectrum in the database).

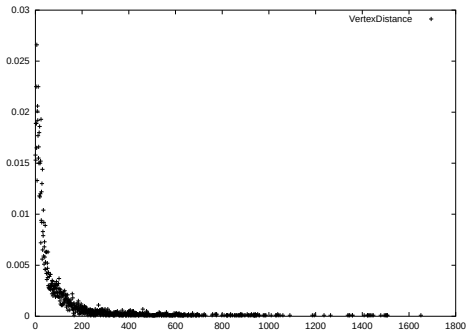
Given that it is a possibility that a database of graphs will be dominated by graphs with an extremely large number of vertices, we investigate another spectrum comparison technique. This technique is based on technique used by Shokoufandeh et al. [19] that uses sums of graph eigenvalues in the computation of hashes for graph structures. We investigate *EigenDistance Sum* that compares two graph spectra based on comparing the sums of their entries. This has the advantage in that the sum is easily precomputed and its storage requirements are much smaller than needed to store the entire set of eigenvalues, particularly for large graphs.

We intend to further refine the *EigenDistance* metric to make more use of more of the information provided in the graphs to perform distance measures. First, we intend to partition graphs into smaller subgraphs, that have semantic or structural cohesiveness. After recursively partitioning the graphs, the subgraphs' eigenvalues can be compared to extract a more finely-grained comparison of the graphs. In addition, we intend to develop comparison techniques that exploit more information within graphs, particularly through the use of vertex and edge labels. It is typical, for instance, that graph isomorphism algorithms use such labels with great success for quickly determining the existence of isomorphisms.

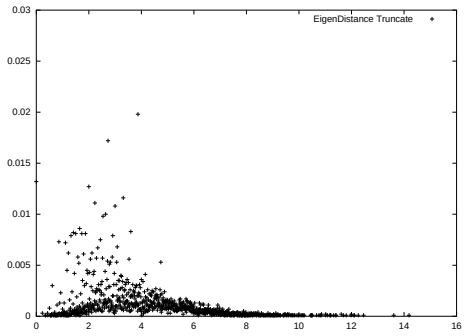
Eigenvalue Spectrum Example. Figure 4 depicts the Laplacian matrix generated from the Model Signature Graph for the *spinner3* model (see Figure 7). Note that it is a real-valued symmetric matrix, and all of the eigenvalues are real and positive.

3.2 Indexing Mechanism

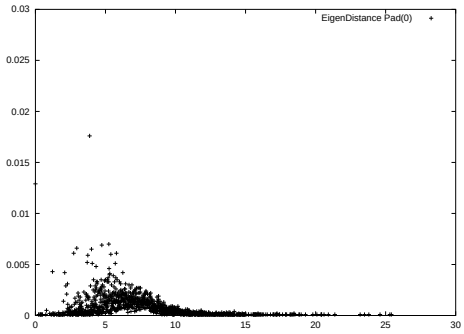
We implement our M-Tree indexing structure in the freely available PostgreSQL RDBMS. Our imple-



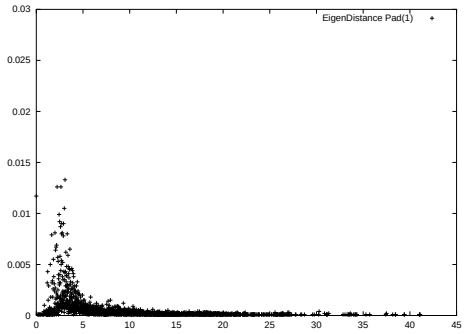
(a) $VertexDistance$



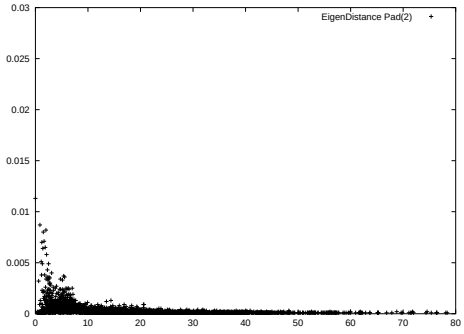
(b) $EigenDistance\ Truncate$



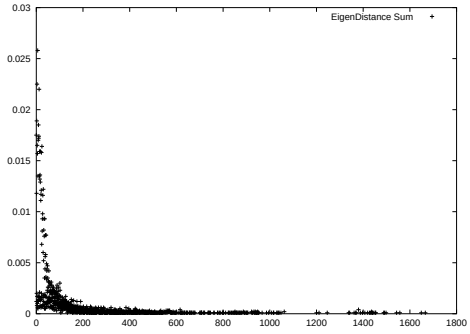
(c) $EigenDistance\ Pad(0.0)$



(d) $EigenDistance\ Pad(1.0)$



(e) $EigenDistance\ Pad(2.0)$



(f) $EigenDistance\ Sum$

Figure 5: Probability distribution of distances within the spaces defined by the $VertexDistance$ and $EigenDistance$ measures over a sampling of graphs derived from the National Design Repository.

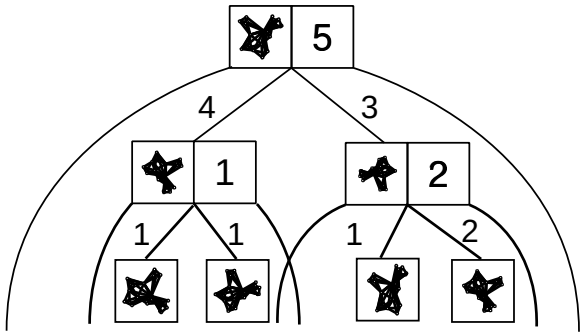


Figure 6: A pictorial depiction of an M-Tree data structure for a database of graphs. The root of the tree contains a graph and indicates that all graphs stored underneath it are closer than 5 units from that graph. The edges from the root node to its children are labeled with the distance their graphs are from the root node graph. Likewise for the remaining nodes.

mentation makes use of the Generalized Search Trees (GiST) [9] indexing API provided in PostgreSQL. The GiST architecture consists of a generalized approach to database indexing, in which new indices are constructed by specializing a set of operators:

1. `Consistent(element,tree)`—Return false if `element` cannot be found under `tree`.
2. `Union(elements[])`—Construct and return a predicate that can be used to describe a set of elements.
3. `Compress(element)`—Return a representation of an element in the index (may be lossy).
4. `Uncompress(element)`—Given a compressed representation, reconstruct the database element.
5. `Penalty(element,tree)`—Return a cost of how bad it would be to insert `element` under the `tree`.
6. `PickSplit(elements[])`—Split a set of elements into two sets.
7. `Same(pred1,pred2)`—Return true if the two predicates are the same, false otherwise.

The representation that we use in our implementation is conceptually similar to that done by Ciaccia et al. [5]. Query predicates are of the form: $\{(G, r) | r \in \Re^+)\}$, and consist of a model graph G and a positive-valued range r.

Unlike many approaches to storing graphs in databases, we chose to implement graphs as database "BLOBS"—opaque data structures for which the database system has special operators to compare and inspect, instead of storing the graphs as relations in the database. We believe that the use of distance functions between graphs will provide enough information to enable the database system to adequately manage the graphs. To this end, we have constructed a system through which PostgreSQL can handle graphs stored in a popular graph file format (LEDA [13]) as first-rate database objects such as integers or strings.

The `Consistent(`G`,tree)` operator returns whether the distance between the graph G and the graph in the predicate describing the children of the `tree` is less than the range specified in the predicate.

The `Union(elements[])` operator constructs a predicate to describe the set of elements by heuristically choosing an element from the set (it may be either a predicate, or a graph element itself), promotes it to a routing object. The range is computed to be large enough to enclose all of the elements in the set. This involves computing the maximum of the distances of the routing object to the graphs in the set and of the sum of the distances to the graphs summed with the ranges contained within the predicates. The heuristics that we make use of are discussed later.

The `Compress(element)` operator leaves predicates unchanged, and converts leaf graphs into predicates themselves with zero ranges. The `Decompress(element)` operator does nothing to the element.

We have been experimenting with different `Penalty(element,tree)` operators. Our current implementation computes the distance between the graph in the element, and the graph in the predicate describing the tree. The heuristic is then defined to be the distance when the distance is less than the range in the predicate, and the distance summed with the distance from the element graph to the edge of the range of the predicate.

The `PickSplit(elements[])` operator is trivially defined. It merely divides the set of elements into two sets, without any extensive computation. In the future, more work must be spent on deciding optimal splitting techniques to ensure a more efficient tree. The `Same(pred1,pred2)` operator is defined when the graph and the range in the predicate are identical. Given that our implementation uses references to graph objects in the index, we compare only to see if the references are the same, and never perform a full graph isomorphism test between the

graphs, making this test constant time.

One of the central differences are that while the current available implementation is only for the GiST database simulation package, rather than a practical database for engineering work. The differences between the GiST API's provided in each of these packages are somewhat significant, and the formulation of the M-Tree index must be done slightly differently. Our formulation of the M-Tree in the PostgreSQL GiST environment makes the reuse of the distance calculations slightly more difficult than it is in the implementation provided by Ciaccia. Our current implementation simply recalculates distances as needed, without reusing previous distance results. Proper reuse of distance calculations would lead to a significant savings in computing time, especially when the tree contains large and complex models.

In addition, our implementation makes use of a number of heuristics for eigenvalue spectrum-based indexing to improve its performance, primarily in the `Union` GiST operator.

First, in order to compare two graphs G_1 and G_2, the eigenvalues corresponding to the graphs must first be computed, and compared. If the eigenvalue computation is being done on the fly when comparisons are needed (as they are, in our implementation), computing time on the order of $O(|V[G_1]|^3 + |V[G_2]|^3)$ is required, assuming that $|V[G]|$ is the number of vertices in the graph. In addition, regardless as to whether the computation is done on- or off-line, $O(min(|V[G_1]|, |V[G_2]|))$ computing time is required to perform the eigenvalue-vector comparison. Each node in the M-Tree contains a representative "routing node," and every search passing through this node will need to have its distance from the routing node computed. As a result, we intelligently select routing objects such that the number of vertices in the routing graph is minimized. Although we currently do not precompute and store eigenvalue computations for the graphs, we have found that eigenvalue computations begin to dominate the indexing and retrieval times when the graphs are significantly large. We are currently investigating a means to precompute graph eigenvalues and store them in the index.

A second heuristic that we make use of is done with respect to the size of the ranges in the predicates. We make the assumption that predicates with excessively large ranges make good candidates to insert objects below, as they will be more likely already be large enough in order to contain the new object and objects inserted under it. After first sorting the candidates for promotion to routing objects in the `Union` operation, we then sort based on predicate range.

4 Experimental Testbed: Shape Simalarity-Based Indexing CAD Data

We validate our approach by using it to index the set of solid models that make up the National Design Repository. This database consists of a set of more than 55,000 solid models taken from practical engineering CAD environments. The database is hand-indexed by parameters such as the vendor that published the models, the model format, and other external attributes.

Our experimentation consisted of first precomputing the *Model Signature Graphs* for each of the models in the collection. This step is conceptually relatively straightforward, unfortunately the translation process is a inaccurate transformation. A number of models in the database have errors and abberations that lead to inappropriate Model Signature Graphs being generated. Fortunately, these cases are relatively easy to detect, and they are relatively infrequent, making them easy to eliminate from the computations. These graphs are generated in a LEDA file format, and are associated with the solid model that they were generated from.

The second step in the experimentation phase consists of loading these graph files, and the model files that generate the graphs into the PostgreSQL database, and the construction of the M-Tree index over the dataset.

4.1 Example Domain: Model Signature Graphs

We make use of a specialized graph structure in order to represent solid models in order to perform similarity comparisons. This structure, called a *Model Signature Graph (MSG)*, is constructed from the boundary representation of a solid model in a manner similar to that by Wysk et al. [20] and the Attributed Adjacency Graph (AAG) structures used to perform Feature Recognition from solid models [12].

The boundary representation (BRep) [11] 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 3D space. The MSG for a solid model is defined as a labeled graph, $G = (V, E)$. Each face, f in the BRep is represented by a vertex v in V. The vertex is labeled with attributes that describe the properties of the face within the solid model.

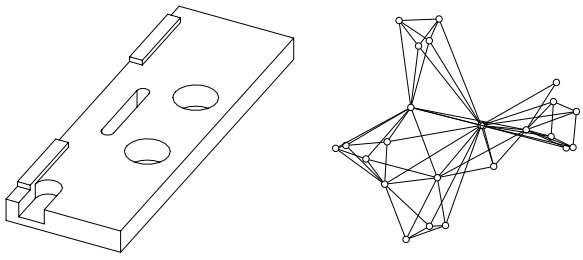
The attributes that we currently keep track of in the MSG include:

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

Edges in the MSG correspond to the solid model edges that connect faces in the model BRep. An edge $\langle v_1, v_2 \rangle \in E$ exists whenever the faces f_1 and f_2, that correspond to vertices v_1 and v_2 respectively, both share a BRep edge e. MSG edges are labeled with a number of features, including:

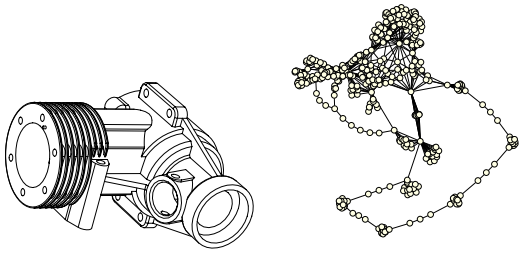
1. topological identifier for the edge e in the model;
2. concavity/convexity of e;
3. underlying geometric representation of the curve of e, i.e. the type of function describing the curve;
4. the length of the curve of e.

The asymptotic complexity of the transformations we apply from solid models to MSGs is linear with respect to the number of faces and edges in the BRep of the solid model. In practice, the translation is done in moments on our hardware (Sun Ultra 60 workstations), and we will ignore this cost in further analysis.



(a) Solid Model (b) Model Signature Graph

Figure 7: The *spinner3* (15KB) model and its transformation into a Model Signature Graph.



(a) Hidden Lines Removed (b) Model Signature Graph

Figure 8: Representations of the *crankcse* (651KB) model as both a solid model and MSG.

MSG Example. Figure 7 illustrates the transformation from the BRep of a simple solid model into a Model Signature Graph. Figure 8 depicts the transformation from the BRep of a crankcase to a Model Signature Graph. The complexity of the relationships between the faces and edges in this model's BRep gives a good idea of the difficulty in performing shape similarity based on BRep data. Effective practical algorithms in the research literature for determining the similarity of adjacency structures of this form are scarce. MSGs extract the essential topological data in the BRep for use in comparison, but the resulting graph is still exceedingly complex. Making use of MSGs for similarity assessment proves to be an extremely challenging problem to solve.

4.2 MSG Space Indexing

Metric Functions. Given the distance measures introduced in Sections 3.1.1 and 3.1.2, we can investigate how beneficial each of the measures would be when indexing the metric space. To further understand the global properties of these metric spaces, we applied the distance measures to random pairs of MSGs chosen uniformly from the National Design Repository database. The results of the distance samples were then used to build an approximate probability distribution function for the distance function over our set of MSGs. We then proceed to evaluate these distributions using techniques from information theory.

Figure 5 reflects the nature of the spaces defined by the $VertexDistance$ and $EigenDistance$ measures. The distributions associated with each of these distances is an extremely important aspect of the metric space. In particular, it determines how well an optimal indexing technique will be able

to organize the data within the metric space. In the worst case, the distribution looks like a Dirac Delta function, in which the distance function is $\delta(x, y) = \{0, k\}$ for some constant k. In this case, no information can be extracted from a distance calculation except for the fact that two elements are the same, or they're not. A full search over all elements must be performed to locate a given object within the set. In the best case, the distribution is a uniform distribution, and each measurement would provide a lot of information about the space.

Closer examination of Figure 5 reveals that each of the distributions that we are investigating are generally Gaussian distributions. The extremely heavy spike in the $VertexDistance$ probability density function indicates that almost all distances are in the range of 0-200 on a scale that reaches to nearly 20,000 in our particular sampling. This indicates an extremely small amount of entropy in the system, and indicates that the benefits of indexing the space will be generally limited [3].

Most of the $EigenDistance$-based measures appear to have a biased Gaussian-like distribution. The most distinctive of the plots appears to be generated by comparing the sums of eigenvalues within the graph spectra. The plots of the distance measures seem to indicate that padding the eigenvalue vectors with either 0.0 or 2.0 for comparison would yield the best results. The fact that the measures based on the eigenvalue spectra appear more evenly distributed than the $VertexDistance$ measure is reassuring and supports the possibility that using $EigenDistance$-based measures for indexing may yield performance benefits.

In order to quantitatively analyze each of our distance measures, we make use of Shannon's and Weaver's measure of information [18]. This function is a measure of how much information (in terms of bits) is required to determine elements within a space. This is analogous to the number of distance computations that are required to locate an element or set of elements within a metric space. To compute the amount of information in each of our distance functions, we discretize the probability distribution function for the distance measures into a set of ranges $R[1]$ through $R[N]$, for a positive constant N. We can then evaluate the following sum:

$$H = \sum_{i=1}^{N} -P(R[i])log(P(R[i]))$$

$P(R[i])$ represents the probability of evaluating a distance function on two graphs within the metric space, and the resulting distance to be found within the range $R[i]$.

The information contained in each of these measures is as follows:

Distance Function	Information
$EigenDistance\ Truncate$	5.4033
$EigenDistance\ Pad(0.0)$	5.1086
$EigenDistance\ Pad(2.0)$	4.935
$EigenDistance\ Pad(1.0)$	4.597
$EigenDistance\ Sum$	3.4009
$VertexDistance$	0.9886

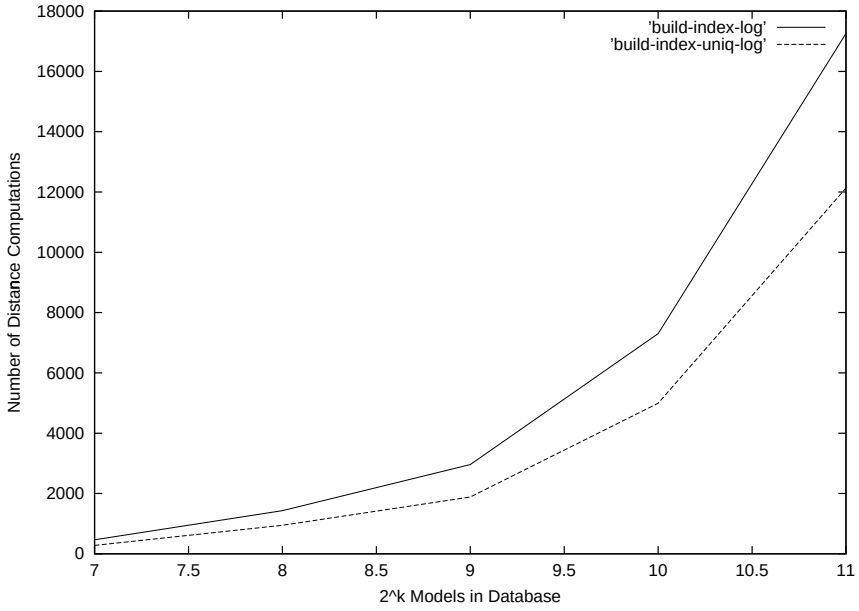
These measures of information seem to correspond with the visual impression gathered from the probability distribution functions. It should be noted that the distance function with the largest measured information quantity is the variant of $EigenDistance$ in which the longer eigenvalue vector is truncated and compared with the other eigenvalue vector. Unfortunately, the lack of metric properties of this measure make it less suitable for use in indexing than it would otherwise appear.

Further refinements may be done to these distance measures in order to enable them to provide more information to the user. On the other hand, we must be careful in that the distance measure can be made "too good," leading to distances that don't have a meaning that is easily understood by human users.

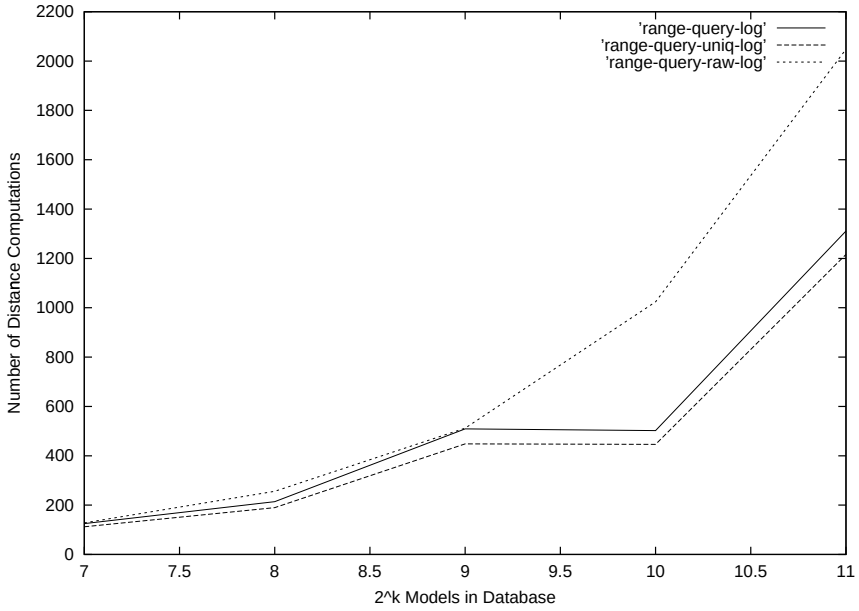
M-Tree Evaluation. We evaluate the performance of the index by maintaining a count of the number of distance computations that are performed while the index is being built and when querying the database. This is an accurate measure of the performance of the graph database system given that the distance calculations we perform are computationally expensive. For extremely large graphs, the eigenvalue decomposition operation may require several minutes to compute.

Ciaccia et al. [5] evaluate the performance of the M-Tree index by counting the number of *unique* distance calculations performed in building and using the index. We believe that the total number of computations performed, regardless of caching previous values is an important measure of database performance. In very large databases, it quickly becomes impractical to cache large portions of the $O\left(n^2\right)$ distance computations that may be performed by the database system.

Figure 9, part (a) depicts the number of distance computations performed by our indexing software as the size of the database grows. The line for `build-index-log` represents the number of computations performed by our system, whereas the



(a) Building the Index



(b) Performing a Range Query

Figure 9: The number of distance computations performed during the indexing and querying phases of solid model databases of varying size. When distance computations are expensive, they dominate the complexity of the indexing and retrieval operations.

line `build-index-uniq-log` represents the number of unique distance calculations done. It should be noted that while the number of computations that are redone becomes increasingly more significant as the size of the database grows, both plots appear to exhibit asymptotically similar behavior.

We also evaluate our system by performing a number of range queries over the database after it has been indexed. We have found that the cost of an individual range query can vary fairly wildly depending on the range of the query, as well as the graph used as a basis for the query, so we examine the average performance of such a query. Figure 9, part (b) depicts the number of distance computations performed while executing a range query in the database. Without an index, the query executor must examine each model in the database, and perform a distance calculation between it and the query graph. The number of computations is illustrated by the plot of 'range-query-raw-log' in the figure, which naturally grows linearly with the number of models in the database. 'range-query-log' indicates the number of distance calculations done by our software with the M-Tree index enabled. It is interesting to note that the cost of the query doesn't seem to be decreased until the number of models in the database reaches more than 2^9 (512). After this point, the number of distance calculations with the index is consistently several hundred fewer than without. Another important point to note is that the number of recomputed distances during the range query execution phase is extremely small. As a result, the performance benefits of caching distance calculations seems much less beneficial during runtime—most of the performance benefit will be had during the construction of the database. As a result, if the database is generally static, this may not be a necessary optimization.

5 Conclusions

Contributions. This paper presented our approach and preliminary experimental results toward a general indexing scheme for databases of graphs. In this approach, graphs are mapped into a vector-based *graph comparison space*, in which we can compute metric distances between graphs effectively. The *EigenDistance* measures we construct enable us to build a spatial index using an M-Tree data structure and develop heuristics that exploit properties of the graph comparison space to ensure that the M-tree indexing operations remain efficient. Our approach has the following advantages:

1. Our data model for storing graph-based information in databases additionally fits well into the standard relational database paradigm, reducing the learning costs required to use other graph storage techniques.

2. Our approach also provides an efficient mechanism for dealing with inexact comparisons between a large collection of independent graph structures. Existing graph-based data models fail to handle data in this form adequately.

3. Through the use of metric distance functions as a basis for indexing a database of graphs, we demonstrate that graph databases do not require intimate knowledge of the graph data structure to efficiently handle graph data. This is especially beneficial if a domain requires a concept of approximate similarity between graph data structures, as extremely efficient comparison operators may be designed for such purposes.

We provided an proof-of-concept that showed how graph databases can be used to index solid models of mechanical designs. In this approach, CAD data is mapped to *Model Signature Graphs*, through which we can compute metric distances among other CAD models in the graph comparison space. We validated this approach for representing CAD data with the models contained within the National Design Repository and present results that assess the quality of the *EigenDistance* measures in pairwise comparison of solid models via MSGs as well as the graph-based M-tree.

This work represents a contribution of significance to both the database and engineering communities, addressing a unique space of database and data management problems that are beyond the present scope of existing multimedia and spatial database technologies. Our contributions include a novel approach to manage a unique media type (graphs) and answer queries of practical importance to engineering design and manufacturing enterprises. We see this research as a foundation to enable comprehensive and flexible indexing of engineering design data and meta-data to allow engineers to execute knowledge-rich queries about device structure (e.g., shape), behavior (e.g., physical properties and performance) and function (e.g., design rationale and intent).

Future Work. A number of paths to improve upon our approach to graph databases exist. The data model and index methods that we make use

of only exploits structural connectivity between vertices within graphs. Both labeled graphs and directed graphs provide extra structure that can be exploited in such a graph database system. First, these extensions may be used to improve the representational powers of the system. Second, adding labels to vertices and edges provides extra information that graph comparisons can exploit.

Further development can be done on the graph similarity functions that we construct. In the CAD domain, vertices within Model Signature Graphs represent a set of constituent components of a model object. Some constituent elements may relate to others in implicit ways not captured directly within the graph itself. Domain information may be exploited in the construction of comparison operators for graphs in this manner to achieve better results.

Additionally, in the case of general graph structures, recursive partitioning techniques can be used to construct a hierarchical decomposition of a graph, based on minimization of edge crossings or other cost functions. Done properly, we believe that comparisons of graphs based on such a hierarchical decomposition would improve the quality of similarity metrics.

One of the central goals of this research has been to construct similarity measures between graphs to provide practical metrics for locating similar graphs from a database. In many domains based on graph data (including our example CAD domain), the very definitions of similarity are frequently ill-defined. In the CAD domain, one measure of similarity would be purely based on human intuition — two MSGs are similar when a human observer believes they are similar. Alternatively, a similarity measure may be based on whether two MSGs represent models that share machining and feature information (ie: both models are solid castings, with two through-holes).

The development of graph attributes in the domain of solid modeling and engineering requires an ontology of engineering features and attributes that are practically applicable to databases of CAD data. While no canonical set of CAD features exists, a number of practical systems that detect and make use of features have been created and deployed in industry. Feature recognizers for CAD data can translate BRep information into a set of semantically relevant features for machining or otherwise building a model. Information about the locations, dimensions, and orientations of features may be used to improve the performance of our similarity measures. By leveraging some earlier work [6, 16], we hope to incorporate additional feature attribute information into our indexing and comparison algorithms. In addition, information about engineering tolerances, surface finishes, constraints and parametrics, etc. all can be used to augment the basic techniques presented here. Using such domain-specific knowledge in other domains of graph data, index operations can be improved to be more semantically relevant to database users.

We believe that the graph-based similarity indexing approach that we have presented is applicable to other 3D media domains: solid free-form (SFF) fabrication techniques often use Stereolithography (STL) format files; VRML is a common modeling format for web data; and video games, which employ discrete 3D world models. In other domains, chemical structures, collections of communications networks, software component program and data flows, and other graph structures are ideal for storage using our approach. In each of these cases, there are geometry, topology, shape and semantics which can be used to create database access methods that can store the data in order to enable meaningful queries.

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 and Grant ENG/DMI-9713718.

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] Norman L. Biggs. *Algebraic Graph Theory.* Cambridge Tracts in Mathematics 67. Cambridge University Press, 1974.

[2] Tolga Bozkaya and Z. Meral Özsoyoglu. Distance-based indexing for high-dimensional metric spaces. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 357–368. ACM Press, 1997.

[3] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of VLDB 1995*, pages 574–584, 1995.

[4] Fan R. K. Chung. *Spectral Graph Theory*. Number 92 in Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[5] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB*, August 1997.

[6] 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 1997. ACM, ACM Press. Atlanta, GA.

[7] A. Fu, P. Chan, Y.L. Cheung, and Y.S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *VLDB Journal*, 9:154–173, 2000.

[8] Ralf Hartmut Güting. GraphDB: Modeling and querying graphs in databases. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 297–308, 1994.

[9] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *VLDB'95, Proceedings of the 21st International Conference on Very Large Databases*, 1995.

[10] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

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

[12] 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.

[13] Kurt Mehlhorn and Stefan Näher. *LEDA A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[14] M. Otterman. Approximate matching with high dimensionality R-trees. *M.Sc Scholarly paper, Dept of Computer Science, Univ. of Maryland, Collage Park, MD*, 1992.

[15] Sylvain Petitjean. The enumerative geometry of projective algebraic surfaces and the complexity of aspect graphs. *International Journal of Computer Vision*, 19(3):1–27, 1996.

[16] William C. Regli and Vincent Cicirello. Managing digital libraries for computer-aided design. *International Journal of Computer Aided Design*, 32(2):119–132, Februrary 2000. Special Issue on *CAD After 2000*. Mohsen Rezayat, Guest Editor.

[17] Andy Schuïr. PROGRESS: A VHL-language based on graph grammars. In *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 641–659, 1990.

[18] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, 1949.

[19] Ali Shokoufandeh, Sven J. Dickinson, Kaleem Siddiqi, and Steven W. Zucker. Indexing using a spectral encoding of topological structure. *Computer Vision and Pattern Recognition*, 2, 1999.

[20] Tien-Lung Sun, Chuan-Jun Su, Richard J. Mayer, and Richard A. Wysk. Shape similarity assessment of mechanical parts based on solid models. In Rajit Gadh, editor, *ASME Design for Manufacturing Conference, Symposium on Computer Integrated Concurrent Design*, pages 953–962. ASME, Boston, MA. September 17-21. 1995.