This document contains the draft version of the following paper:

A.S. Deshmukh, A.G. Banerjee, S.K. Gupta, and R. Sriram. Content-based assembly search: A step towards assembly reuse. *Computer Aided Design*, 40(2):244-261, 2008.

Readers are encouraged to get the official version from the journal's web site or by contacting Dr. S.K. Gupta (skgupta@umd.edu).

Content-Based Assembly Search: A Step towards Assembly Reuse

Abhijit S. Deshmukh, Ashis Gopal Banerjee, Satyandra K. Gupta[1]
Department of Mechanical Engineering and The Institute for Systems Research
University of Maryland, College Park
MD 20742, U.S.A.

Ram D. Sriram
Manufacturing Systems Integration Division
National Institute of Standards and Technology
Gaithersburg, MD 20899, U.S.A.

ABSTRACT

The increased use of CAD systems by product development organizations has resulted in the creation of large databases of assemblies. This explosion of assembly data is likely to continue in the future. In many situations, text-based search alone may not be sufficient to search for assemblies and it may be desirable to search for assemblies based on the content of the assembly models. The ability to perform content-based searches on these databases is expected to help the designers in the following two ways. First, it can facilitate reuse of existing assembly designs, thereby reducing the design time. Second, a lot of useful design for manufacturing and assembly knowledge is implicitly embedded in existing assemblies. Therefore a capability to locate existing assemblies and examine them can be used as a learning tool by designers to learn from the existing assembly designs. This paper describes a system for performing content-based searches on assembly databases. We identify templates for comprehensive search definitions and describe algorithms to perform content-based searches for mechanical assemblies. We also illustrate capabilities of our system through several examples.

Keywords: Content-based assembly search, graph compatibility, assembly mating conditions, and assembly characteristics.

1 INTRODUCTION

Over the last ten years Computer-Aided Design (CAD) systems have become very popular in the industry. These CAD systems are being used to generate models of parts and assemblies. These models are used as a basis for engineering analysis and generate manufacturing plans. CAD models also allow virtual prototyping. This reduces the need for physical prototyping. Nowadays, organizations routinely set up databases of CAD models to enable all participants in the product development process to have access to 3D data to support their functions. Specifically, manufacturing and service engineers are expected to greatly benefit from these databases. These databases are updated with the latest versions of parts and assemblies. This significantly improves information dissemination. CAD databases for even moderate size companies are expected to be large in size. A product assembly can contain many subassemblies and each subassembly can contain many parts. Therefore, even a small organization that has multiple product lines may add hundreds of assemblies to their database every year.

[1] Corresponding Author

In addition to supporting downstream manufacturing and service operations, assembly databases can be very useful during the design phase as well. There are two main uses of an assembly database during the design stage. The first possible usage is to locate existing assemblies that can be reused in a new product. Such reuse reduces product development time by eliminating the need for modeling, analysis, and process planning for the assembly being reused. The second possible usage is to provide access to existing design knowledge. Once designers manage to find an assembly with the desired characteristics, they can also access associated data such as cost, reliability, and failure reports.

Currently, designers have access to several types of search tools. If the assemblies are stored in hard drives, they can use file name based search tools. This strategy only works if a meaningful file naming convention based on assembly content is adopted. Unfortunately, many organizations do not name assembles in a meaningful way. For example, a research lab that we visited names assemblies by the product numbers. Any designer who has not worked on a particular product will not be aware of the existence of that product number. A molding shop that we visited names mold designs by dates and customer names. Moreover, when a company acquires another company, often the file naming convention used by two companies are not consistent. Such inconsistencies may require significant amount of manual labor to rename files. These problems suggest need for content-based search tools.

Another way to index assemblies is to attach text notations to assemblies and store them in a Product Data Management (PDM) database. In this indexing scheme, assemblies need to be manually annotated or text labels need to be extracted from CAD models. This scheme provides search capabilities based on text. However, text based search is not always the best way to search for assemblies.

Recently Kopena and Regli have developed archival and retrieval schemes based on semantic web technologies [Kope03a, Kope03b, Kope03c]. These schemes can be used to perform archival and retrieval based on object labels and simple relationships among objects. The goal of their work was to demonstrate how semantic web technologies can be applied to building engineering repositories. Identification of comprehensive set of attributes and relationships that are useful for performing efficient and user friendly assembly search was not part of their scope. Other representative work in the field of product design archival and retrieval systems include [Bohm05, Regl00].

Increasingly, limitations of text-based search are being recognized and methods are being developed to perform search based on the content itself. Such methods are able to exploit rich domain dependent information present in the data [Ense00]. For example, in the recent past, several part geometry-based search tools have emerged. However, these tools, although useful for part searches, are not very effective for assemblies [Gupt06]. They can only account for the overall shape of assemblies and do not take into consideration the relationships and structures that exist in them. Currently, content-based search tools do not exist for searching assemblies based on these aspects. Therefore, designers locate assemblies by manually opening various files and browsing through them using a CAD system. This is a highly inefficient use of designer's time. This becomes a serious problem as the numbers of assemblies in the database grow.

Assemblies contain a very large amount of information and complex relationships. Performing computationally efficient and easy-to-specify search on assemblies will require careful selection of the search criteria and associated definitions. Our central premise is that users will utilize distinguishing characteristics of the assemblies to initiate search to locate the desired assemblies from the database. In order to perform efficient searches, these distinguishing characteristics will need to include information other than text labels associated with the assemblies. Moreover, these distinguishing characteristics will be different in different domains. A large number of assemblies might be present in the database. But often certain distinguishing characteristics can be recalled by the designers to get a small number of desired results.

In this paper, we describe a system for performing content-based searches on an assembly database. We discuss the main requirements for the content based assembly search system. The high-level concept behind the content-based assembly search is shown in Figure 1. We present templates for defining a wide variety of assembly search queries that we have identified after studying user needs. We also describe in detail new algorithms for performing mating relationship based search and characterize their performance. Finally, we describe our system implementation and show several examples to illustrate possible usage of content-based assembly search system.

2 SYSTEM REQUIREMENTS

We conducted a survey involving total of seven users to help us identify assembly search system requirements. These included engineers at a government laboratory, mold designers at a molding shop, engineers working at a small company building custom electronics hardware, and students designing robots. All of these users use assembly design modules. Based on the user responses, we derived the system requirements listed below.

- *Comprehensive Search Criteria*: Different users are likely to remember different characteristics based on the context and their prior familiarity with the assembly that they are looking for. For example, a person who has not seen the assembly but only heard about it may recall different things about the assembly than the person who has actually seen the physical assembly. In order to ensure that the system caters to the needs of a wide range of users and different search contexts, it must support comprehensive search criteria for defining queries. Possible search criteria include information used to define assembly using a CAD system. In addition, any information or characteristic that can be extracted from the information present in assembly models and is likely to be utilized by users during the initiation of the assembly search must also be supported.

- *Ease of Use*: A typical assembly modeling module in a commercial CAD system records a lot of information about the assembly. Hence, comprehensive search criteria requirement is likely to result in hundreds of different criteria. In presence of such a large number of criteria, it might be difficult for users to locate the criteria they are looking for. Hence, they must be organized in a navigation taxonomy that groups similar search criteria together. Navigation taxonomies usually form a tree. We are interested in grouping search criteria such that the navigation tree is not very deep and yet at any level there are not too many branches.

Furthermore, such grouping should be intuitive to navigate so that a new user familiar with assembly modeling concepts can locate the desired search criterion with four or fewer mouse clicks. There are many other additional measures of ease-of-use that can be explored in future implementation [Niel94].

- *Search Flexibility*: Often users like to initiate search with bare minimum pieces of information and adjust the search criteria based on the search results. Hence, the system should support an iterative search refinement process. In other words, if the search results are too few then the user should be able to reduce the strictness of the search criteria by increasing the cut-off values. Also, if the search results are too many then the user should be able to introduce additional criteria in the form of more assembly characteristics in subsequent searches. Each consecutive search with additional constraints should be performed on only the set of assemblies identified during a previous search.

- *Search Efficiency*: Ultimately for the system to be useful, it should produce search results quickly. Hence, it should employ efficient search algorithms. Our goal is to be able to return search results within one minute for a 10,000 assembly database. The rationale for this target was the following. Many small and medium sized organizations have assembly databases smaller than 10,000 in size. In order to use assembly search in iterative refinement mode, the search time has to be smaller than one minute.

Our goal was to gather main system requirements to develop a research prototype. Additional criteria described in ISO/IEC 9126 standard should be explored in future extensions [Jung04].

3 SELECTION AND ORGANIZATION OF SEARCH CRITERIA

To build a list of all the characteristics of an assembly on which a search can be performed, an extensive review of three existing CAD systems (Pro/Engineer, Unigraphics, and SolidWorks) and literature in the assembly modeling field [Anan96, Brun00, Lee85, Moll93, Shah93] was performed. A suitable format, independent of any CAD system, was developed to store all the characteristics in an assembly as its signature. The assembly format defined in [Gupt01] is used as the basis to store the signature of an assembly. The current implementation supports the mating conditions and joints available in Pro/Engineer.

The identified characteristics were then classified into different categories to make it easier for the user to navigate through the search criteria. As can be seen from Figure 2, a typical assembly is made of parts and the relationships among them. The parts are related to each other either by the articulated joints created between them or by the mating conditions that hinder their movements. The assembly design process, along with this top-level breakdown of any typical assembly, forms the backbone for organizing our search criteria. The identified characteristics are categorized into four main classes. These criteria are described in Sections 3.1 through 3.4.

3.1 ASSEMBLY STATISTICS AND ANNOTATIONS

A possible way to search for existing assemblies is based on the overall assembly statistics. The following scenario illustrates why this type of search is useful in certain situations. Let us

consider the case of an organization that designs and builds prosthetic devices. When a customer approaches the organization with his own specific requirements, the designers in this organization would prefer to locate an existing assembly that is close to the given requirements and then adapt this existing assembly to the new requirements. The ability to effectively locate the most appropriate existing assembly will eliminate the need to design the assembly from scratch and hence reduce the design time significantly. A possible way to search for existing prosthetics will be based on the size of existing prosthetic assemblies. This scenario illustrates the benefits of being able to search based on the overall assembly statistics.

Based on our analysis of user needs for performing content-based assembly search, we have identified the following search criteria related to assembly statistics and annotations:

- ***Size***: The user can search assemblies based on the bounding box size or bounding sphere size of the assembly. The bounding sphere is defined using the radius of the sphere and the bounding box is defined by the length, width, and height of the rectangular box.

- ***Number of Parts***: The user can search assemblies based on the number of parts in an assembly. In addition, the user also has an option to either include or exclude the standard fasteners from the part count in the assembly. This option has been provided to overcome the situations where a user would remember the main parts in the assembly but not remember the total number of fasteners used in the assembly.

- ***Number and Types of Articulated Joints***: The user can search assemblies based on the number and types of joints in the assembly. All joints that are currently available in Pro/Engineer are supported. This type of search is defined by indicating the number of joints in each selected joint type. Even though the system currently uses Pro/Engineer joint types, we can easily extend it to work with joint types found in other CAD systems.

- ***Number of Usages in Other Assemblies***: An assembly such as a motor can be a popular subassembly and might be used in many other assemblies. Hence some users might remember the number of usages associated with such a popular subassembly. Hence, users can specify this as a possible definition of search. This might be an effective way of searching a frequently used assembly.

- ***Overall Shape Characteristics***: Currently, we support the following two methods to search for assemblies based on overall shape characteristics. First, the user can specify the percentage of rotationally symmetric parts in the assembly. Second, the user can specify the percentage of sheet metal parts in the assembly.

- ***Names of Conformance Standards***: Often assemblies are designed to meet certain testing and/or performance standards. Names of these standards are often included as notes in an assembly drawing. Therefore, a possible way to search for assemblies is to specify standards to which an assembly conforms. We allow users to specify names of conforming standards as strings.

- ***Designer Name***: Assembly file attribute also contains the name of the person who created the assembly. Therefore, assemblies can also be searched by specifying the designer's name as a string.

Many of the above criteria require the users to specify a positive real number for constraining the search. Two types of search definitions are implemented for specifying such searches. The first type of definition is based on range. In this case the user can specify an upper and lower limit on the search attribute. We also allow the user to leave either the upper limit or the lower limit as unspecified. The second type of definition is based on the target attribute value. In this case the user specifies only the target value. All relevant entries in the database are compared and ranked against this target value. A user can also select multiple different criteria from the above list to define a search.

3.2 CONSTITUENT PARTS

Assemblies can be searched based on the constituent parts of the assembly. Consider a scenario where the designer wants to search for a rocket motor assembly that contains a Beryllium liner of a specific size. Rocket motor assemblies are custom made to satisfy specific requirements. The designer would then search for an assembly by specifying the size and material of a part present in the assembly.

Based on our analysis of user needs for performing content-based assembly search, two criteria for search based on constituent parts are:

- ***Geometry***: The geometry-based assembly search has different inputs based on whether a part is a standard part or a custom part.

  - ***Standard part:*** Every organization has a library of standard parts. This criterion is useful when a designer knows that a certain set of standard parts are used in the assembly. In this method, the user can select any set of standard parts from the library and search for assemblies containing these parts.
  - ***Custom part:*** This is useful in a scenario when the designer knows that a part used in the assembly is similar to a part in the database. The user can specify a part to be matched approximately with the parts in the assembly database.

- ***Part Characteristics***: The following criteria for part characteristics-based search are supported:

  - ***Material of the part****:* Some assemblies contain a part made of a specific material. This criterion is useful to search for assemblies that contain a part that is made of an uncommon material. Users can specify any material from the available list of materials in the database of the organization.
  - ***Name of the owner****:* CAD files generally store the name of the creator and the modifier in part history. In most CAD systems, this refers to the login names of users in the Operating System (OS). This data is useful since designers can search for assemblies by the name of

a designer who worked on a specific project. The user is allowed to select the name of a designer from the database of designers' names in the organization.

The user can define any combination of the above criteria to define a single search. Range and target value definitions for numbers and exact and approximate match options for strings are also available. The problem of finding an approximate match is usually referred to as "search for similar parts." This problem has been explored in many different design and manufacturing contexts [Card03, Li04]. There are broadly two different methods. The first method uses the overall object shape in identifying similar parts. The second method uses shape features in identifying similar parts. Representative techniques in this area include [Card06a, Card06b, Cici01, Rame01]. Both these approaches have their own relative merits and demerits. Depending on the type of the application, one might prove to be better than the other. We use the approach described in [Card06b].

3.3 SEARCH BASED ON PART MATING CONDITIONS

Mating conditions are the restraints (constraints) imposed on the location of a part with respect to other parts in the assembly. Mating conditions play two fundamental roles in design and manufacturing of assemblies. First, they directly correspond to the product structure and its architecture. In many products, there is a base part and many other parts are attached to it. So how the parts mate with each other in the assembly gives the product its unique structures. Often users are also able to visualize and remember how the main parts in the products are connected to each other. For example, many users will be able to quickly recall how parts in a table fan are connected to each other. Second, mating conditions also often directly relate to the assembly process. So the users that are trying to recall an assembly from the manufacturing point of view are able to recall which parts will be assembled to which other parts. These process steps often directly translate into mating relationships in the assembly. These two roles of mating conditions make them very useful distinguishing characteristics for identifying assemblies. Hence, we have developed a search criterion based on mating conditions.

In order to use this search criterion, the user specifies the mating condition between parts of a desired subassembly or assembly by building a query mating graph. Individual parts are represented as nodes or vertices and edges connect two nodes when mating conditions exist between the two corresponding parts. This query mating graph is compared with mating graphs corresponding to the assemblies in the database. The results of the search process are all assemblies whose mating graphs are compatible with the query mating graph. Formal definition of compatible graphs is presented in Section 4.

Each node in the query mating graph has the following attributes:

- ***Category***: This represents whether the part is a standard part or a custom part. The user can select between either of the two options, or leave this attribute unspecified.

- ***Geometry***: This attribute is a pointer to the geometry of the part. The geometry for standard parts is referenced from the library of standard parts in the organization, whereas for custom

parts it is referenced from the best approximate match found in the assembly database as explained in the previous sub-section. This attribute can also be left unspecified.

- ***Type***: This attribute is defined only for standard parts and specifies the subcategory of the standard part. These options are available to the designer in a pull down menu. The designer can select a specific variant of the part from another pull down menu after selecting the category to be a standard part. This attribute can also be left unspecified.

- ***Degree***: It represents the total number of parts that are mated to the part represented by a node. This attribute can be given a specific value. If no value is specified, it is taken to be zero. The query graph may involve nodes with a very high degree. This scenario often happens in case of assemblies where a base is used to mount all parts. Printed circuit boards (PCBs) or fixtures with a common base plate are examples of such assemblies.

Every edge in the query mating graph has the following attributes:

- ***Type***: This represents the type of mating condition represented by an edge. This attribute can also be left unspecified.

- ***Vertex-1***: This attribute stores the identifier of the node from where the edge originates. This attribute cannot be empty. The designer needs to specify the node from where the edge originates.

- ***Vertex-2***: This attribute stores the identifier of the node where the edge terminates. The query graph specified by the designer can be a partial graph with unspecified terminating node for an edge.

Most users only want to focus on defining a few key mating relationships to save time in defining the search query. Hence, the query mating graph need not be a fully specified graph. Many of the attributes in the query graph can be left unspecified (i.e., equivalent to wild cards in string search definitions). This means that the query mating graph is not a unique graph and many different database graphs might be compatible with the query graph. Figure 3 shows an example of a query mating graph containing wild card entries. It may also be noted here that missing mating condition data in the database graph may lead to erroneous results. So if users are aware that data is missing in assembly models, then they can use wild cards in the search definitions to account for missing data.

3.4 JOINT ORIENTATION RELATIONSHIPS

Joints in an assembly are directly related to its kinematic function. There is a well developed taxonomy that describes how different positioning and orienting functionalities can be realized by certain spatial special arrangement of joints [Tsai01]. So types of joints and their spatial relationships are directly related to the kinematic function of an assembly. Hence, a user looking for a certain kinematic functionality may use the spatial arrangement of joints to indirectly specify this functionality. However, the mapping between kinemtaic functions and spatial arrangement of joints is one-to-many [Tsai01]. So the user interested in searching for a

particular kinematic functionality may need to perform search multiple times with several different alternative spatial arrangement of joints.

Search queries based on joint types and number of joints can be defined using the criteria described in Section 3.1. Relative orientations of joints with respect to each other play a significant role in identifying assemblies. Since assembly pose changes in articulated assemblies based on joint parameter values, relationships among joints may also change. However, orientation relationships among joints that belong to a single, rigid part do not change. Hence, we use orientation relationships among joints belonging to individual, constituent parts as the basis for defining this search criterion.

To initiate the search, the user specifies the type of the joint and the orientations between various joints. The user can select a joint of any type that is available in Pro/Engineer. The possible orientations between joints are as follows: *Parallel, Perpendicular, Angle* (This is specified as a range value for angle between two joints), *Unknown relationship*, and *No relationship***.**

The "angle" relationship allows the designer to specify a range of values for the angle between the joints. The "unknown relationship" allows the designer to specify a wild card for search when the exact relation is not known. If orientation relationship cannot exist between two joints (e.g., two spherical joints), then the designer must choose "no relationship". In case of a planar joint, the angle is assumed to be defined with respect to the plane of joint. The angles are assumed to be defined at the zero position of the joints.

User input is provided in the form of a list of 2-tuples i.e., doubles. A representation of the double is as follows: $D = \{\{j_1, j_2\}, m\}$. The first element of the double is a set of the type of joints represented by j_1 and j_2. The second element represented by m is the relation between them. Each element of the set as well as the second element of the double is a string, which can be any one of the appropriate type described above. Only joints defined on a single rigid part can be represented as the first element of any particular double. For every assembly in the database, a list of doubles defining all relations in the assembly is extracted and stored. The system searches for an assembly that has all the query doubles in the list of database doubles.

Consider the example shown in Figure 4. The joint relationships present in the query assembly are given by:

$D_1 = \{\{prismatic, prismatic\}, unknown\}$

$D_2 = \{\{revolute, revolute\}, parallel\}$

$D_3 = \{\{prismatic, prismatic\}, \text{"120"}\}$

$E_1 = \{D_1, D_2, D_3\}$

Similarly, the database assembly joint relationships can be expressed as:

$D_1' = \{\{prismatic, prismatic\}, perpendicular\}$

$D_2' = \{\{revolute, prismatic\}, "120"\}$

$D_3' = \{\{revolute, revolute\}, parallel\}$

$D_4' = \{\{revolute, spherical\}, unknown\}$

$E_2 = \{D_1', D_2', D_3', D_4'\}$

This query assembly can then be matched with the database one as D_1, D_2 and D_3 correspond to D_1', D_2' and D_3' respectively (as shown by arrows in Figure 4). The order in which joints are specified in the first element of the double is immaterial. Presence of an additional joint relationship D_4' also does not affect our matching as extra joint relationships are allowed to be present in the more detailed database assembly. This is because the user may only recollect a portion of the entire database assembly and specify that portion (sub-assembly) in the form of a query.

4 CHECKING COMPATIBILITY OF MATING GRAPHS

The system builds a mating graph for every assembly in the database off-line. The parts are represented as nodes. For each node four attributes namely, category, geometry, type and degree, are determined and initialized. If two parts are mated together in an assembly, then an edge is created between the nodes representing the parts. The type of mating condition used and the identifier of two mated parts are the attributes of the edge. As all the information about part and mating conditions can be extracted from the Pro/Engineer files, this graph is a completely specified graph and does not include any wild cards that are typically associated with query mating graphs. However, since multiple mating conditions are possible between two parts, two nodes may be connected by an edge having composite labels in the database mating graph.

4.1 DEFINITIONS

Let *G1* = (*V1*, *E1*) be a query graph and *G2* = (*V2*, *E2*) be a database graph. *V1* and *V2* are sets of nodes. *E1* and *E2* are sets of edges.

Definition 1: Let $v \in V1$ and $v' \in V2$, then v and v' are said to be *compatible nodes* if the following conditions are satisfied:

a) Degree of v is less than or equal to the degree of v'.
b) Category of v is left unspecified or it is identical to that of v'.
c) If category of v is equal to custom then either geometry of v is not specified or it is equal to that of v'.
d) If category of v is equal to standard then either type of v is not specified or it is equal to that of v'.

Definition 2: Let $e_1 = (v_1, v_1')$ s.t. $e_1 \in E1$, $v_1, v_1' \in V1$ and $e_2 = (v_2, v_2')$ s.t. $e_2 \in E2$, $v_2, v_2' \in V2$, then e_1 and e_2 are said to be *compatible edges* if the following criteria are met:

a) v_1 and v_2 are compatible nodes and v_1' and v_2' are compatible nodes; Or, v_1 and v_2' are compatible nodes and v_2 and v_1' are compatible nodes
b) Type of e_1 is either left unspecified or it is identical to that of e_2.

Definition 3: Query mating graph *G1* is *compatible* with database mating graph *G2* if: (1) $\forall v \in V1$, there exists a corresponding unique node $v' \in V2$ s.t. v and v' are compatible nodes; and (2) $\forall e \in E1$, there exists corresponding and unique edge $e' \in E2$ s.t. e and e' are compatible edges. Figure 5 presents schematic examples of a pair of compatible and incompatible graphs.

We are basically interested in developing an algorithm that can determine if a given query graph *G1* is compatible with the database graph *G2* or not. Section 4.2 presents an algorithm for this. This algorithm uses the concept of matching sets defined below.

Definition 4: The *matching set* for node $v \in V1$ is defined as the set $V = \{v_{12}, v_{22}, \ldots, v_{n2}\}$ s.t. $V \subseteq V2$ and v is compatible with every element $v' \in V$.

4.2 OVERVIEW OF ALGORITHMS

The graph compatibility problem defined above is significantly different from a typical graph matching or isomorphism problem. Some of the key differences are listed as follows:

- **Presence of wild cards:** Many of the node and edge attributes can be left unspecified by the user as he/she may not recollect all the details about the particular assembly that he/she is trying to locate. Furthermore, most users like to specify minimal possible information to initiate a search. The algorithm needs to handle such cases and come up with all possible compatible graphs.

- **Presence of conditional node attributes:** Many of the node attributes are dependent on each other. For example, type attribute is defined only if category attribute is standard. Again, the definition of geometry attribute depends on the category. Hence, the algorithm must have a provision for incorporating such conditional behavior.

- **Need for precise match:** Any database assembly retrieved by the search algorithm should precisely match the user specified criteria because users are only interested in graphs that match their criteria. Hence, graphs need to be strictly compatible in the rigorous, mathematical sense as defined in the previous sub-section.

Approximate graph matching techniques are not going to work well for identifying compatible mating graphs due to the need for precise match. Moreover, the available subgraph isomorphism techniques cannot be used in their present forms to solve this problem [Boos06, Fort96, Rein77]. In the following paragraphs we review some of the graph matching algorithms that have inspired the design of our algorithm.

- Ullmann's [Ullm76] approach was one of the first attempts to solve the subgraph isomorphism problem. He used brute force backtracking search, which is a depth first tree

search method to solve this problem. It is used to find all isomorphisms between two connected graphs without taking into account any previous knowledge of correspondence between nodes of query graph and a database graph. Our algorithm is modeled after this approach.

- Yu and Wang [Yu04] have used a 2D continuous Hopfield Neural Network model to obtain a subgraph that is isomorphic to a given query graph. They construct a neural network with dimension equal to the number of nodes in the two graphs. They define essential conditions for subgraph isomorphism that can be used as pruning conditions and have been adopted in this paper. However, Ullmann's approach has been found more adaptable for implementing bounding conditions in the present work.

- Fuchs et al. [Fuch00] have proposed an error tolerant algorithm that uses prior knowledge of correspondence between nodes of a query graph and a database graph. They report that a prior knowledge of correspondence between nodes from query graph and database graph can be used to significantly reduce the search space. We have found that query graphs in assembly search scenario were quite small and hence we did not use this technique.

- Cordella et al. [Cord04] have suggested an algorithm for large graphs. They report significant improvement over Ullmann's approach as their approach is almost independent of the number of nodes in the query graph. One of the main contributions of this algorithm is the memory efficient data structure used during the exploration of search space. It also includes five rules for feasibility. The algorithm explores the search graph using a depth first strategy. However, their conditions are not directly applicable to our problem.

Since users are only going to remember certain specific aspects of the assembly that they are searching for, size (i.e., number of nodes and edges) of query graphs is expected to be relatively small. Hence, heuristic-based graph searching is a viable option in terms of computational speed. That is why, by adapting existing graph isomorphism techniques, we have developed a depth-first branch and bound algorithm to perform graph compatibility check. Different heuristics have been suitably modified and combined to design this algorithm. Since it needs to test various combinations of possible node compatibility, this process can be computationally expensive. Thus, to ensure that results are obtained in real-time, a two stage pruning process is initially carried out before the actual graph compatibility check is undertaken. Since the two classes of search criteria explained in Sections 3.1 and 3.2 are computationally very cheap, if applicable, they are used first. Only the assemblies that satisfy all the criteria are retained for further tests.

The top-level algorithm attempts to find all database graphs which are compatible with the query graph. It initially calls a function to assign priority score to each node belonging to the query graph based on its degree and other attributes. The priority score is used to identify the most constrained node so that it can be utilized as the root for depth first search while traversing the graph. As has been explained in [Boos06], if we can label the vertices properly, we can reduce the search space significantly. Based on our simulation experiments, higher-degree nodes are given greater preference than lower-degree ones. However, other attributes such as whether the geometry of a part (corresponding to a node) is specified, whether it is a standard part or a custom part also need to be taken into account.

Initially an absolute priority score is assigned to each node based on its degree, starting from zero with the highest possible rank being two less than number of nodes in query graph. If two or more nodes have the same degree, then identical absolute score is assigned to all of them. This score is then scaled to a relative value between zero and one. In addition, an intermediate rank is assigned with highest score given to a node that represents a custom part for which geometry is known and least score is assigned to a node for which no attribute is specified. As custom parts are used rarely in an assembly, they are assigned higher priorities. A weighted sum of the two (weights being chosen empirically based on simulation performance) is then chosen as the final score for each node. The node with highest score is the most constrained one and acts as the root of the depth first search forest.

The top-level algorithm then invokes CompatibilityCheck algorithm which takes as input a query graph and a database graph and finds whether the two graphs are compatible.

COMPATIBILITYCHECK (*G1, G2*)

Input

- *G1,* a query graph
- *G2,* a database graph

Output

- *match_found_status*, a Boolean variable that returns the status to indicate whether the two graphs are compatible

Internal Variables

- *matched_nodes_list* is a list of pair of nodes that have been found to match during depth first search (DFS)
- *active_nodes_inG1* is a list of nodes in query graph that have been discovered during DFS but for whom a match has not been found
- *active_nodes_inG2* is a list of nodes in database graph that have been discovered during DFS but have not been matched to any node from query graph
- *matching_set* is a list of all $q_i \in V_2(G_2)$ for all $v_i \in V_1(G_1)$: v_i can match with every q_i present in its list of feasible matches

Steps

1) Call PRELIMINARYCOMPATIBILITYTEST(*G1*,*G2*) and set Boolean *graph_pruning_check_pass* as the output of this function. If *graph_pruning_check_pass* is TRUE go to step 2, else set *match_found_status* as FALSE and return
2) Set *match_found_status* as FALSE
3) Select the node *p* with highest priority score in *G1*
4) Set *Q* as the list of feasible matches for *p* from the *matching_set* and arrange the elements in *Q* in non-descending order of the degree of nodes
5) While *Q* is not empty and *match_found_status* is FALSE, do the following:
    a) Pop the first element *q* from *Q*
    b) Set *matched_nodes_list* = {}

c) Set *active_nodes_inG1* = {}
d) Set *active_nodes_inG2* = {}
e) Call MATCHNODE (*p*, *q*, *matched_nodes_list*, *active_nodes_inG1*, *active_nodes_inG2, G1, G2, matching_set*) and set output as *match_found_status*

6) Return *match_found_status*

The COMPATIBILITYCHECK algorithm calls the PRELIMINARYCOMPATIBILITYCHECK algorithm to check whether the database graph passes all the pruning conditions. Six conditions are used to prune the list of feasible database multigraphs. Some of these conditions are analogous to the use of vertex invariants in solving the subgraph isomorphism problem [Boos06, Fort96].

PRELIMINARYCOMPATIBILITYCHECK algorithm performs the following tests:

- The number of nodes in the database multigraph should be greater than or equal to the number of nodes in the query graph. This is essential as all the nodes in the query graph can never be matched with distinct nodes in the database graph otherwise.
- Similarly the number of edges in the database multigraph should also be greater than or equal to the number of edges in the query graph as all the edges in the query graph need to be matched with unique edges in the database graph.
- The number of standard parts in the database graph (i.e. nodes having "standard" as the geometry based attribute) should also exceed or at least equal the number of standard parts present in the query graph so that all such query graph nodes can be possibly matched with distinct nodes having identical attributes in the database graph.
- The number of custom parts in the database multigraph must also be equal to or greater than the number of custom parts in the query graph. This follows from the same argument given in the first three cases.
- For every node in the query graph, at least one distinct node should exist in the database multigraph such that its degree is greater than or equal to the degree of the query graph node. This condition ensures that an injective relationship exists between the two graphs under consideration.
- The *matching set* corresponding to every node in the query graph should be non-empty. This set will help us in pruning certain DFS paths later on as well.

The following algorithm is the main DFS algorithm that matches each node from query graph to a node from database graph by recursive operations.

MATCHNODE (*p, q, matched_nodes_list, active_nodes_inG1, active_nodes_inG2, G1, G2, matching_set*)

Input

- *p*, a node from the query graph
- *q*, a node from the database graph
- *matched_nodes_list***,** a list of pair of nodes that have been found to match during DFS
- *active_nodes_inG1*, a list of nodes in query graph that have been discovered during DFS but for whom a match has not been found so far

- *active_nodes_inG2*, a list of nodes in database graph that have been discovered during DFS but have not been matched to any node as yet
- *G1,* a query graph
- *G2,* a database graph
- *matching_set*, a list of all $q_i \in V_2(G_2)$ for all $v_i \in V_1(G_1)$: v_i can match with every q_i present in its list of feasible matches

Output:

- The function will return a value of TRUE or FALSE to transfer control between recursions

Internal Variables

- Boolean *match_found_status* is a global variable and *Match_node* function will set the value for this variable to indicate if query and database graphs are compatible
- v_i, the number of nodes in *V1*

Steps

1) Call CHECKNODECONSISTENCY (*p*, *q*, *match_nodes_list*). If the output is TRUE go to step 2 else return FALSE.
2) Call CHECKMATCHNODEAVAILABILITY(*p, q, match_node_list*). If output is TRUE then go to step 3 else return FALSE
3) Set *matched_nodes_list* = $(p, q) \cup$ *matched_nodes_list*
4) For every *matching_set* of $v_i \in V_1(G_1)$, do the following
    a. Delete list of feasible matches of *p* from *matching_set*
    b. Delete *q* from list of feasible matches of all $v_i \in V_1(G_1)$ from the *matching_set*
5) If size of *matched_nodes_list* = v_i, then set *match_found_status* as TRUE and perform a global exit
6) Find set of all neighbors *P* for *p* in *G1*, that are not in *matched_nodes_list*
7) If *P* is a Null set, then go to step 14; else go to step 8.
8) Sort *P* in non-descending order of degree of the nodes
9) Place all the nodes in *P* that have an edge of the type unknown at the end of the list irrespective of their degree
10) For each $p_i \in P$, if $p_i \notin$ *active_nodes_inG1,* insert p_i to *active_nodes_inG1* at the beginning of *active_nodes_inG1*
11) Find all neighbors *Q* for *q* in *G2* that are not in *matched_nodes_list.*
12) Sort *Q* in non-descending order of degree of the nodes
13) For every $q_j \in Q$, if $q_j \notin$ *active_nodes_inG2* insert q_j to *active_nodes_inG2* at the beginning of *active_nodes_inG2*
14) While *active_nodes_inG1* is not a Null set and *match_found_status* is FALSE
    a. Find the list of nodes *K* in the *matched_nodes_list* that are neighbor of p' that is obtained by popping the first element in *active_nodes_inG1*.
    b. While *match_found_status* is FALSE and *K* is not a Null set do the following:
        i) Pop the first element $k \in K$.
        ii) Find the node *l* that matches with *k* from the *matched_nodes_list.*
        iii) Find the list of nodes $M \in$ *active_nodes_inG2* that are neighbors of *l*.

iv) While *match_found_status* is FALSE and *M* is not a Null set, then pop the first element from *M* and set it as q' and do the following:

(i) If *edge*(*k*, p') and *edge*(*l*, q') have the same label or *edge*(*k*, p') has label unknown then edges are compatible. If edges are not compatible then go to step 14-b-iv-ii. If edges are compatible then, do the following:

I) Set *active_nodes_inG1 = active_nodes_inG1* – { p' }

II) Set *active_nodes_inG2 = active_nodes_inG2* – { q' }

III) Call MATCHNODE(p', q', *matched_nodes_list*, *active_nodes_inG1*, *active_nodes_inG2, matching_set*). If function returns FALSE, then go to step 14-b-iv-ii. Else go to step 14-a.

(ii) Pop the next element from list *M* and repeat step 14-b-iv-i.

v) If *M* is a Null set then pop the next element from *K* and go to step 14-b-ii.

c. If *K* is a Null set, then return FALSE.

The above algorithm calls CHECKNODECONSISTENCY and CHECKMATCHNODEAVAILABILITY functions. These two functions are described below:

- CHECKNODECONSISTENCY function tests if the query graph and database graph nodes are compatible. If the query graph node is connected to another node that has already been explored, then it checks whether the database graph node also has a compatible edge to a node that has been previously found to be compatible with the explored query graph node.

- CHECKMATCHNODEAVAILABILITY function ensures that the current database graph node that is being tested for compatibility with the query graph node is not the only possible match for any other unexplored node present in the query graph. If the only possible compatible node for an unexplored query graph node is being considered currently (as can be judged by going through its matching set), then the two graphs cannot be compatible and hence this path can be pruned. Thus, it implements an efficient bounding condition during depth first search.

4.3 ALGORITHM DESIGN RATIONALE

Graph compatibility problem arising in the context of assembly search has the following unique characteristics that are exploited by our algorithm design:

- Each node has four different types of attributes. Conceptually, we can treat these attribute vectors as node types. Each of these attributes can be assigned many different values. This effectively increases the number of available node types to a very large value. Hence, in our application the available number of node types is significant larger then the number of nodes present in a typical database assembly mating graph. Furthermore, typically the user enters very few nodes in the query mating graph. Hence the number of nodes in a typical database mating graph is significantly larger then the number of nodes in a query mating graph. Let us assume that the effective number of node types is α, the number of nodes in a database mating graph is β, and the number of nodes in a query mating graph is γ. Then, the probability of nodes in a randomly selected query mating graph being compatible with nodes in a randomly selected database mating graph is very small. This can be shown mathematically using the following reasoning. The probability that the first query graph node

is compatible with the first database graph node is equal to 1/ α. Hence, the probability that it is not compatible with the first node in the database graph is given by (1-1/ α). Since there are β nodes in the database graph, probability that it is not compatible with any of the nodes in the database graph is equal to $(1-1/\alpha)^{\beta}$. Hence, the probability that it is compatible with at least one of the nodes is given by $\left(1-(1-1/\alpha)^{\beta}\right)$. Similarly, the probability that the second query graph node is compatible with at least one of the remaining (β-1) database graph nodes is equal to $\left(1-(1-1/\alpha)^{\beta-1}\right)$. Proceeding exactly like this, for the γ^{th} node, this probability is equal to $\left(1-(1-1/\alpha)^{\beta-\gamma+1}\right)$. Thus, the overall probability of finding a compatible node set in a database and a query graph is given by $\left(1-(1-1/\alpha)^{\beta}\right).\left(1-(1-1/\alpha)^{\beta-1}\right)\ldots\left(1-(1-1/\alpha)^{\beta-\gamma+1}\right)$. Since, α is of the order of thousands, β is of the order of hundred and γ is typically less than ten, the probability value is very small. This can be readily verified from the probability expression as the value of every term in the product is close to 0, as $(1-1/\alpha)^{\beta-\lambda} \to 1, \because \alpha >> 1, 0 \le \lambda \le \gamma - 1$. Furthermore, this pruning is performed very efficiently by our algorithm by creating matching sets. Hence our algorithm efficiently exploits the problem characteristics that α is much greater than the size of the database and query graphs.

- Whenever a user leaves wild cards in the query graphs, it effectively reduces α because a node with wild card(s) can match with many different nodes. However, usually γ is small and hence the number of neighbors that need to be tried for a compatible node during the search process is very small. This reduces the combinatorics encountered during the search process considerably. In most situations, users do not want to create query mating graphs with more than four or five nodes. Furthermore, in most search definitions, at least one of these nodes is defined by the user based on its distinguishing and uncommon characteristics (e.g., base part in a printed circuit board assembly) to effectively utilize query mating graph. Hence, even in the presence of wild card entries the heuristics used for assigning priority scores in our algorithm and small size of γ ensure that the search process is quite fast.

Based on the above observations, we can conclude that our algorithm will work well when the number of node types is larger than (1) the number of nodes in the database mating graph, and (2) the number of nodes in the query mating graph. The first condition is satisfied in our domain due to the large number of attribute values used in our formulation. The second condition is satisfied because most users are not interested in creating large query mating graphs. They instead prefer to create fairly small query graphs with nodes having highly distinguishing characteristics. Therefore, our algorithms perform well despite presence of wild cards in the query mating graphs.

4.4 COMPUTATIONAL EXPERIMENTS: RESULTS AND DISCUSSIONS

A database of 1000 mating graphs was created randomly to study this algorithm (additional experiments involving mating graphs of 200 actual assemblies will be described in section 5). We decided to use randomly generated graphs to eliminate any bias in the data. The number of nodes in each graph varied between 10 and 100. Since our algorithm consists of two major steps, namely pruning and actual depth first search (DFS), we decided to investigate the performance

of the DFS algorithm separately to characterize the overall algorithm in a better way. Four experiments were conducted, which have been described as follows.

The first experiment investigated how the total DFS computation time changes with respect to the number of nodes in the query graph. Each time a database graph passes the matching set pruning criteria a depth first search is performed to test compatibility of the query graph with respect to the database graph. The total DFS computation time reported in all the plots is the total time required for testing compatibility of query graph with all such database graphs. The number of nodes in query graphs varied from 8 to 17. Query graphs used in this experiment did not have any wild cards. The graph in Figure 6 shows the plot for the total DFS computation time against the number of nodes in the query graph. As can be observed from the graph, the time required for DFS increases more or less linearly with the number of nodes. For each node, another recursion call is made to the main function in the algorithm. This recursion involves matching the node attributes, querying and updating the list of nodes being currently processed both in the query graph and the database graph. This increases the amount of time required for the depth first search operation.

The second experiment investigated how the total DFS computation time changes with respect to the number of edges in the query graph. The total number of nodes in all the query graphs was 9 during this experiment. The number of edges in query graphs varied from 8 to 12. The query graphs used in this experiment did not have any wild cards. The graph in Figure 7 shows the plot for the total DFS computation time versus the number of edges in the query graph. As can be seen from the graph, there is no considerable change in time required for depth first search. The maximum variation in the timing of DFS is of the order of 0.0018s and is negligible. If the number of nodes is kept constant, addition of edges in the query graph may only lead to additional cyclic conditions. However, the cyclic edges do not lead to any new recursion. Instead, when the nodes are matched, few extra edges between nodes that have already been explored are matched. This involves string matching operations and the time required to retrieve edge information (attribute) from the graph data structure. The retrieval time again depends on the total number of edges in the graph. This does not increase the time for DFS significantly. Thus, the insensitive nature of the computation time with respect to the number of query graph edges is easily explained from a theoretical standpoint.

The third experiment investigated how the presence of wild card entries in the query graph affects the total DFS computation time. The total number of nodes and edges in query graphs were kept fixed at 10 and 9 respectively. We know that every node has two optional inputs. While a node representing a custom part has category and geometry attributes as optional inputs, a node representing a standard part has category and type attributes as optional inputs. Every edge has one optional input in form of the type of mating condition that it represents. The percent of unspecified (wild card) entries in the query graph can be calculated using the following formula.

> Percent of wild card entries in a query graph = (Number of wild card entries in the query graph) / (2 x the number of nodes in the query graph + the number of edges in the query graph).

Figure 8 shows a plot of total DFS computation time against percentage of wild card entries in the query graph. The query graph finds one matching assembly in all cases till 62% or lesser percentage of wild cards was present. The number of results increases thereafter eventually producing 15 matches when 79% wild cards were used. When the number of wild card entries for nodes is increased, multiple possible compatible nodes (arranged in the form of matching sets) can be found for each node at the time of its discovery in DFS. If only the category is specified, then the expected size of the matching sets will increase. If the category is also not specified, it can result in further increase in the size of the matching sets. Moreover, if the edge attribute is left unspecified, then multiple paths can be explored in DFS. As the number of possible compatible nodes and explored edges for every possible compatible node increase linearly, an overall quadratic increase in time is expected. This corroborates well with experimental results. As the percentage of wild card entries in the query graph are increased and more results are found, the time required for DFS increases almost quadratically.

The forth experiment investigated the effectiveness of the proposed heuristics. As a part of this experiment, we suppressed various heuristics and studied performance changes. If only degree and category attributes of nodes are used to create matching sets (instead of all the pruning conditions), then time taken increases by 691 times on an average. Moreover, if category is also not considered while creating matching sets, then average computational time increases by 2497 times as compared to considering all pruning conditions. Again, if a random query graph node is used as the root node instead of selecting the most constrained one, then average performance degrades by 87%.

Thus, it can be inferred from these experiments that the graph compatibility algorithm is computationally efficient and works well for typical query graphs. Moreover, the heuristics play an important role in reducing the DFS computation time.

5 IMPLEMENTATION AND RESULTS

5.1 IMPLEMENTATION

We have developed a system to support content-based searches. This system has been implemented using C++ and Microsoft Foundation Classes (MFC) library in Windows XP Professional platform. The MFC library provides the user interface (UI) component of the code. Microsoft Visual C++ .NET 2003 is used as the integrated development environment (IDE) to build an event-driven software. Figure 9 shows the framework of our assembly search system.

This search software system supports CAD data from Pro/Engineer Wildfire 3.0 educational edition. Pro/Toolkit, an application programming interface (API) provided with Pro/Engineer is used to extract the data from assembly files. A program can extract relevant information about the assembly such as the names of the constituent parts, mating conditions between different parts, the tree structure of the assembly, the material and mass properties of the parts and so on.

The API program requires the user to select a folder where the Pro/Engineer assemblies are stored. It then iteratively searches all sub folders and exports assembly data. The .stl files and XML file is exported in the same directory as the assembly. The program can extract information

from assemblies with multiple levels of hierarchy in its tree structure. The signature of each assembly is extracted before it can be used in the database of assemblies. The search software does not require any CAD software as it searches only based on the assembly signature. A separate viewer (shown in Figure 10) is provided to browse the assemblies and the search results.

The search based on assembly statistics, constituent parts and joints relationships uses approximate and exact string matching and number matching. This system supports number search based on range and target. For range matching, the user specifies a minimum and maximum number and the system searches for assembly that has the specified criteria within the given range. For target matching, the user specifies a target value and the system searches for an assembly that has the specified criteria lying within a pre-specified percentage from the target value.

We have collected 200 Pro/E assemblies from different student projects at the University of Maryland. A database consisting of these 200 assemblies will be used in the examples described in Sections 5.2 through 5.6. Each of the assembly scenario described in these sections was inspired by a past event that required search for an existing assembly model.

5.2 ASSEMBLY SEARCH EXAMPLE # 1

A specialized wand assembly was designed and manufactured for supporting assembly in a virtual environment. A part of this assembly broke down and it needed to be manufactured again. In order to do so, the assembly model of the wand had to be found. The student who designed the wand assembly had already left the campus. Hence a new student had to locate the wand model. It took the new student several days to locate the assembly by manually browsing through different Pro/E files on several different computers in the lab.

The above scenario was recreated to assess the performance of our system and a user familiar with our system played the role of the student in the above scenario. The user had seen the broken wand and he recalled the following three distinguishing characteristics of the assembly in order to initiate the search:

1. Approximate size of the overall wand assembly.
2. Approximate number of parts in the assembly. It was unclear whether a certain subassembly would have been modeled as a single part or a detailed subassembly. Hence, there was some ambiguity regarding this number.
3. Mating graph of four key parts.

Our system supports search based on all of the criteria. The following information was entered into the assembly search system:

Rectangular bounding box of the following dimensions: The length range is given between 6 to 8 inches, the width between 4 to 6 inches and the height between 2 and 3 inches.

Number of parts: The assembly consists of 6 parts. However, the user is not sure whether few parts directly procured from the market were modeled as separate parts or as a single part. The

overall assembly consists of 3 such parts brought from the store. Hence, the user specified a range of 4 to 6 parts. These parts do not include any fasteners.

Query mating graph: A battery casing is connected to the main body of the wand. This is in turn mated with a block which is again mated with an inertial sensor. Figure 11 shows the graph entered by the user that represents all these mating conditions. Moreover, the wand body as well as the sensor is a custom part which is specified as the corresponding node attribute.

The search was initiated using the above criteria. The database of 200 assemblies was searched and the system produced a single assembly as the output (shown in Figure 12). The system took 0.157 seconds to generate the result. In contrast, tracing a particular assembly in a database of 200 would have taken the team close to two hours.

To assess the effectiveness of our approach, we compared this example with performing the assembly search by opening assemblies in Pr/Engineer CAD system and examining them. On the average it took about 8 seconds to open an assembly in Pro/Engineer. Then it took the user additional 50 seconds to examine an assembly superficially before concluding that it was not the correct match. Finally, the user took on the average about 18 minutes to examine an assembly thoroughly by examining all the mating conditions, parts, and, other details in the assembly. Thus, in our database of 200 assemblies, in this example the user would need to thoroughly examine 2 assemblies, superficially examine another 8, and conclude by just opening the file that the remaining 190 do not satisfy desired criteria. Hence it would take them about 69 minutes to locate the correct assembly. On the other hand, the system took only fraction of a second to come up with the same output assembly. In industrial settings where thousands of assemblies will be present, it will take many hours to trace the desired assembly using browsing. This method is also error-prone; particularly when large number of assemblies does not match the search criteria, it is natural for a user to fall into the habit of wrongly rejecting an assembly without examining it properly.

5.3 ASSEMBLY SEARCH EXAMPLE # 2

A student team was looking for 6 degrees of freedom (DOF) Stewart Platform to design a seat platform for a truck simulator which would simulate realistic motion of the seat in response to the virtual truck being driven over the virtual highway. The student team identified several promising designs in Tsai's book [Tsai99]. However, they wanted to see how 3D models would look to figure out what design will be promising for their project. They knew that a previous graduate student has collected few Stewart Platform designs. So they needed to locate and study a specific design involving spherical joints. The user came up with the following distinguishing characteristics of the assembly:

1. Relationships among the joints connected to the main platform.
2. The total number of different types of joints present in the platform.

As our system supports search on both these criteria, following information was entered into the assembly search system:

Joint relationships: 6 pairs of spherical-prismatic joints are specified on the main platform, leaving the angles between all the joints pairs as unknown entities. This is done because 6 such pairs are always attached to any Stewart Platform, irrespective of its particular configuration in order to impart all the necessary degrees of freedom.

Total number of joints: A target value of 18 is assigned to this field as a total number of 18 joints (6 spherical joints linked to 6 prismatic joints which are again linked to 6 other spherical joints) are always present in any such platform.

The system retrieved one assembly as the output (shown in Figure 13). It satisfies the criteria specified by the user and he can study it for simulating motion of the truck seat in a virtual environment. It took 0.157 seconds to generate the results in this case.

5.4 ASSEMBLY SEARCH EXAMPLE # 3

A student team was designing a new robot. They had studied reports written by students in the previous years and have found a circuit design that looked very promising. Hence they wanted to construct the controller board based on this proven circuit design. The student team was in the process of modeling the assembly of their robot structure. In order to assess the location of center of gravity and the overall mass of their design, the student team needed to include a detailed model of the controller board sub-assembly. Rather than creating this sub-assembly from scratch, they wanted to reuse the sub-assembly created by the previous team. The user has seen the photograph of the old controller board assembly and the circuit design. So he came up with the following distinguishing characteristics of the assembly:

1. Approximate number of components.
2. Approximate overall size.
3. Mating graph of a key component (e.g., main circuit board) with respect to other smaller parts.

In this case, the following information was entered into the assembly search system:

Number of parts: A range of 15 to 25 is given as reasonably large number of components is expected to be present in the assembly. No fasteners are included within this part range count.

Rectangular bounding box of the following dimensions: The length range is given between 3 to 5 inches, the width between 1.5 to 4 inches and the height between 0.25 and 2 inches.

Query mating graph: Two nodes, representing two custom parts, namely the main circuit board and an oscillator, are connected by an edge. However, the unique, distinguishing feature of this assembly is specified in the form of the high degree (13) that is assigned to the node representing the main board. This degree indicates that the student expects the matching database assembly to contain at least 13 other parts (in addition to the oscillator) that are connected to the main board.

After searching the database, the system returned four assemblies as the output (shown in Figure 14). The system took 0.156 seconds to generate the results.

5.5 ASSEMBLY SEARCH EXAMPLE # 4

A student team was designing a new gait for four-legged locomotion which must conserve total energy without exceeding the torque constraints on the motor. They wanted to simulate the new gait. In order to setup the simulation, they needed the assembly model of a four-legged robot. They knew that in previous years, other student teams had designed four-legged robots. So they wanted to search for a suitable robot. Since the gait required a particular configuration of leg joints, the user came up with the following distinguishing characteristics of the assembly in order to initiate the search:

1. Approximate number of parts.
2. Joint relationships expected in the main body.

Since our system supports search on all of these criteria, following information was entered into the assembly search system:

Number of parts: A range of 10 to 100 is given as large number of components (links attached by joints) should be present in the assembly.

Joint relationships: 6 pairs of revolute-revolute (RR) joints are specified on the main body or platform, leaving the angles between all the joints pairs as unknown entities. This is based on the sound rationale that at least 6 RR links need to be attached to the main body in order to impart all the required degrees of freedom (flexibility) to the robot structure. However, some of these links may be parallel to each other, whereas others may be perpendicular. Hence, joint angles are not explicitly specified.

The system retrieved two assemblies as the output (shown in Figure 15). Figure 15 (a) shows a lizard robot containing servo motors, whereas Figure 15 (b) represents an elephant robot. Both assemblies satisfy the criteria specified by the user and it is up to him to select the more suitable one. It took 0.172 seconds to generate the results in this case.

5.6 ASSEMBLY SEARCH EXAMPLE # 5

A student team needed to mount a shaft using two bearings which would operate under highly varying temperature conditions. Hence the length of the shaft would fluctuate during its operation. Completely constraining the bearings would lead to a very high load on them and reduce their life considerably. Therefore, the bearings had to be carefully mounted on the shaft and provided with adequate degrees of freedom. The student team had no prior experience in mounting bearings. Hence, they wanted to study existing assemblies that used ball bearings to learn implicitly embedded Design for Assembly rules associated with mounting of bearings. The user came up with the following distinguishing characteristics to identify assemblies of interest:

1. Presence of two bearings which are standard parts.
2. Mating conditions associated with the bearings (the shaft design partially dictates at least few specific mating conditions).

In this case, the following information was entered into the assembly search system:

Constituent parts: Two bearings are present in the assembly, both of which must belong to the category of standard parts.

Mating conditions: A shaft is connected to two bearings (each of which has category standard and type bearing as two of the corresponding node attributes). Moreover, the shaft is also attached to a housing structure. No other information is provided regarding the shaft or housing parts.

After searching the database, our system returned two different shaft designs (shown in Figure 16) as the output. Figure 16 (a) shows a design where one of the bearings is unconstrained or has an additional degree of freedom. This will provide greater thermal stress resistance to the shaft, thereby enhancing its lifetime. On the other hand, the bearings in the design shown in Figure 16 (b) are completely constrained. Hence, the user should select the first design. The system took 0.453 seconds to generate the results.

6 CONCLUSIONS

This paper describes a framework for performing content-based search for mechanical assemblies. Specific contributions of this paper are described below:

- We have identified a comprehensive set of search criteria to define assembly search queries. As illustrated by five different example scenarios, the identified search criteria allow the users to define a wide variety of search queries. These criteria were selected after a careful consideration of the search algorithm and ease of use. These criteria expand the search options available to users in a typical PDM system. Our proposed scheme also preserves the search options available to users in current PDM systems.

- We have categorized identified search criteria into logical groupings to make it convenient for users to locate the desired criteria.

- We have developed a new algorithm for performing mating graph-based searches. This new algorithm allows checking of compatibility of a query graph containing wild card entries with a fully specified mating graph of an existing assembly. Our algorithm will work well when the number of node types is larger than (1) the number of nodes in the database mating graph, and (2) the number of nodes in the query mating graph. Both of these conditions are easily met in our application. We have characterized the computational performance of this algorithm and shown that this algorithm works well for typical small query graphs encountered in the assembly search.

- We have successfully developed a proof of the concept system to demonstrate the feasibility of our ideas and algorithms. We have shown using five real-life inspired examples that the system is capable of meeting all the system requirements outlined in Section 2. Most of the search criteria have been put to use in one or more of the examples.

We expect that the system described in this paper will serve two purposes. First, it will allow designers to reuse existing assemblies by giving them a tool to identify assemblies with the desired characteristics. Second, it will provide designers access to the DFMA knowledge contained in the assembly database, and hence transfer best practices to new designs.

The ideas described in this paper are consistent with modern PDM systems and can be easily implemented inside such a system to expand its typical capability. The current implementation only provides a tool to extract assembly signature from the Pro/Engineer CAD system. The search system will need to be extended to search assemblies from other CAD systems such as Unigraphics and SolidWorks by providing the tools to extract assembly signatures.

An assembly can be viewed as a collection of parts designed to fulfill one or more functions. The function is thus an important characteristic of an assembly. However, it is not usually explicitly stored in CAD files. Often, it cannot be inferred from the geometric characteristics of the assembly either. Hence, designers cannot easily search for an assembly fulfilling a particular function. Significant additional work is needed to expand the capability of our system to include this additional search criterion.

The proposed search method works only on the basis of the geometric characteristics and relationships present in the assembly. This search is thus not applicable to assemblies that have characteristics from other domains of engineering like electromechanical or electrochemical engineering. The search also works on the assumption that the designer explicitly defines joints and mating conditions and such relations can be extracted to form a signature of the assembly. If this data is not available, the assembly cannot be included in the search and this may result in false negatives during the search process.

ACKNOWLEDGEMENTS

This research is supported in part by the Center for Energetic Concepts Development at the University of Maryland and National Institute of Standards and Technology's (NIST) Manufacturing System Integration Division. We thank Maxim Schwartz for developing the assembly viewer.

DISCLAIMER

Any commercial product or company name in this paper is given for informational purposes only. Their use does not imply recommendation or endorsement by NIST.

REFERENCES

[Anan96] R. Anantha, G.A. Kramer, and R.H. Crawford. Assembly modelling by geometric constraint satisfaction. *Computer-Aided Design* 28 (9): 707-722, 1996.

[Bohm05] M. R. Bohm, R. B. Stone, and S. Szykman. Enhancing Virtual Product Representations for Advanced Design Repository Systems. *Journal of Computing and Information Science in Engineering*, 5(4): 360-372, 2005.

[Boos06] Boost Graph Library. http://www.boost.org/libs/graph/doc/

[Brun00] G. Brunetti, and B. Golob. A feature-based approach towards an integrated product model including conceptual design information. *Computer-Aided Design*, 32 (14): 877-887, 2000

[Card03] A. Cardone, S.K. Gupta, and M.V. Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *Journal of Computing and Information Science in Engineering*, 3(2): 109-118, 2003.

[Card06a] A. Cardone, S.K. Gupta, A. Deshmukh, and M. Karnik. Machining feature-based similarity assessment algorithms for prismatic machined parts. *Computer-Aided Design*, 38(9): 954-972, 2006.

[Card06b] A. Cardone and S.K. Gupta. Shape Similarity Assessment Based on Face Alignment using Attributed Applied Vectors. *Computer-Aided Design & Applications,* 3(5): 645-654, 2006.

[Cici01] V. Cicirello, and W.C. Regli. Machining feature-based comparisons of mechanical parts. In Proceedings of the *International Conference on Shape Modeling & Applications*, pp. 176-185, Genoa, Italy, 2001

[Cord04] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 26(10): 1367-1372, 2004.

[Ense00] P. Enser. Visual image retrieval: seeking the alliance of concept-based and content-based paradigms. *Journal of Information Science*, 26(4):199-210, 2000.

[Fort96] S. Fortin. Graph isomorphism problem. Technical Report 96-20, University of Alberta, Edmonton, Alberta, Canada, 1996.

[Fuch00] F. Fuchs, and H. Le-Men. Efficient Subgraph Isomorphism with 'A Priori' Knowledge. *Lecture Notes in Computer Science*, 1876: 427-436, 2000.

[Gupt01] S.K. Gupta, C.J. Paredis, R. Sinha, and P.F. Brown. Intelligent assembly modeling and simulation. *Assembly Automation*, 21(3): 215-235, 2001

[Gupt06] S.K. Gupta, A. Cardone, and A. Deshmukh. Content-Based Search Techniques for Searching CAD Databases. *Computer-Aided Design & Applications,* 3(6): 811-819, 2006.

[Jung04] H. W. Jung, S. G. Kim, and C. S. Chung. Measuring software product quality: ISO/IEC 9126. *IEEE Software*, 21(5): 88-92, 2004.

[Kope03a] J. B. Kopena, and W. C. Regli. Functional Modeling of Engineering designs for the Semantic Web. *IEEE Data Engineering Bulletin*, 24(4): 55-62, 2003.

[Kope03b] J. B. Kopena, and W. C. Regli. Design Repositories for the Semantic Web with Description-Logic Enabled Services. *1st International Semantic Web and Databases Workshop (co-located with VLDB 2003)*, pp. 349-356, Berlin, Germany, 2003.

[Kope03c] J. B. Kopena, and W. C. Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web. *IEEE Intelligent Systems*, 18(3): 74-77, 2003

[Lee85] K. Lee, and D. C. Gossard. An hierarchical data structure for representing assemblies: part 1. *Computer-Aided Design*, 17(1): 15-19, 1985.

[Li04] Z. Li, M. Liu, and K. Ramani. Review of product information retrieval: representation & indexing. In Proceedings of the *ASME Design Engineering Technical Conferences*, Paper No. DETC2004-57749, Salt Lake City, UT, 2004.

[Moll93] E. Molloy, H. Yang, and J. Browne. Feature-based modelling in design for assembly. *International Journal of Computer Integrated Manufacturing*, 6(1-2): 119-125, 1993.

[Niel94] J. Nielsen, and R. L. Mack. *Usability Inspection Methods*. John Wiley & Sons, New York, NY, 1994.

[Rame01] M.M. Ramesh, D.Y. Hoi, and D. Dutta. Feature-based shape similarity measurement for retrieval of mechanical parts. *Journal of Computing and Information Science in Engineering*, 1(3): 245-256, 2001.

[Rein77] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1977.

[Regl00] W. C. Regli, and V. A. Cicirello. Managing digital libraries for computer-aided design. *Computer-Aided Design*, 32(2):119-132, 2000.

[Shah93] J. J. Shah, and M. T. Rogers. Assembly modeling as an extension of feature-based design. *Research in Engineering Design*, 5(3-4): 218-237, 1993.

[Tsai99] L.W. Tsai. *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. Wiley Interscience, New York, NY, 1999.

[Tsai01] L.W. Tsai. *Mechanism Design: Enumeration of Kinematic Structures According to Function.* CRC Press, Boca Raton, FL, 2001.

[Ullm76] J.R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the Association for Computing Machinery*, 23(1): 31-42, 1976.

[Yu04] E. Yu, and X. Wang. A Subgraph Isomorphism Algorithm Based on Hopfield Neural Network. *Lecture Notes in Computer Science*, 3173: 436-441, 2004.

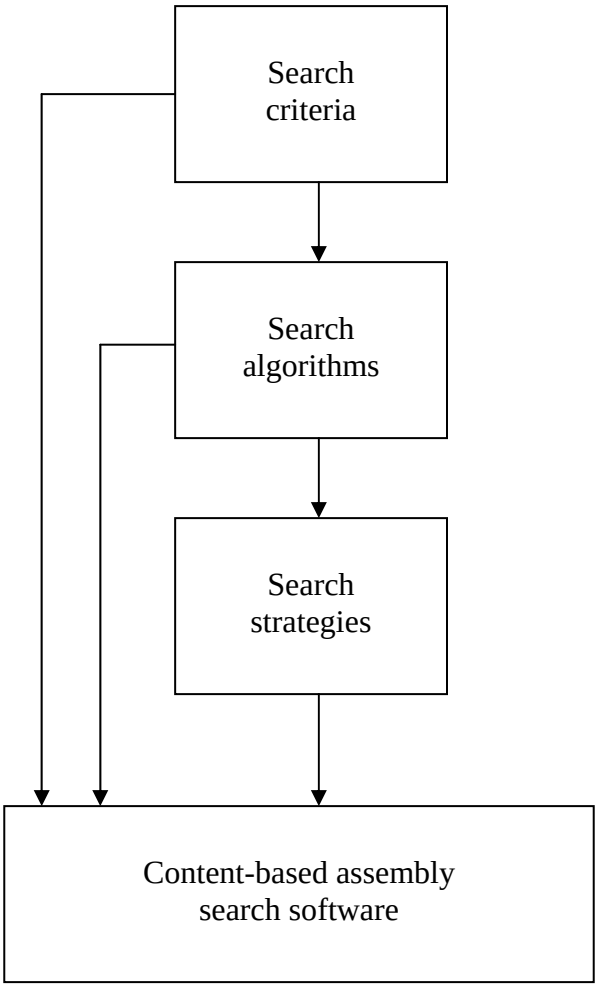


Figure 1: Main components of content-based assembly search

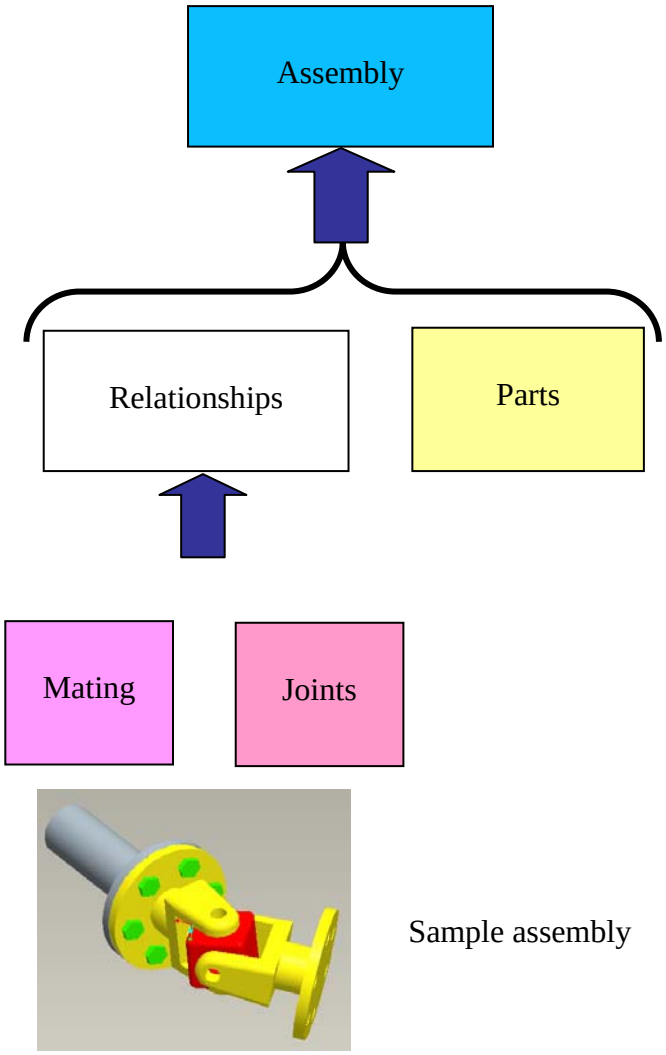


Figure 2: Organization of assembly search criteria

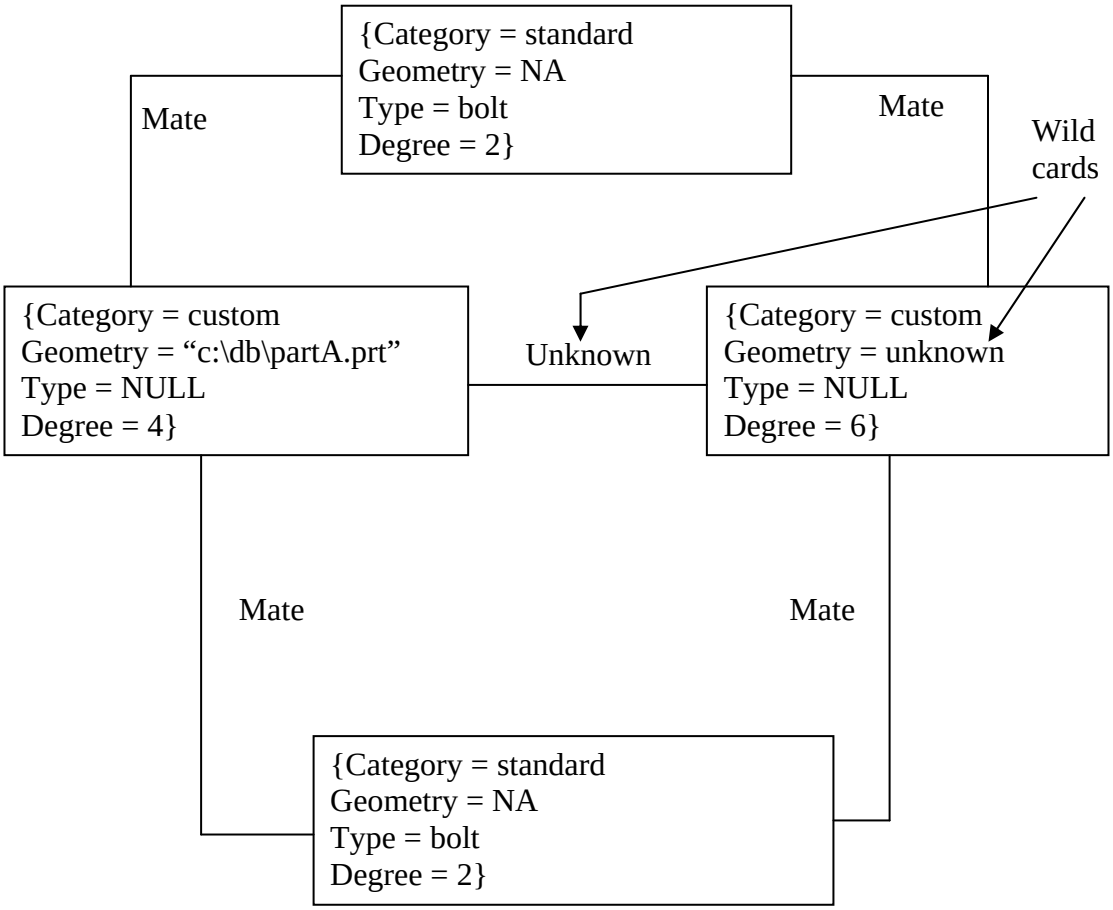
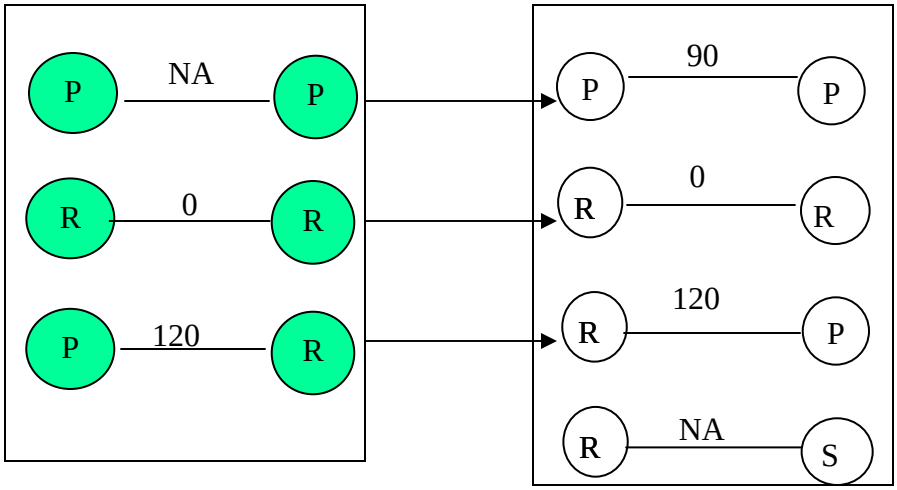


Figure 3: Query mating graph containing two wild card entries

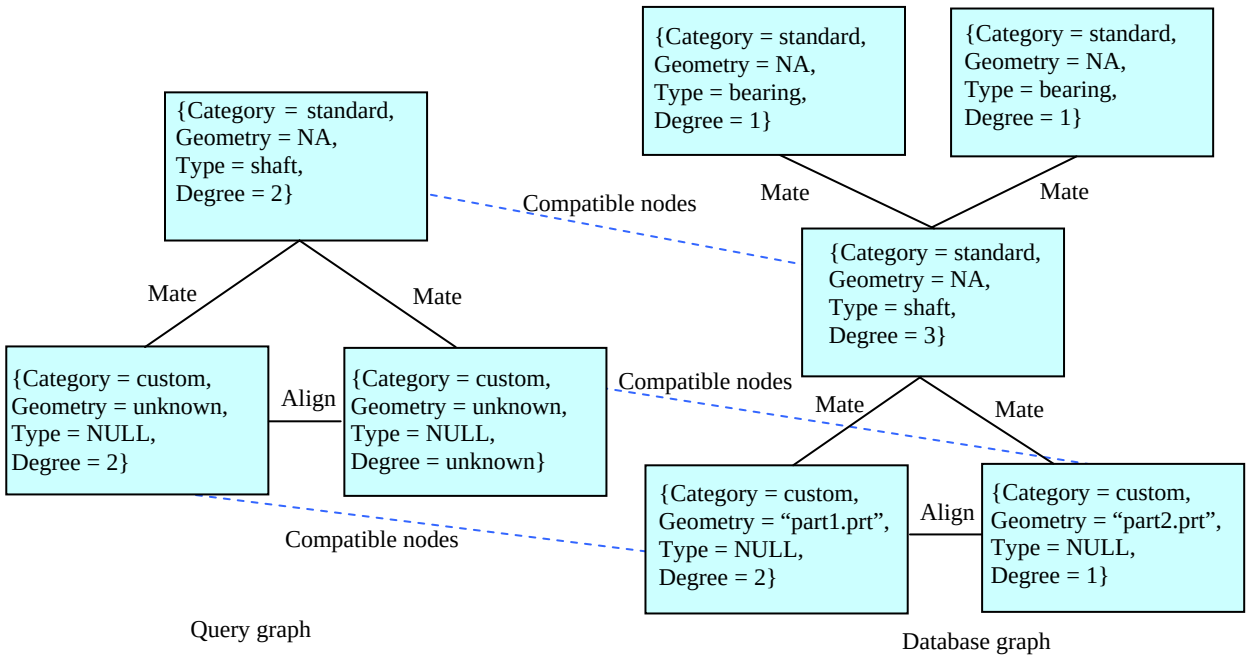


Query: 3 relationships

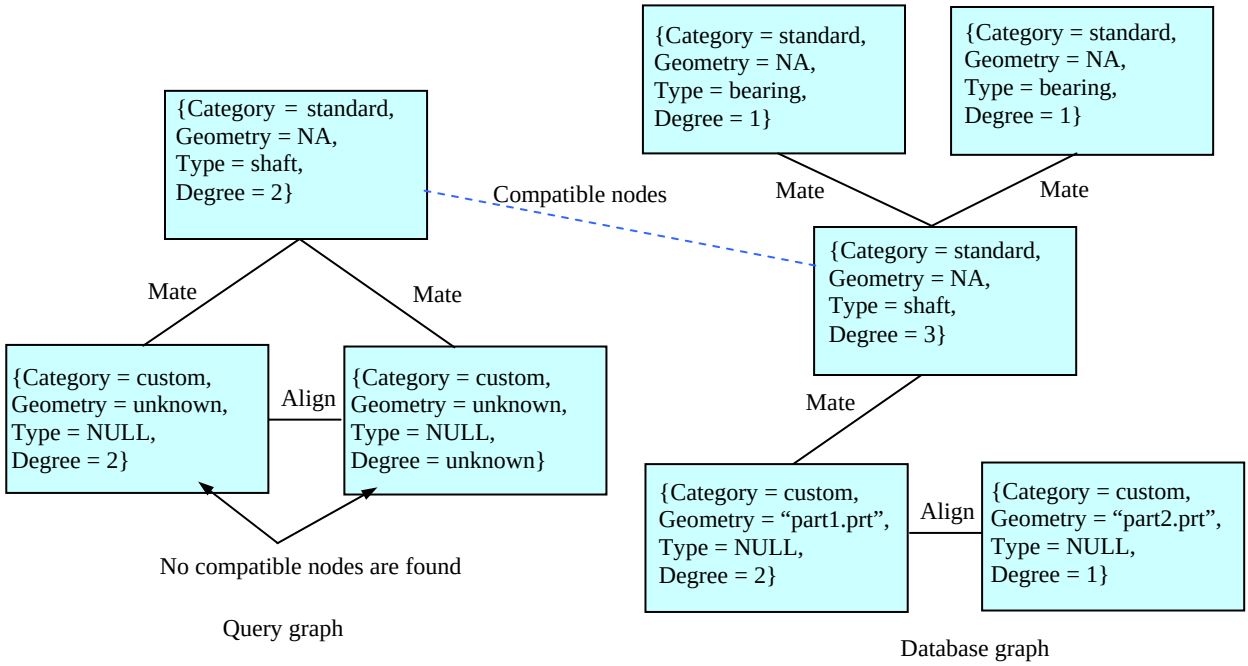
Relationships in a database assembly

P: Prismatic joint
T: Revolute joint

Figure 4: Matching of query and database assemblies based on joint relationships



(a) Example of compatible graphs



(b) Example of incompatible graphs

Figure 5: Examples of compatible and incompatible graphs

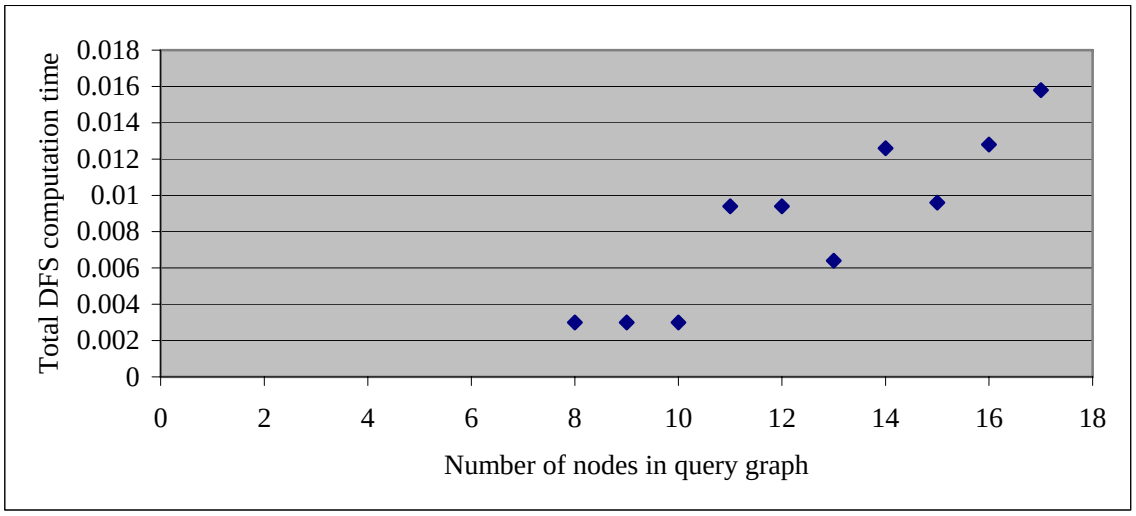


Figure 6: Total DFS computation time (in s) versus number of nodes in query graph

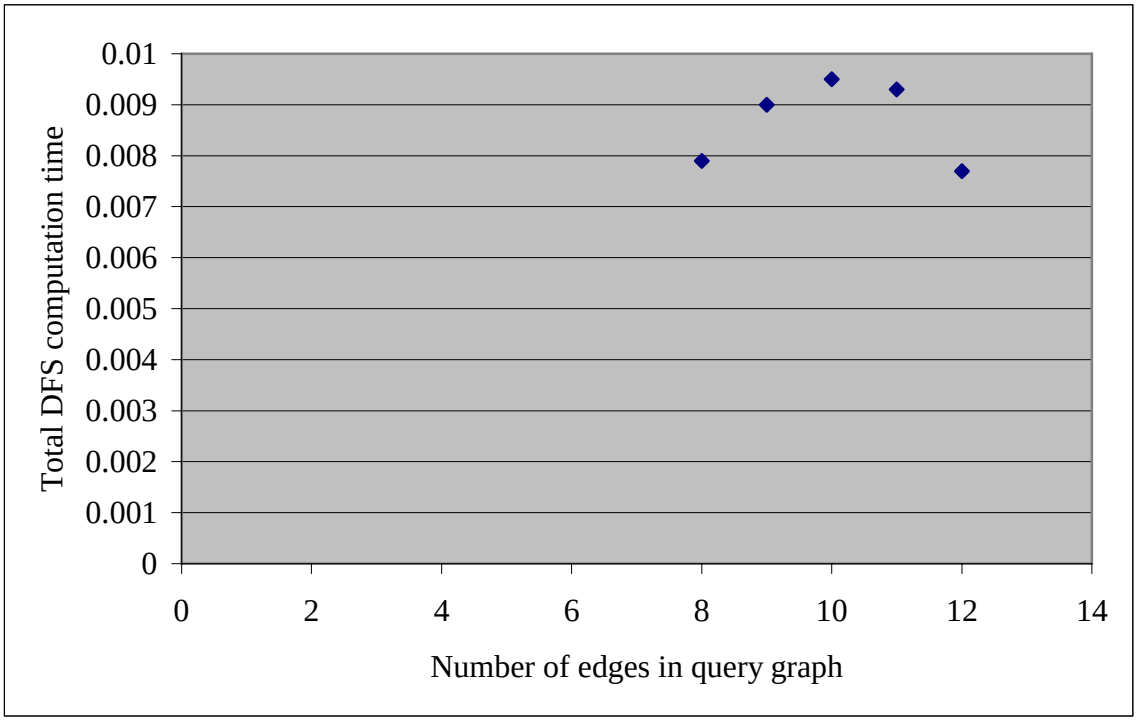


Figure 7: Total DFS computation time (in s) versus number of edges in query graph

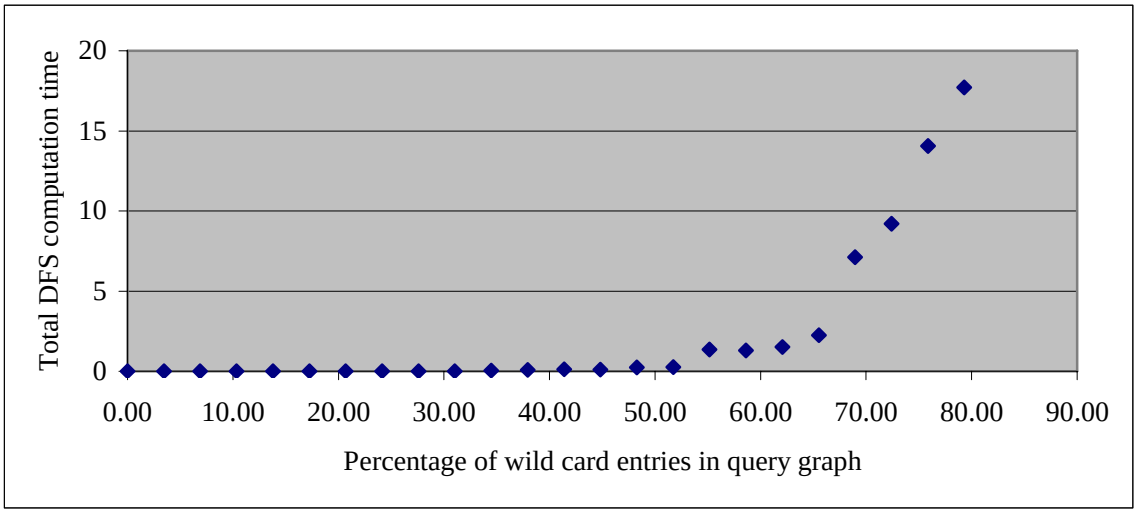


Figure 8: Total DFS computation time (in s) versus percentage of wild card entries in query graph

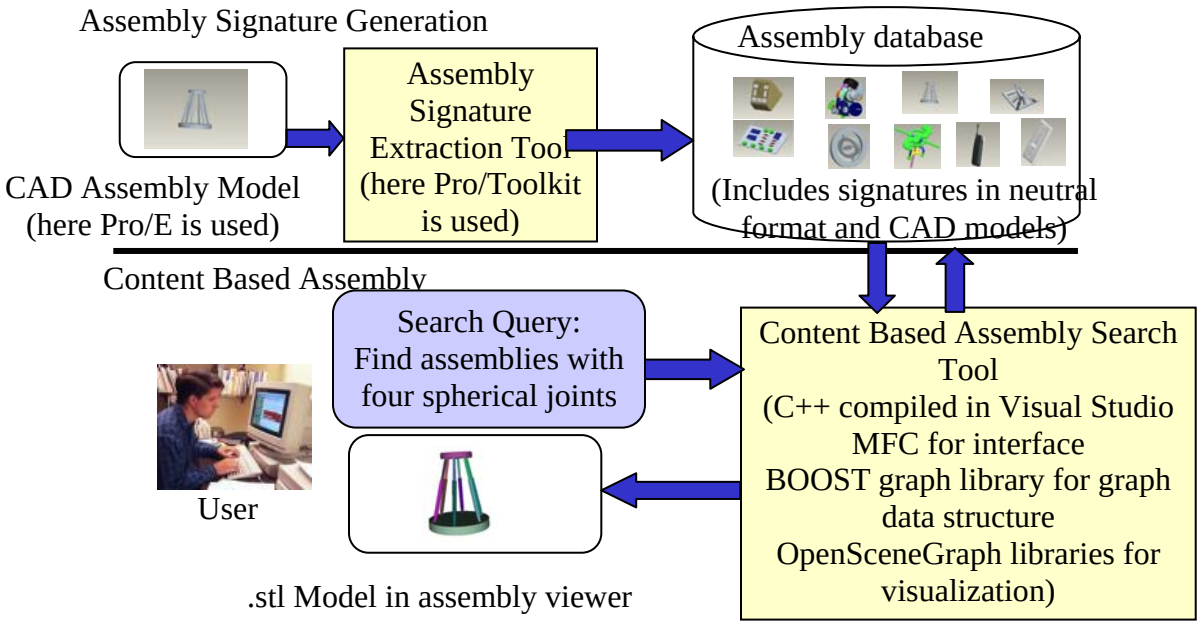


Figure 9: Overview of assembly search system

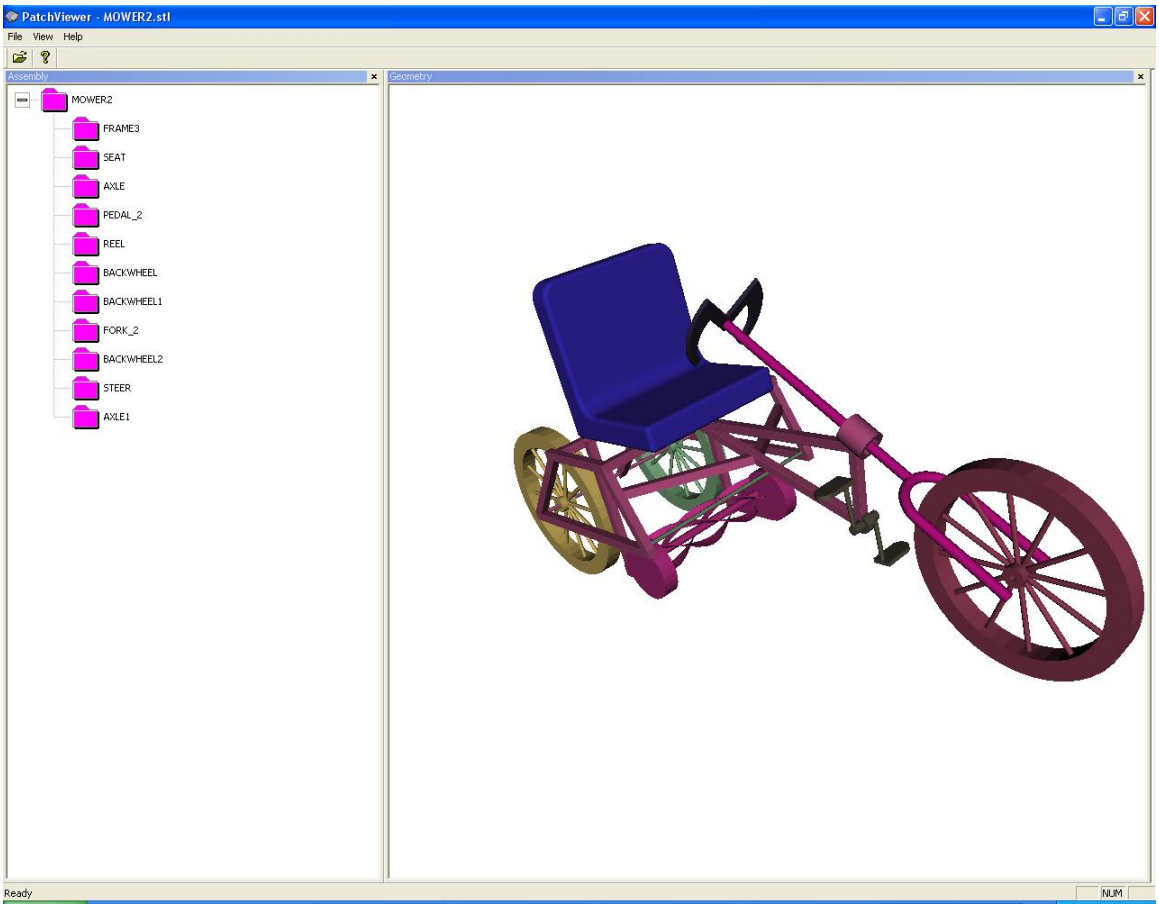


Figure 10: Assembly search results viewer showing an assembly

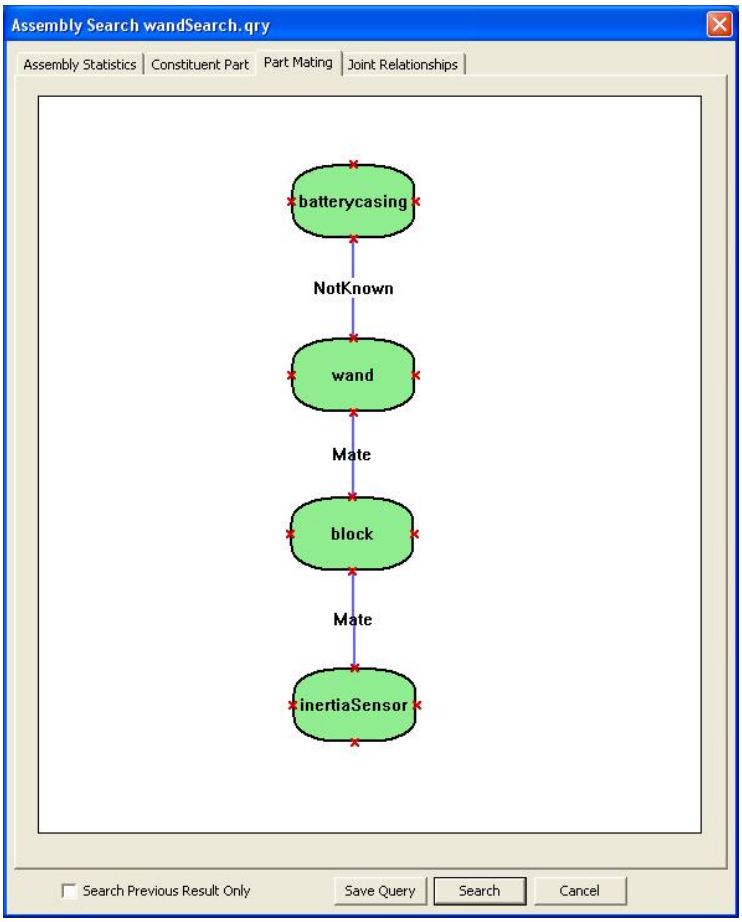
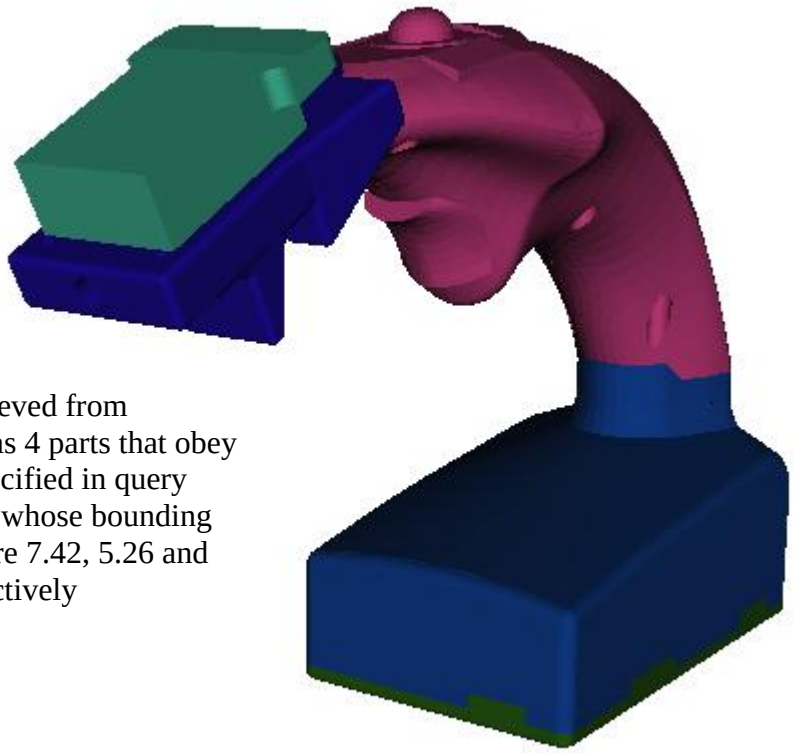


Figure 11: Query mating graph defined by the user to search for wand assembly in Example #1



An assembly retrieved from database which has 4 parts that obey the conditions specified in query mating graph and whose bounding box dimensions are 7.42, 5.26 and 2.43 inches respectively

Figure 12: Wand assembly returned by assembly search system in Example #1

An assembly retrieved by search system that contains 6 pairs of spherical-revolute joints

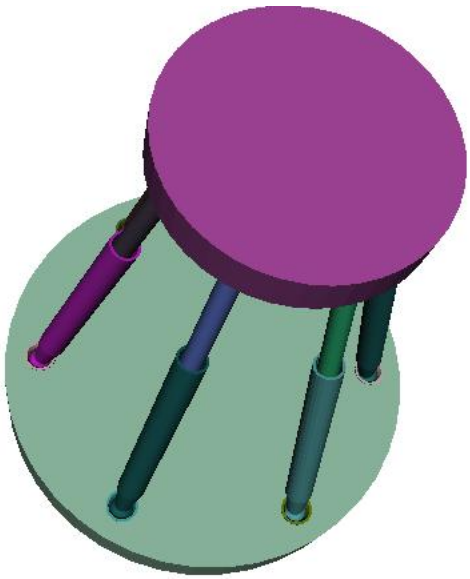
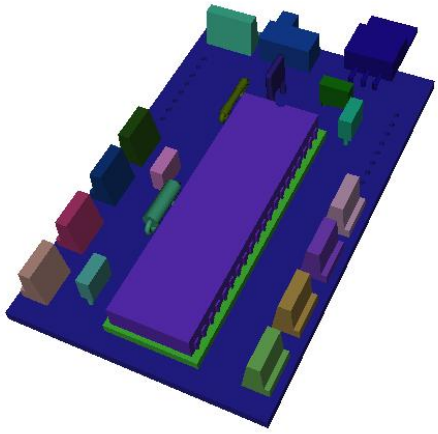
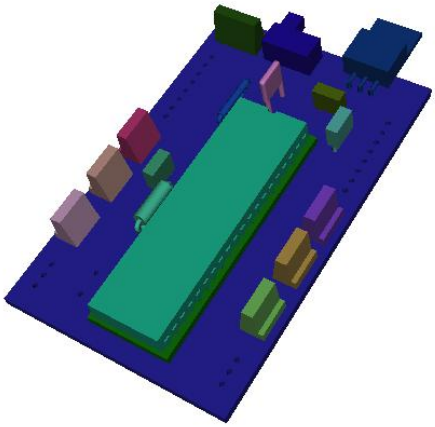


Figure 13: Stewart platform assembly returned by assembly search system in Example #2

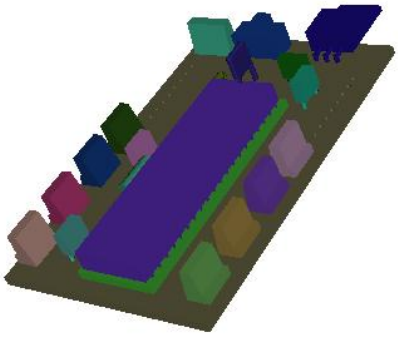


19 parts attached to main circuit board

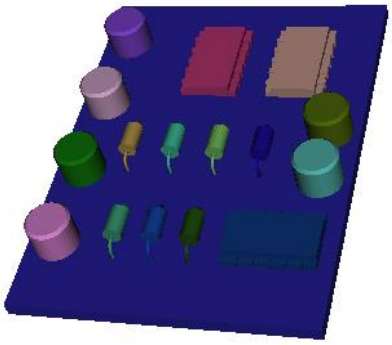
Four assemblies retrieved from database, each of whose bounding box dimensions are within the following range: length is between 3 to 5 inches, width is between 1.5 to 4 inches, and height between 0.25 and 2 inches.



16 parts attached to main circuit board

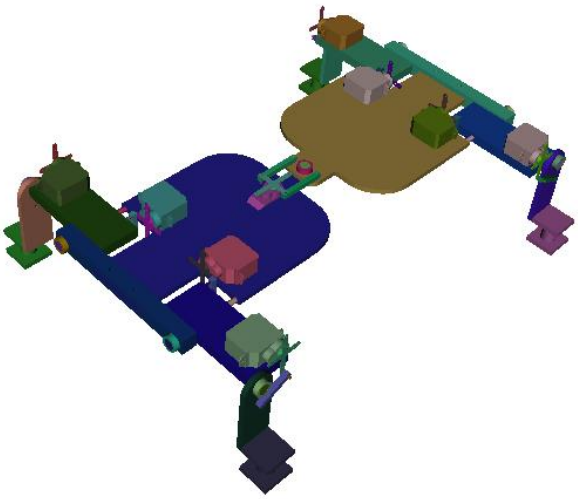


19 parts attached to main circuit board



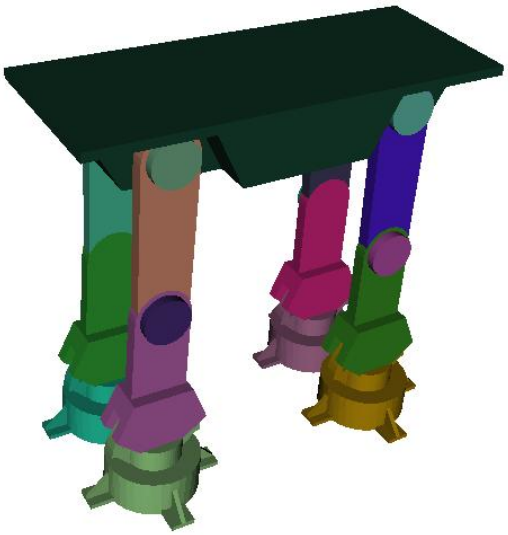
16 parts attached to main circuit board

Figure 14: Four PCBs returned by assembly search system in Example #3



Assembly retrieved by search system
that contains 86 parts and 25 RR links

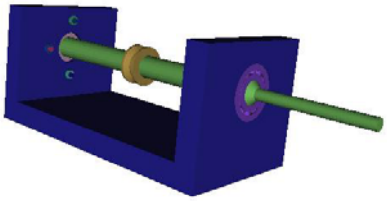
(a) Lizard-inspired robot



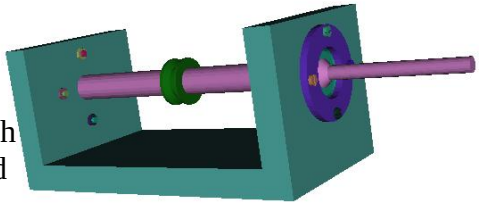
Assembly retrieved by search system
that contains 21 parts and 8 RR links

(b) Elephant-inspired robot

Figure 15: Two bio-inspired robots returned by assembly search system in Example #4



Two assemblies retrieved from database, each of which contains 2 bearings mounted on a shaft attached to a housing structure



(a) Partially constrained shaft assembly

(b) Fully constrained shaft assembly

Figure 16: Two bearing mounted shaft assemblies returned by assembly search system in Example #5