# Abstract

Title of Thesis: CONTENT BASED SEARCH OF MECHANICAL ASSEMBLIES

Degree Candidate: Abhijit Deshmukh

Degree and Year: Master of Science, 2006

Thesis directed by: Associate Professor Satyandra K. Gupta

Department of Mechanical Engineering and Institute of Systems Research

The increased use of 3D CAD systems by product development organizations has resulted in large databases of assemblies; this explosion of assembly data will continue in the future. Currently, there are no effective content-based techniques to search these databases. 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 (DFMA) knowledge is embedded in existing assemblies. Therefore a capability to locate existing assemblies and examine them can be used as a learning tool by the designers to learn from the existing assembly designs and hence transfer the best DFMA practices to new designers.

This thesis describes a system for performing content-based searches on assembly databases. It lists the templates identified for comprehensive search definitions and describes algorithms to perform content-based searches for mechanical assemblies. The characteristics of mechanical assemblies were identified and categorized based on their similarity and computational complexity to perform comparison. The characteristics were extracted from the CAD data to prepare a CAD independent signature of the assembly. The search methodology consists of exact and approximate string matching, number matching and computing graph compatibility. Various research groups have solved the former two problems. This thesis describes a new algorithm to solve graph compatibility problem using branch and bound search. The performance of this algorithm has been experimentally characterized using randomly generated graphs.

This search software provides a CAD format independent tool to perform content based search of assemblies based on the form of assemblies. The capabilities of the search software have been illustrated in this thesis through several examples. This search tool can contribute to significantly reduce the design time and reuse of the knowledge in existing designs.

# CONTENT BASED SEARCH OF MECHANICAL ASSEMBLIES

**by**

**Abhijit Deshmukh**

**Thesis submitted to the Faculty of the Graduate School of the**

**University of Maryland, College Park in partial fulfillment**

**of the requirements for the degree of**

**Master of Science**

**2006**

## DEDICATION

To my entire family and all those who contributed immensely to this work and made it possible

# ACKNOWLEDGEMENTS

I would like to sincerely thank my academic advisor Dr. Satyandra K. Gupta for giving me an opportunity to conduct research under his supervision. I have a firm belief that the knowledge gained during this research would enable me to fulfill my professional responsibilities henceforth. His exemplary perseverance and focused approach to any problem would help me beyond the realms of professional life. I would also like to thank Dr. Linda Schmidt and Dr. David Mount for readily accepting to be on my thesis committee.

I am thankful to the Department of Mechanical Engineering and the Institute of Systems Research for the support during this research. I also want to express my gratitude to Ashis Banerjee for help in the algorithms for graph compatibility and providing me a random graph generator. A special note of thanks is due for Ashis for his encouragement. I would like to thank Maxim Schwartz for providing the assembly search tool. I would like thank Alok Priyadarshi, Mukul Karnik and Maxim Schwartz for the valuable suggestions during the implementation of this system. I would also like to thank Arne Gfell for functions to draw graphs and XML Parser and Changxin for integrating the code to draw graphs with the code for assembly search.

I would also like to thank my friends: Sameer Kibey, Saurabh Srivastava, Arvind Ananthnarayanan, Kapil Sahasrabudhe, Shirsho Sengupta, Suresh Kumar Santhanavannanna, Aditya Kalyanpur and many others for their valuable support and suggestions during my research.

# TABLE OF CONTENTS

## LIST OF FIGURES

**Chapter 1**

# INTRODUCTION

This chapter is arranged in the following manner: Section 1.1 discusses the mechanical assemblies in CAD software systems, Section 1.2 describes the motivation behind the research for assembly search system, Section 1.3 describes the issues involved in the research and Section 1.4 describes the outline of the thesis. Most of the research in this thesis is based on the work reported in [Desh05, Gupt06].

## 1.1 Mechanical Assemblies in CAD

Over the last ten years, 3D CAD systems have become very popular in the industry. These CAD systems are being used to generate 3D models of parts and assemblies. These models are used as a basis for engineering analysis and generating manufacturing plans. 3D models also allow virtual prototyping and hence reduce 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. Specially, design, manufacturing and service engineers are expected to greatly benefit from these databases. Design engineers can access the designs of parts and assemblies in the database to design a product for a similar application. Manufacturing engineers can use these databases to find the manufacturing plans and vendors to manufacture parts and assemblies. Service engineers can use the strategy to disassemble and assemble the products for maintenance and servicing. These databases are updated with the latest versions of parts and assemblies and hence significantly improve 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.

The mechanical assemblies consist of either the pointers to or a copy of the geometry of the constituent parts. An assembly in a CAD system can be made of individual parts or subassemblies. The subassemblies are made of its constituent parts. The constituent parts and sub assemblies are represented in a tree structure that represents the bill of materials of the assembly. The constituent parts and subassemblies are placed in specific position using part mating conditions. The assemblies are virtual representation of a product or an important part of a product. Some products have articulations. The articulation is shown in assembly by joints. These joints can be simulated to show the movement of different parts of the assembly. Before a product is manufactured, different analysis are carried out on the parts and assembly. Some examples of such type of analysis are manufacturability, strength and motion analysis. Often the results of such analysis including the product manufacturing (PMI) data are stored with the assemblies. The mechanical assemblies thus contain much more information about the products of an organization as compared to CAD files that only contain the geometry of a part in the product.

The assembly databases, besides supporting downstream manufacturing and service operations, can be very useful during the design phase as well. There are two main uses of 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 of existing designs is beneficial from many different perspectives. It reduces design time by eliminating the need for modeling and analysis for the assembly being reused. Furthermore, the existing assembly is already tested and has an established manufacturing plan. This further reduces the product development time and cost. Sharing assemblies across multiple product lines also allows a company to take advantage of the economy of scale. The design of universal joint shown in the figure 1.1 can be reused to design another universal joint.

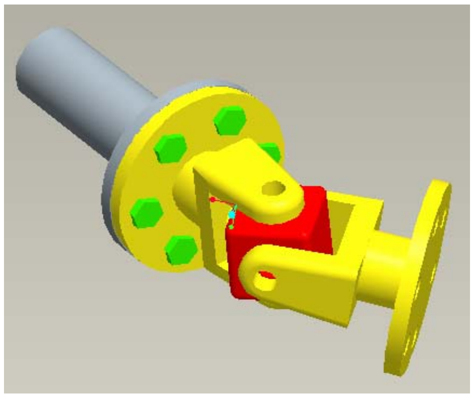


Figure 1.1: A universal joint – reuse of existing design

- The second possible usage is to provide access to existing design knowledge. Designing assemblies requires considerable effort. Creating good assembly designs require thoughtful analysis and careful application of Design for Manufacturing and Assembly (DFMA) principles. The design thumb rules used in an organization are embedded in the design. These rules include the tweaking of design to suit the manufacturing capabilities in the organization and its vendors. New designers can adopt and copy successful design templates. Moreover, once designers manage to find an assembly with the desired characteristics, they can also access associated data such as cost, reliability, and failure reports. The camera frame assembly shown in the figure 1.2 can be used to access the knowledge about an assembly that fit together tightly.

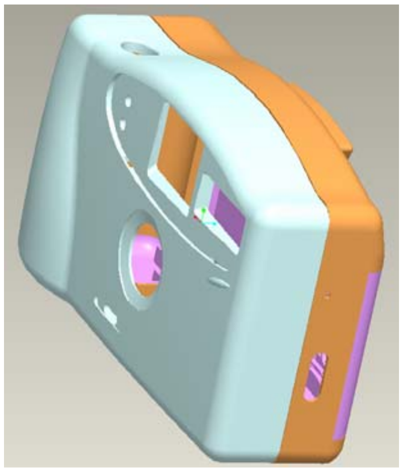


Figure 1.2: A camera frame – use of design knowledge

## 1.2 Motivation

This section describes the motivation behind this work.

### 1.2.1 Need for assembly search system

Saaksvuori reports that up to 70 percent of a designers time can be saved if the existing knowledge base of an organization can be reused for new designs. [Saak04] It is thus very important to have the capability to search the database based on any characteristics for a desired assembly. Currently, content-based search tools do not exist for searching assemblies based on the specified criteria. Therefore, designers locate assemblies by combining the text based and part search methods and manually opening various files and browsing through them using a CAD system. This is a highly inefficient use of designer’s time, and becomes a serious problem as the numbers of assemblies in the database grow. This also requires that designers should have access to the CAD software. 3D CAD software are costly and it would be helpful to search database without requiring to access CAD software. If a designer can access the information of an assembly and study the design without using CAD, it would result in significant cost reduction.

### 1.2.2 Existing search methods and their limitations

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 contents is adopted. However, developing and deploying a content-based naming convention is impractical in many large organizations. Many organizations have manufacturing plants located across different geographical locations. The merger of two different organizations in different geographical locations can also lead to two completely different naming conventions that are individually sufficient to cater to the needs of the different manufacturing units but cannot be used as search criteria in a single assembly search system. In such case, information about product designs cannot be used by designers in two different manufacturing units. A newly developed convention cannot be used to search the legacy data in the organization. Any organization usually has very large quantity of legacy data that makes it very difficult to change or implement naming conventions followed in the organization. Another way is to attach text notations to assemblies and store them in a Product Data Management (PDM) database. This scheme only provides limited search capabilities and has a limited discrimination power. Moreover, assemblies need to be manually annotated. The text based search cannot be used to define all attributes of the mechanical assembly. Moreover, manual annotations introduce human errors that cannot be avoided thereby reducing the accuracy of a tool that searches for assemblies based on these text attributes. In the recent past, several geometry-based search tools have emerged. However, these tools, although useful for part searches, are not very effective for assemblies. They can only account for the overall shape of the assembly and cannot account for relationships and structure that exist in assemblies. The overall shape may differ for an articulated assembly. For example, the four bar link mechanism shown in the figure 1.3 occupies different volume in different positions. Only text based search tools and geometry based search tools are clearly insufficient to search for assemblies. This research has been started to provide a content based assembly search tool for designers an organization.

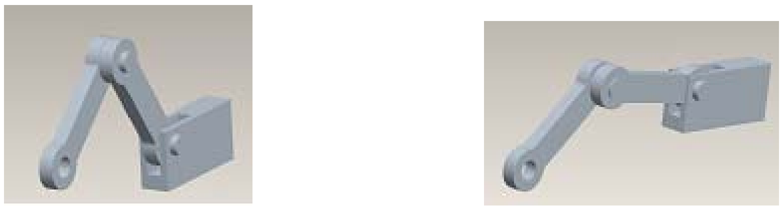


Figure 1.3: Slider crank mechanism occupying different volumes due to different position of relative parts

## 1.3 Research Issues

### 1.3.1 Desired characteristics of software search system

The goal of this research is to develop a content-based assembly search system for searching assemblies from a database of existing assemblies based on different characteristics. The characteristics used by the system to search the database need to be extensive and also include most of the characteristics of a typical assembly. Hence, the system will need to support a comprehensive list of characteristics of assemblies based on which the user can define a search. The characteristics included in the system are enumerated in subsequent chapters. The system should be flexible and allow the user to search based on any combination of the characteristics. It should also handle cases that result in too few or too many search results. Thus, if the search system results are too few then the user should be able to lower the constraints (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 perform iterative refinement. This is achieved by constraining the search by including more assembly characteristics in the search, then performing search and again refining search definition. This iterative refinement is very effective in producing the right number of search results. At any time in the search, the user should be able to exclude any assembly from further search. Finally, the system should have an easy-to-use interface and should be efficient so as to locate assemblies from a database in few seconds. The objectives of this research are summarized in the figure 1.4.

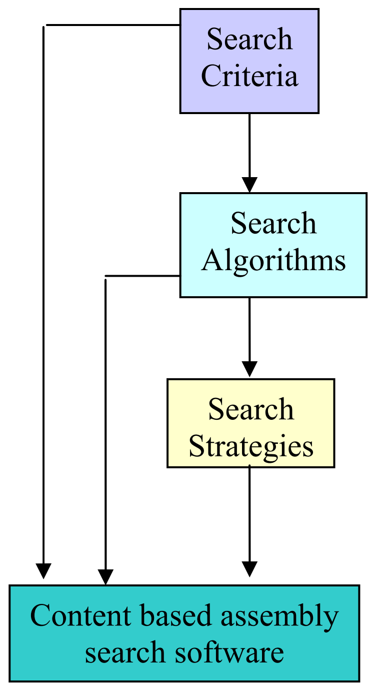


Figure 1.4: Objectives of research

The thesis describes a system for performing content-based searches on an assembly database. It is followed by the description of the templates for comprehensive search definitions that have been identified after studying various assembly models used in modern CAD systems. It also describes the algorithms developed to perform content-based searches for mechanical assemblies based on these search definition templates. These algorithms have been implemented in a system. The thesis also has illustrations of the possible usages of the prototype system with some examples.

### 1.3.2 Identifying and extracting characteristics of assemblies for search

The initial part of research is to identify a comprehensive set of characteristics of the assembly. These characteristics should cover all possible information about an assembly that can be used by a designer to search for the assembly. To build a list of all the characteristics of an assembly on which search can be performed, an extensive review of existing CAD systems and literature in the assembly modeling field [Anan96, Boot94, Brun00, DeFa87, Home91, Khos89, Lee85, Lee93, Moll93, Shah93] was performed. We decided to base our system with the Pro/Engineer CAD system. Therefore, we also studied the list of all characteristics available in Pro/Engineer. Based on the published assembly characteristics and information available in Pro/Engineer models, we developed a preliminary list of assembly characteristics to support content-based searches. To ensure that these characteristics are not specific to Pro/Engineer, we also studied the assembly characteristics available in another CAD system – Unigraphics. We ensured that our list is compatible with the information available in Unigraphics. This research shows that data used in the search can be extracted from a CAD system. The application programming interface (API) of a CAD system can be used to extract the data in the signature of the assembly before execution. As characteristics used in this research are based on Pro/Engineer, API program for that CAD should be used to demonstrate the capability to extract the signature from the CAD system. This will enable the search system to work independent of any CAD system. The figure 1.5 summarizes the process of identifying assembly characteristics.

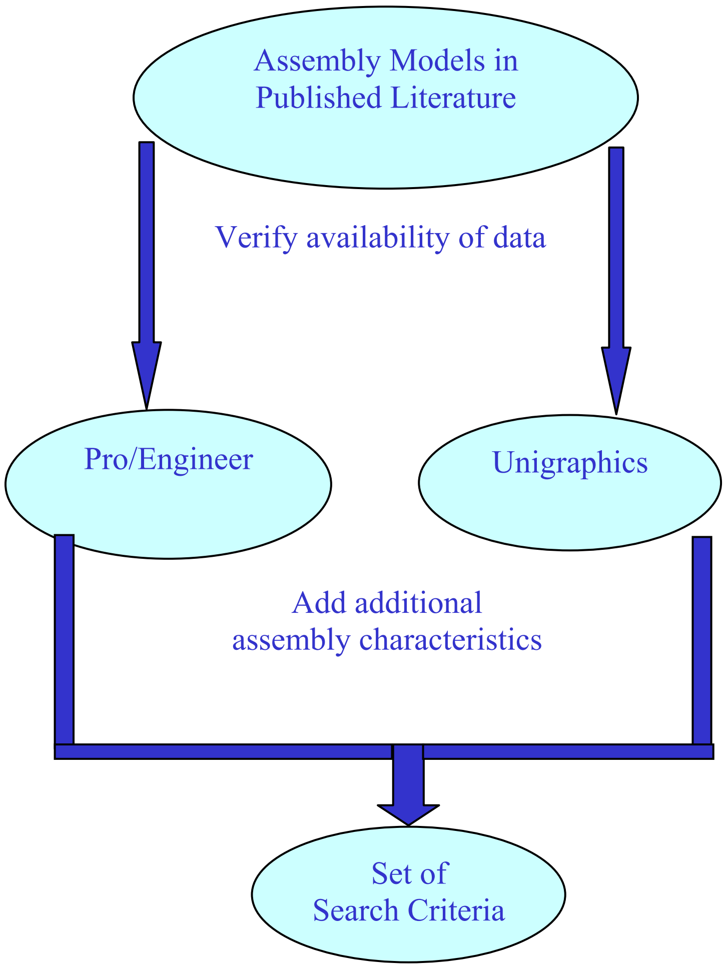

Assembly Models in Published Literature
Verify availability of data
Pro/Engineer
Unigraphics
Add additional assembly characteristics
Set of Search Criteria

Figure 1.5: Method to identify assembly characteristics

An assembly has large quantity of characteristics associated with it. It is important to have an intuitive distinction between different characteristics to develop a search system that is based on all of these characteristics and yet has an intuitive interface to define search. The identified characteristics were categorized into four main categories. The assembly design process was used as the basis of categorization of characteristics. The figure 1.6 shows the top level characteristics of an assembly.

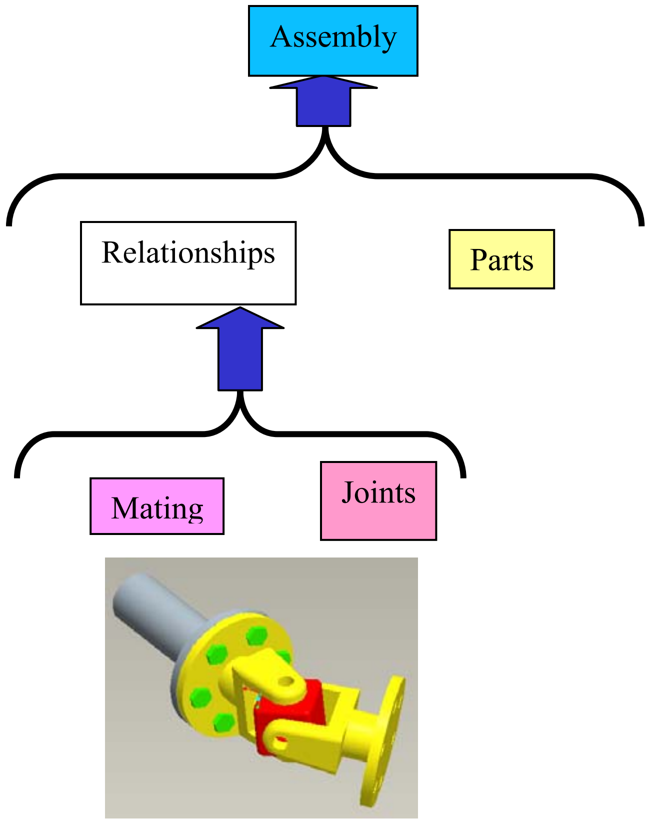


Figure 1.6: Top level assembly characteristics

Then, 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 basis to store the signature of an assembly. In [Gupt01], each part is described by a name, a pointer to the geometry, and a transformation, which places the part in its assembled position in the assembly. Every joint is described by a type, a name, and the names of the base and attached parts forming the joint. The joint and mating condition data stored in the signature can be read to construct a graph. The representation is used to support only the mating conditions and joints that are supported in Pro/Engineer. The signature fully represents an assembly and does not require the use of CAD files. Thus, these assemblies can be viewed and searched independent of any CAD software. As these signatures are not dependent on any single CAD software, they can be used to browse and search assemblies designed using different CAD software. This capability is important for large organizations that use different CAD softwares or for those who collaborate with external organizations in design.

### 1.3.3 Search methodology for different characteristics

Many of the characteristics of assembly are text or geometry based. For search based on text, string matching algorithms can be used. This problem has been studied extensively and various algorithms for exact and approximate matching strings are available. For search based on geometry, various approaches have been suggested for exact geometry match and either of them can be used to compare geometry. This research will undertake experiments to optimize the search time by ranking the efficiency of the each of the criteria. The search based on part mating condition will be based on graph compatibility. The user would define a graph which can be a part of a graph representing mating conditions in an assembly from the database. The two graphs need to be matched with each other and this would require a graph compatibility algorithm. Search based on joint relations uses double, as each relation between two different joints in the query as well as database is stored in a double. It is required to create data structure that can store and provide efficient access to all joint relations defined in either query or database. The search method involves exact string comparison to find the existence of joint relations in query and assemblies from database.

## 1.4 Outline of Thesis

The thesis has been arranged in the following way. The four major categories of assembly characteristics that are used for defining search are dealt with in different chapters as described below. The assembly statistics include assembly characteristics such as size and number of parts in the assembly. Chapter 3 deals with the search based on these statistics. Some parts in an assembly can have specific characteristics with regard to material, size and other characteristics. Use of an uncommon part in an assembly characterizes the assembly and can be used as search criteria. Chapter 4 deals with the searches that utilize characteristics of the constituent parts. The mating conditions in an assembly play an important role in its function. Therefore, we need to support searches based on the type of mating conditions that exist in assemblies. This search is discussed in Chapter 5. Joints in articulated assemblies define the possible motion between the parts. Various joints restrict different degrees of freedom and are used as identifiers of the assembly. Therefore, we need to support searches based on joints attached to rigid links. This search is discussed in Chapter 6. Chapter 7 discusses the implementation details of the research. Chapter 8 presents the conclusions of the research, the anticipated benefits out of the research and direction for future work in this area.

# Chapter 2

# RELATED WORK

This chapter provides a review of the state of the art in assembly modeling and representation, part geometry based search, function based assembly search and subgraph isomorphism. Assembly modeling and representation has been studied widely and is important to identify characteristics to search and represent these assemblies. Various methods have been suggested for geometry-based search and these approaches are important for a search tool for assemblies. Existing techniques for assembly search are mostly based on their function and behavior and their limitations with respect to form of assemblies and legacy data are important considerations to identify characteristics of assembly search. Various attempts have been made to represent the mating conditions between parts in a graph. A part of the graph can be used to search for the assembly it represents. This can be achieved using widely studied subgraph isomorphism. This chapter is arranged in the following manner: Section 2.1 discusses the approach for assembly modeling, Section 2.2 discusses the approach for part based search, Section 2.3 discusses the approach for function and behavior based search of assemblies and finally Section 2.4 presents an overview of subgraph isomorphism.

## 2.1 Assembly modeling

The common interpretation of an assembly is a collection of parts that have certain relationship among them. Most assemblies only utilize form based relationships.

Recently efforts are being made to emphasize function and behavior based relationships as well. In order to support computer aided assembly modeling, various attempts have been made to represent the assembly in computers. The models proposed by different research groups store different characteristics of assemblies for various end applications. In order to support content based search we need to assess these representations and decide which characteristics can be supported during search. These representations also provide a basis for creating assembly signature that stores all searchable characteristics of the assembly in a format that can be used in a computer based search system. This subsection has an overview of assembly representation proposed by different researchers.

CAD systems usually have two separate modules for creating part and assembly geometry. Geometry of individual parts is created in the parts and is referenced in the assembly modules. The location of parts is constrained by using low level mating conditions that are available in CAD system. User can select and specify joints from the list of joints available in a CAD system. The data about mating conditions and joints can be extracted from CAD systems. Current CAD systems do not store data about the function and behavior of the assembly.

Gupta et al. [Gupt01] propose an intelligent assembly modeling and simulation (IAMS) environment for assembly simulation and visualization. They provide detailed data structures to represent parts and assemblies along with tools, workspaces and plans. Their representation in form of nested lists has the capability to group all parts and store attributes to parts. Each part is described by a name, a pointer to the geometry, and a transformation, which places the part in its assembled position in the assembly. The representation can be extended to store textual attributes of a part. Every joint is described by a type, a name, and the names of the base and attached parts forming the joint. The mating condition data stored in the signature can be read to construct a graph. They provide an assembly editor that allows user to define attributes, assembly sequences and plans.

The assembly format defined by Gupta et al. is used as basis to store the signature of an assembly in the research presented in this thesis. The single data structure proposed by them stores data related to the form of an assembly and can be extended to include data from the additional criteria proposed in this thesis. Out of the many data files proposed by them, assembly files and part files are combined and used in this thesis to represent assemblies. Their tool however generates data like contact between parts from the part geometry or depends on the user to enter parameters for assembly sequencing by using an assembly editor. These capabilities of the model are not useful for a content based assembly search tool because they either generate data that is not known to the user or requires manual input to generate and thus cannot be used with assembly models generated by common CAD systems. Thus the research work in this thesis uses a part of assembly representation which can created using the data extracted from CAD models of assemblies.

Noort et al. [Noor02] propose an assembly representation that combines data for a product from its parts and assembly. This combined representation ensures that any change made either in the part representation or assembly representation is reflected in the other representation. It provides separate views for part feature and assembly features and updates the other representation when any change is made. The representation supports conceptual design, part detail design, assembly design, part manufacturing and assembly planning view.

As assemblies are a collection of parts, the features of assembly as well as individual parts is important in a content-based search system. This approach provides such a combined representation. This representation is generic in nature and can thus be used for multiple applications like assembly sequence planning and part manufacturing. As the views are related, the effects of change in either parts or assembly on the assembly are evident. This representation however does not use the mating condition specified in the CAD files but instead stores higher level relations that are inferred from part geometry. Although these are more intuitive to remember and use, current CAD systems do not support such relations. Thus, the high level relations cannot be conveniently used in assembly search.

Nanda et al. [Nand05] present an approach based on web ontology language (OWL) to systematically develop and deploy product families during all stages of design. This model captures the needs of the customer, function of the product and the components of the product. The entity list in the model can be used to map each need to a function and the component used to perform that function. This representation uses OWL data structures that provides the meaning of the data structure. The OWL representation is better than a XML representation due to the use of these data structures. The use of Description Logic (DL) enables a computer-based interpretation of the semantics without any human intervention.

The advantage from this approach is in sharing of parts in different designs. It will also be helpful in search of design information during all stages of assembly design. This model is helpful in capturing the function and behavior of the assembly. The OWL based representation used in this approach can be used to create search tools based on function and behavior of the assembly. This approach however does not consider the form of assembly. The research in this thesis presents algorithms to search for assemblies based on their form characteristics. Hence we have not used this representation. The signatures developed in this thesis can be easily stored in OWL format as well. Hence the work performed in this thesis is complementary to this approach.

Rachuri et al. [Rach06] present an object oriented design definition of an assembly called as Open Assembly Model (OAM) to represent electromechanical assemblies. This model represents the form, function and behavior of the system. Form of an assembly consists of geometry of the part and other constraints on the parts like mating conditions between parts and kinematic joints in the assembly. Form only considers the geometric rules to represent designs. Function denotes the transformations from input to output. Behavior of an assembly denotes its intentional and unintentional physical interactions. Function is thus said to be a subset of behavior. It consists of a conceptual model and as well as hierarchical model. The model consists of classes with artifact as the parent of all classes or the base classes. The information about the assembly is stored in classes. The model also stores the design rationale. The other information like size, geometric tolerance, material is attached to individual tolerance model class. They use the Unified Modeling Language (UML) to represent assemblies.

OAM can be used for collaborative design. This approach can be used to design a search system based on any of the characteristics of form, function or behavior. The model can thus be used at any stage of the design. This representation extends the ideas of *Core Product Model (CPM)* and *STEP* standard for representing an assembly. The research work in this thesis can be extended to use the CAD independent signature of the assemblies as proposed in this approach. The representation proposed by Rachuri et al. can reduce the time to access the assembly signature as text files from the CAD database. The representation can also be used to extend the capabilities of this research work to function and behavior of the assembly when tools to extract them from assemblies without any manual intervention are available.

Wang and Ozsoy [Wang90] propose the use of assembly graph. They represent the assemblies, subassemblies and the parts using the root, intermediate and terminal nodes. A set of mating conditions is recorded for every connection between each instance of a part or a subassembly. They use this proposed data structure to perform tolerance analysis for the assembly. The mating graph and the sum dimension of the assembly are used to find chain links in the tolerance chain and subsequently derive the fundamental equation.

The graph based representation proposed by Wang and Ozsoy has been used in this thesis. Wang and Ozsoy however support only three mating conditions viz. fit, against and parallel and use them to generate homogenous coordinates to position a part or subassembly. These three mating conditions are not sufficient to describe constraints on parts in assemblies even in case of non-articulated assemblies. The research in this thesis extends the graphs data structure to cover all possible mating conditions available in Pro/Engineer CAD system.

## 2.2 Part Search Techniques

Assemblies are a collection of parts and they can also be searched based on characteristics of the constituent parts. Thus, this research supports search based on the constituent parts of the assembly. It involves search based on the geometry of the constituent part and search based on its textual attributes. Search based on textual attributes is available in current PLM systems and involves string matching operations. A search system can be extended to provide textual search and techniques for text-based search have been identified in Chapter 4 of this thesis. For geometry-based search, different approaches can be followed based on the application of the user. Many research groups have proposed different techniques for part similarity assessment that can be integrated with an assembly search tool. Moreover, the techniques are based on the geometry of the part and can be used to identify search criteria for assemblies. This section discusses the search techniques based on the geometry of part.

### 2.2.1 Search Based on a Query Part

In many applications, a query part is available and the goal is to find all assemblies that have a similar part from a database of parts. The search strategy for such locating such parts is shown in figure 2.1. A common scenario is the search for jigs and fixtures assemblies that can be reused for similar parts with minor modifications. Various techniques have been developed to perform similarity assessment between 3D solid models. Similarity assessment between two 3D parts involves two main steps. The first step is to compute the shape signature of the object and the second step is to compare the shape signature by a suitable distance function. Major techniques used in the shape similarity assessment area can be classified on the basis of the type of shape signatures being used.

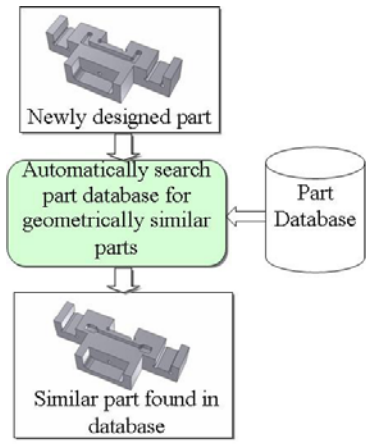

Newly designed part
Automatically search part database for geometrically similar parts
Part Database
Similar part found in database

Figure 2.1 Architecture for part search

The shape similarity assessment technique described in [Osad02] represents the object as a shape distribution sampled from a shape function measuring global geometric properties. This technique can be used as a first-cut filter to identify grossly dissimilar objects. In [Corn03], new filters for shape matching have been proposed. These filtering techniques have been applied to large databases of mechanical parts. In [Iyer04, Lou04], each 3D model is voxelized and represented by a vector whose components are moment invariants, geometric parameters, principal moments and eigenvalues of the skeletal graph.

Graph-based techniques convert solid models into attributed graphs that represent relationships among various geometric and topological entities in the solid model. Among the types of graphs that have been used there are Model Signature Graphs [ElMe03, McWh01b] and Multiresolutional Reeb Graphs [Besp03]. These techniques are simple to implement, but may not have high discrimination power.

Spatial function based techniques use shape signatures that are spatial functions. Spatial functions have been introduced in [Ko03, Ko05]. A technique based on spherical harmonic descriptors is described in [Funk03]. The Fourier transformation based technique described in [Chak05a, Chak05b] performs similarity analysis based on the boundary representation.

Feature-based techniques represent the 3D object referring to their features. Many different types of approaches have been developed [Cici00, Card04, Card05, Karn05a, Rame01]. Feature based techniques appear to be promising for domains such as machined parts. Additional details on shape similarity assessment techniques can be found in [Card03].

Existing techniques provides excellent performance for filtering irrelevant parts. However, when locations, type, and orientations of faces play a major role in determining similarity, existing techniques do not seem to have sufficient discrimination capabilities. This information is important in case of similarity assessment of parts for manufacturing. A shape similarity assessment technique that uses this information has been described in [Card06].

Any of the techniques cited above can be used for searching parts in the assembly search tool proposed in this thesis. The current implementation includes the capability to search parts as described in [Card05, Card06]. The desired part is located using these techniques and is used in the assembly search tool.

### 2.2.2 Search Based on a Sketch

3D models are not always available to act as queries in query based search. Creating a complex 3D model of part to act as query is time consuming activity. It is easier for a designer to represent the relevant characteristics of the required part in form of 2D sketches compared to preparing a 3D model. Thus, in many design reuse applications, users may not want to create detailed CAD model to begin search. In such cases, sketch based tools for representing parts can be used to search for the required part or for locating a query part. These sketches usually are standard 2D views, i.e. top view, front view and side view of the required part.

In [Pu05] a technique capable of retrieving 3D parts from a database based on user free-from sketches is presented. The 3D part retrieval can be enhanced by user further feedback. The part can be searched based on up to three views with user assigned weights for each view. The technique consists of three main steps: (1) determination of 3D part relevant orientations, (2) generation of 2D sketches, and (3) computation of similarity degree between sketches and 3D part projections. In [Min02] another 2D sketch-based technique for 3D shape retrieval is presented. This similarity measure is invariant with respect to rotation. In this case, 2D views are compared based on the Fourier coefficients of functions obtained by intersecting the 2D Euclidian distance transform of the image with a set of concentric circles.

The assembly search technique can be extended to integrate sketch based part search tools to search for constituent parts. These tools can also locate a similar part from the database to be used as a query part. One of the major applications of this technique is the sketch based search technique for assemblies. The mating condition between parts is a sketch-based representation of assemblies. The assembly search system uses a graph based representation of the mating condition between parts and allows users to define a part of such graphs to search for assemblies. This search technique is based on part search technique where a user can define an outline of part to search the part database.

### 2.2.3 Search by Visual Browsing of Part Database

In many situations the user does not have a query part. Moreover, due to the part complexity and/or user's limited familiarity with engineering graphics concepts, it is not possible to use sketch based query methods. In such situations, visual browsing of CAD databases is a possible solution [Karn05b].

Suppose a designer would like to locate a part in the design database consisting of thousands of parts for reusing design information. There are two possible cases. In the first case, the designer has a query part and wants to locate a part similar to the query. In such cases, geometry-based search techniques are useful for locating similar parts. However, in the second case the designer may not remember the exact geometric details of the part to locate it through the geometric search techniques. In such cases, the user will need to locate the desired object by browsing through the CAD databases. Once the designer locates a part similar to the desired part, he/she can use that part as the query in geometry-based technique. Thus, an integrated system that can assist the designer in locating similar parts by providing geometric as well as visual search is useful.

A similar browsing capability has been provided in the assembly search system to browse the assemblies in the CAD database. A user can specify all wildcards in the search criteria that will allow the user to browse the entire database of assemblies. The user can also specify some search criteria to prune away a part of the assemblies' database and browse the remaining assemblies. A separate assembly viewer tool has been provided to view individual assemblies in details.

## 2.3 Function Based Search

The research work in this thesis describes a search system based on the form of an assembly. The function and behavior are equally important characteristics of an assembly. They can be described by textual attributes and this research work can be extended to include such search when tools to extract the data automatically are available. Currently most of the representation of function and behavior is based on information gathered during the design process. As can be inferred from the literature review in Section 2.1, modern CAD systems as yet do not store such design information and depend on external tools like PLM to store this information. This section considers some approaches for searching based on the function and behavior of assemblies.

Shaffer et al. [Shaf05] present a web service based approach which can integrate a number of different approaches to search for assemblies. Additional services are added by providing a web service description language (WSDL) document that describes the new repository services. The meta-data obtained from a WSDL document about the service can be used to generate interface in the target language.

An advantage of such a system is that it integrates different existing approaches to search for assemblies and provides a framework that can be used to integrate any new approach. Thus, the designer does not have to spend time to understand the interface to study different search systems. Moreover, the search systems designed with different end applications like process planning or cost estimation can be used by the designer. This approach does not use relational database models and thus prior indexing of information about models in the relational is not required. The architecture proposed by Shaffer et al. can be used to provide the search tool proposed in this thesis as a web based service. This will make this search tool independent of any CAD software and extendable to include function and behavior based assembly search.

Kopena et al. [Kope05] present an approach based on semantic web to represent conceptual design which can be used to represent and access assemblies. They create descriptions for products using the conceptual design interface that can be used to annotate designs and search based on these annotations. These descriptions can also be used to classify designs. Search based on these class descriptions is one of the applications of this system. The semantic web also allows markups and publishing of design besides allowing large data to be collected and organized in an efficient method as compared to the techniques currently used. The semantic web based assembly representation is defined using description logic. Thus, automated reasoning can be done to identify function of an assembly.

They have proposed an approach to capture the form and function of a design during the conceptual design phase. Once the form and function are captured, search can be defined on the design repository. In their system, a three dimensional sketch is created and is annotated with function and flows. A reasoner, which is included in the system, converts this signature into class description. A search is performed on the database and all designs which are members of the class description so generated are returned to the designer as result of the query. The semantic web based approach can improve the efficiency of the search software. The capability to classify designs can be used to form clusters of similar assemblies in the database which can contribute to improve efficiency of the search software. The approach proposed by Kopena et al. is useful to capture the function of an assembly that is being designed in the organization. However, this approach cannot be used to search for legacy CAD data as the information about the function of assemblies in legacy format is not available. As of now, a tool that can extract function of an assembly from legacy data without any human interaction does not exist. As the legacy data forms very large part of any organization's database, this approach is useful only for new designs.

Karnik et al. [Karn05c] provide a design navigator system that stores the information and allows search based on the functionality of the assembly, changes made during the design process, geometry of the parts and finally by visual navigation of the assembly. As the information is captured in computer interpretable form, it can be readily searched using tools available on a computer. The system also provides a connection between requirements and specifications to the individual parts in the assembly. The system includes a functional modeler, rationale modeler and a design history modeler. Search tools based on different aspects of the assembly have been integrated in a single framework.

This system is good to search for CAD assemblies of products in the organization based on function and design history. However, it can only use the data created through the proposed design modeler. Design history and rationale behinds changes of legacy CAD data in the database of an organization are not available and cannot be inferred from the CAD models. A reliable tool to infer the function of a product from its CAD assembly is not available. Thus, this tool cannot be used for content based search. The approach can however be integrated with a content based assembly search tool to search for assemblies when the required data is available for all assemblies in the database.

Bohm et al. have used the archived function-based design knowledge to generate concepts of design. They use the Chi Matrix and the Morphological Matrix technique to generate the design ideas. They state that 76% specified subfunctions return results and an average of 61.35% components can be derived by using Morphological Search feature. They use Functional Basis language to describe functions. It consists of a noun that describes a flow and verb that describes function. The search returns a number of artifacts that perform similar function which are then used in the generation of concepts.

The Morphological Matrix used in this method is generated manually. A method to extract the Matrix from legacy CAD data must be extracted automatically. This approach can combine different subfunctions and return them as the result of a single input query. The method also considers functions from other domains like electromechanical assemblies. The user however needs to select the input domain for the search. The authors state that with an increase in the size of the database, the number of generated concepts would increase which is the primary objective of this search.

## 2.4 Overview of subgraph isomorphism

Two graphs are isomorphic if there exists one to one mapping between nodes and edges of the two graphs. Two graphs are said to be subgraph isomorphic if a one to one mapping exists between some nodes and edges of one graph and all nodes and edges of another graph. The subgraph isomorphism problem is NP complete. This research involves solving a graph compatibility problem, which is similar to subgraph isomorphism. This section discusses various approaches to solve the subgraph isomorphism problem.

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. Most early methods have used backtracking search method to solve this problem. This method processes the two graphs simultaneously. This method uses an adjacency matrix with 1 and 0 as its elements. It uses a refinement procedure to infer and reduce the visits to successive nodes during the backtracking search. It is used to find all isomorphisms between the two graphs. This method is tested only for connected graphs.

This algorithm is still very popular and is used for query graph with limited number of nodes and edges. The backtracking approach suggested by Ullmann is the basis of many more efficient approaches with refined pruning and search space reduction techniques. However, this algorithm does not prove to be efficient in case of query graph with large number of nodes. This algorithm does not take into account any previous knowledge of correspondence between nodes of query graph and a database graph. Our algorithm is modeled after this approach. The research in this thesis does not search for all isomorphisms but only the first instance and thus can be performed in lesser time as compared to Ullmann's approach.

Yu and Wang [Yu04] have used a 2D continuous Hopfield Neural Network model to obtain a subgraph of a graph that is isomorphic to query graph. This algorithm is used for undirected and connected graphs. They construct a neural network with dimensions equal to the number of nodes in the two graphs. An energy function is defined and parameters of the network are deduced from the energy function. A random or biased neuron initialization is used in this method. They use fourth order Runge-Kutta method to solve the equation.

They define essential conditions for subgraph isomorphism that can be used as pruning conditions in any method to solve subgraph isomorphism. The pruning conditions used in this approach are generic in nature and can be used for any approach for subgraph isomorphism. This thesis has adopted the pruning techniques suggested by Yu et al. However, the Ullmann's approach has been found more adaptable for bounding conditions in this thesis. Thus, the algorithm for subgraph isomorphism proposed in this research is not based the approach based on Neural Network suggested by Yu et al.

Messmer et al. [Mess98] use a decision tree which is created in a preprocessing step. Subgraph isomorphism is detected at run time using these decision trees. They recommend several pruning techniques to reduce the size of the decision tree. The decision tree is constructed by transforming adjacency matrix of the model graphs i.e. the graphs in the database.

They claim that the algorithm has a quadratic worst-case asymptotic time complexity with respect to the number of nodes in the query graph and is independent of the number of model graphs and the number of edges in any graph. However, preprocessing step to create decision tree is not included in this time complexity. A major drawback of this approach is that the decision tree can be exponential. The worst case complexity of this algorithm is $O(Lm^n n^2)$ where $L$ is the number of graphs in database, $m$ is the number nodes in the query graph and $n$ is the number of nodes from the model graph. They have suggested pruning techniques based on prior matching of nodes, eliminating a set of nodes from database graph with peculiar permutations, and considering a subgraph of fixed size while testing for subgraph isomorphism. However, this approach has not been used in the formulation of subgraph isomorphism in this thesis because a technique to reduce the complexity of decision tree has not been suggested.

Fuchs et al. [Fuch00] have proposed an error tolerant algorithm that uses prior knowledge of correspondence between nodes of a query graph and a graph from the database of graphs. They recursively decompose the graphs from database into subgraphs and propagate the external information during the decomposition of the graph. This algorithm has been used in 3D reconstruction of buildings from images. They have also suggested editing of the graph data structure to include external information.

The worst case complexity of the algorithm with the prior knowledge of matches between some nodes is $O(Lm^q q^3)$ where $L$ is the number of graphs in database, $m$ is the number nodes in the query graph and $q$ is the number of nodes from the query graph for which corresponding nodes are not known. They claim that the step for matching two nodes is more efficient because the graph data structure integrates the external information. This algorithm is useful for graph compatibility because of its error tolerance. A prior knowledge of correspondence between nodes from query graph and database graph can be used to significantly reduce the search space in DFS. The research in this thesis however does not store any knowledge about correspondence and this can be achieved after integrating this search system with a database. The approach is thus useful for further research on assembly search system.

Cordella et al. [Cord04] have suggested an algorithm for large graphs. They claim to have achieved 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. This algorithm can consider attributed relational graph and use the information in the semantic part to provide reduced matching time.

This algorithm has used a state space representation (SSR) of the matching process. It also includes five rules for feasibility and involves syntactic and semantic comparison of nodes. Out of the five rules, two are used to check the feasibility of the solution and three are used to prune the search tree. This algorithm does not assume any constraints on the topology of the graph and thus has generic applicability. The algorithm uses vectors that provide constant time access to its members thus reduces memory requirements making the algorithm usable for graphs with thousands of nodes and edges. The algorithm explores the search graph using a depth first strategy. The query graph in this research is not expected to have more than fifteen nodes, and thus this algorithm has not been adapted for graph compatibility. Moreover, since the query and database graph used in this research have attributes for nodes and edges that can be used in pruning search tree, Ullmann's basic approach has been used to formulate the algorithm.

# Chapter 3

# SEARCH BASED ON ASSEMBLY STATISTICS

This chapter is arranged in the following manner: Section 3.1 describes the different characteristics that can be used to define search in this criteria, Section 3.2 discusses the methods used for searching based on the characteristics and Section 3.3 illustrates the use of this criteria with an example.

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 as shown in the figure 3.1. 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 adopt 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 to search based on the size of existing prosthetic assemblies. This scenario illustrates the benefits of being able to search based on overall assembly statistics.

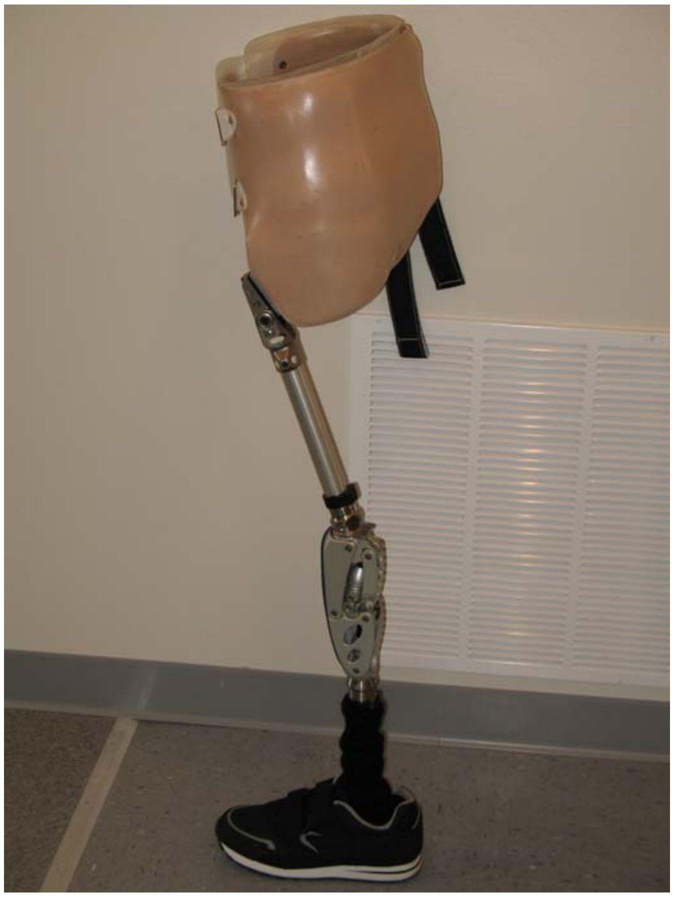


Figure 3.1: Example of a prosthetic device – artificial leg [Pros06]

## 3.1 Search Definition

In our framework assembly search can be performed based on the following criteria related to shape statistics.

- **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, the width, and the height of the bounding box. The data required for performing searches based on these two sizes are obtained from the Pro/Engineer assembly model and Open Scene Graph library. The figure 3.2 and 3.3 shows the bounding box and bounding sphere of an assembly.

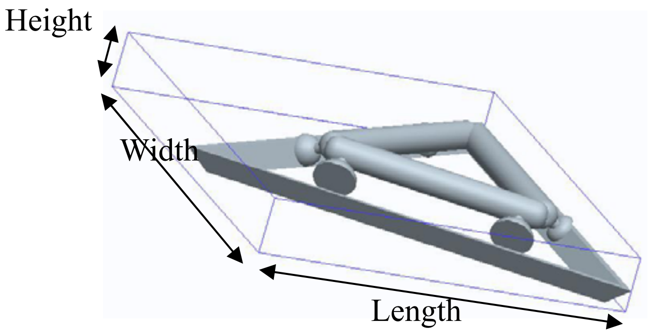


Figure 3.2: Bounding box

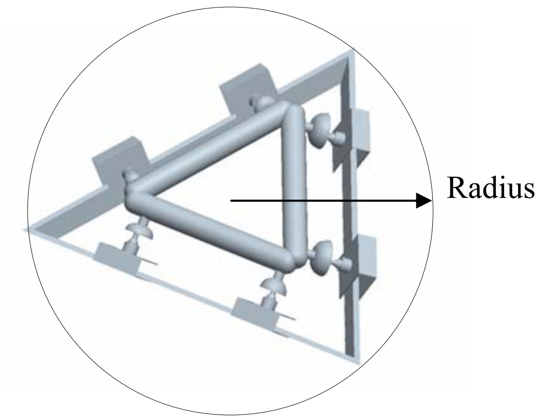


Figure 3.3: Bounding sphere

- **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. To perform this search the number of parts is extracted from the Pro/Engineer assembly model. In addition, we also determine if a part being used in the assembly is a standard fastener. (e.g., screw, bolt, nut, and washer).

- **Number and Types of Articulated Joints**: The user can also search assemblies based on the number and types of joints in the assembly. The types of joints that can be defined in Pro/Engineer are *pin, U-joint, gimbal, cylindrical, slider, planar, ball, weld and bearing*. Besides these joints, Pro/Engineer allows the user to define the connection as *cam-follower, slot-follower,* and *gear pairs.* This type of search is defined by indicating the number of joints in each selected joint type. If a joint type is not selected by the user, then the system excludes that joint type from the search. The type and number of each joint are extracted from the Pro/Engineer file. Even though the system currently uses Pro/Engineer joint types, it can be easily extended to work with joint types found in other CAD systems.
- **Number of Usages in Other Assemblies**: An assembly such as a motor may be a popular assembly and hence used in many other assemblies. So some users might remember the large number of usage associated with an assembly. Hence, users can specify the number of usages of an assembly in other assemblies as a possible definition of search. This might be an effective way of searching a frequently used assembly.
- **Overall Shape Characteristics**: Assemblies may have overall shape characteristics that a user might remember. For example an assembly may predominately consist of rotationally symmetric parts or sheet metal parts. Such characteristics can often be used as a possible way to search for an assembly. Currently, we support the following two ways 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. The figure 3.4 and 3.5 show a sheet metal part and prismatic part.

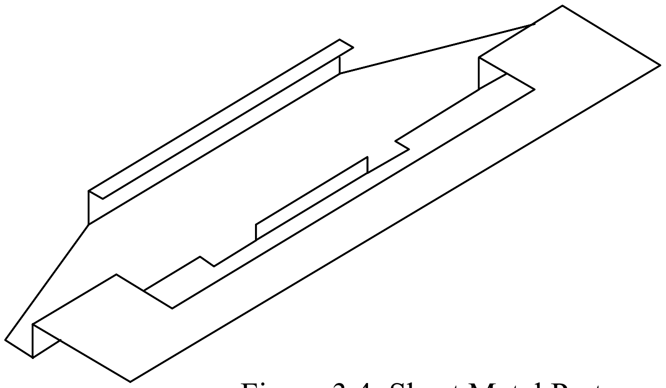


Figure 3.4: Sheet Metal Part

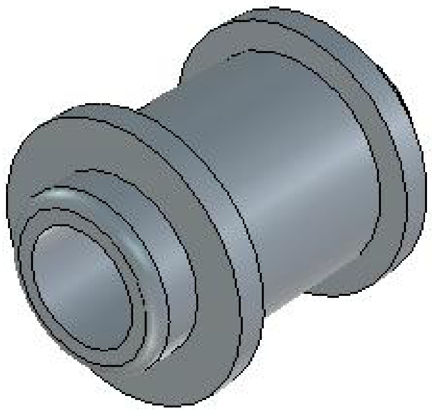


Figure 3.5: Rotationally Symmetric Part

- **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 on an assembly drawing in Pro/Engineer. 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. Pro/Engineer assembly drawing notes are used to extract names of possible standards to which an assembly conforms.

- **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. The name of designer of each part and the assembly designer can be readily extracted from PLM system.

Many of the above criteria require the users to specify a positive real number (may or may not be an integer) 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. For example, a user can indicate that the number of part needs to be between 30 and 50. We also allow the user to leave either the upper limit or the lower limit as unspecified. For example, if the user specifies the lower limit as 30 and leaves the upper limit as unspecified, then the search attribute has to be greater than or equal to 30. If both the upper limit and lower limit have the same value, then the attribute in database assembly has to exactly match the specified value. 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 against this target value and ranked based on their closeness to the target value.

A user can also select the multiple different criteria from the above list to define a search. For example, a user can define a search in the following manner: $30 \leq$ number of parts $\leq 40$ AND $5 \leq$ number of slider joints. We currently only support conjunctive (AND) operators to combine search based on multiple different criteria.

## 3.2 Search Method

All search attributes are either defined using numbers or strings. For search attributes that are defined using numbers with a range option all attribute instances in the database that meet the search definition are considered as feasible matches. We do not rank the results if this type of search definition is used.

For search attributes that are defined using numbers with a target option all relevant attribute entries in the database are compared to this target attribute value and the penalty function $|t-a|/n$ is used to rank order the matches, where $t$ is the target value, $a$ is the value of attribute in the database assembly, and $n$ is the normalization value. The value of $n$ needs to be selected carefully to suit an organization's needs. The value of $n$ will determine how many assemblies are considered as matches for a given target value. The value can be selected based on the size of the database of assemblies in the organization and the expected deviation of the required assemblies from the chosen target value. In addition, one can also set a cut off value: if the attribute value in the database assembly is farther than the cut-off value then that assembly is excluded from the results reported to the user.

For search attributes that are defined using strings, exact string matching is performed.

## 3.3 Example

In the first test, a criterion to find all assemblies that had bounding sphere size between 3 and 20 inches was specified. This criterion gave a list of 8 assemblies from the database. The figure 3.6 shows the results in the assembly search system's result window.

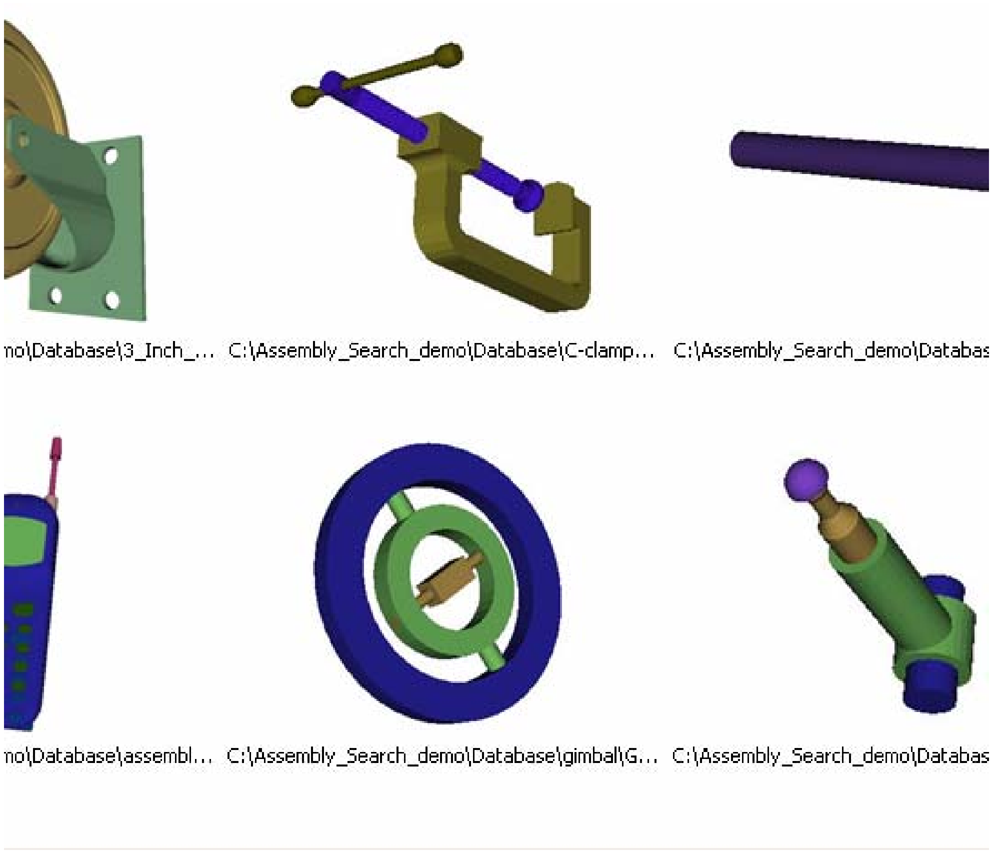

no\Database\3_Inch_...
C:\Assembly_Search_demo\Database\C-clamp...
C:\Assembly_Search_demo\Databas
no\Database\assembl...
C:\Assembly_Search_demo\Database\gimbal\G...
C:\Assembly_Search_demo\Databas

Figure 3.6: Eight assemblies obtained by assembly statistics based search

Then the search criterion was made stricter by specifying the same range for size and another criterion was added for the range of the number of components to be between 15 and 20. A search over the entire database gave a list of 2 assemblies. Thus, more specific search results were found by increasing the strictness of the criteria, i.e., by imposing additional constraints on the criteria. The figure 3.7 shows these 2 assemblies.

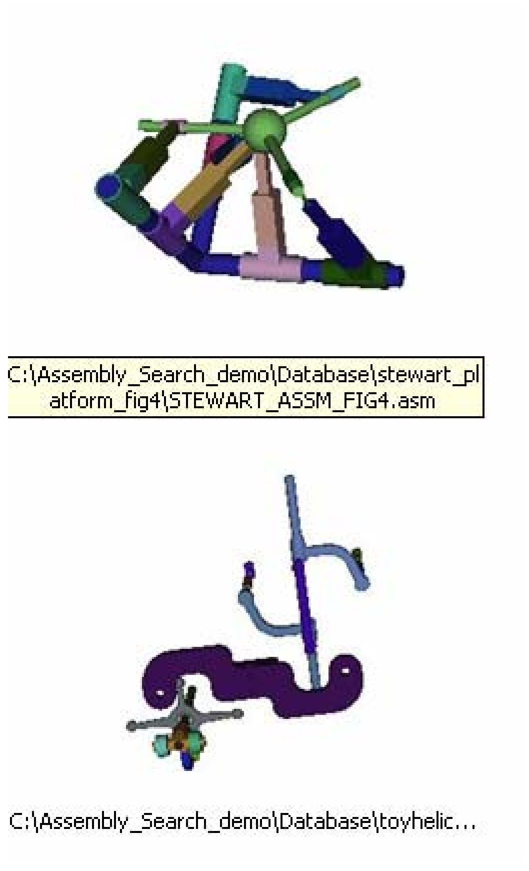

C:\Assembly_Search_demo\Database\stewart_pl
atform_fig4\STEWART_ASSM_FIG4.asm
C:\Assembly_Search_demo\Database\toyhelic...

Figure 3.7: Two assemblies obtained by assembly statistics based search

# CHAPTER 4

# SEARCH BASED ON CONSTITUENT PARTS

This chapter is arranged in the following manner: Section 4.1 describes the different characteristics of a part that can be used to define search, Section 2 of Chapter 3 discusses the methods used for searching based on the characteristics and Section 3 of Chapter 3 illustrates the use of this criteria with an example.

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 search for an assembly by specifying the size and material for a part of the assembly. These criteria will allow the designer to search for an assembly containing a part with specified size and material. The DFMA rules embedded in the assembly can be reused for the design of a new assembly.

## 4.1 Search Definition

The system supports search based on the geometry of the part and the characteristics of the part. A combination of the two criteria is also supported. The two criteria for search 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. A single assembly can contain a set of many of these standard parts. This criterion is useful when designer knows that a certain set of standard parts were 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. The figure 4.1 shows a standard part used in an organization. The part is a gear.

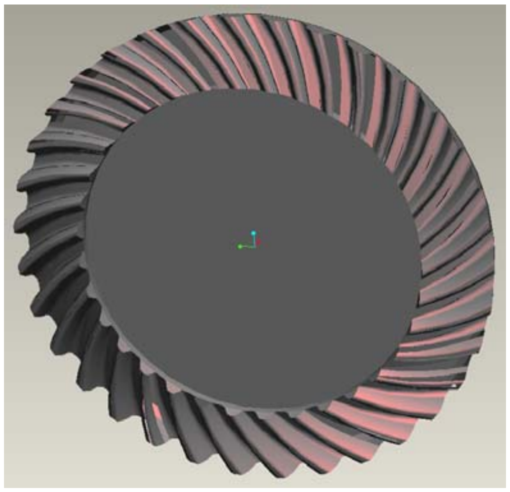


Figure 4.1: Example of a standard part – gear

- **Custom part**: This is useful in a scenario when the designer knows that a part used in the assembly approximately matches with a part in the database. The user can select a .stl representation of any Pro/Engineer part from the database as input geometry. The system allows search based on approximate geometry matching. The figure 4.2 shows a custom part used in an organization.

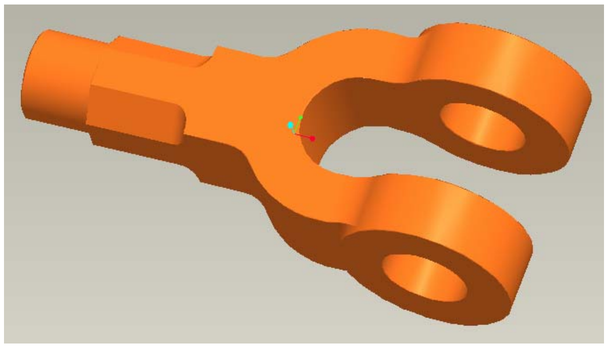


Figure 4.2: Example of a custom part

- **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.
  - **Part attributes**: Attributes are the textual data stored in the CAD files. Organizations have a set of standard attributes that help classify the assemblies in the database. For example, an attribute called *part source* can have values 'bought out part' or 'in house part'. Attributes can have values in the form of a numbers or strings. The system supports search for both the types and values of attribute. The user can define the title and value for attributes. If the value of an attribute is a string, the designer can define either an exact or an approximate search. If the value of the attribute is a real number, the designer can define search based on a target value or a range.
  - **Name of the owner**: CAD files store the name of the creator and the modifier in part history. In most CAD systems, this refers to the login names of users on the

Operating System (OS). This data is useful since the 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.

We support only single criteria based search and conjunctive search, similar to those described in Section 3.2. The user can define any combination of the above criteria to define a single search. The range and target value definitions for numbers and exact and approximate match options for strings are also available.

## 4.2 Search Method

Searches based on attributes defined using numbers are handled exactly the same way as described in Section 3.2.

The problem of finding an approximate match is usually referred as "search for similar parts." This problem has been explored in many different design and manufacturing contexts [Card03, Li04]. There are broadly two different kinds of methods. The first method uses the overall object shape in identifying similar parts. Representative techniques in this area include [Hila01, Karn05b, McWh01a, Osad01, Sung02]. The second method uses shape features in identifying similar parts. Representative techniques in this area include [Card04, Cici01, Rame01]. Both these approaches have their own relative merits and demerits. Depending upon a particular application, one might prove to be better than the other. The figure 4.3 shows the use of part search tools in the assembly search tool.

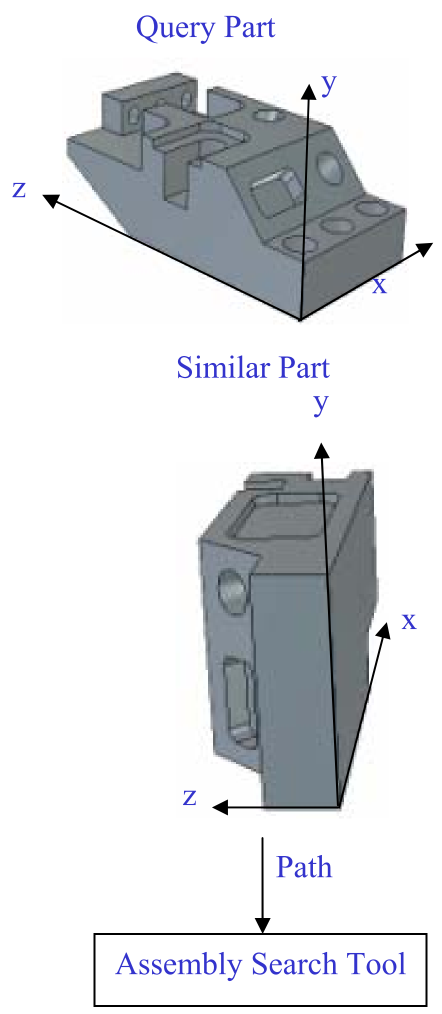


Figure 4.3: Use of geometry based part search tool in assembly search

For search attributes that are defined using strings we use two methods for identifying matches in the database. The first method is based on the exact string matching. In this case all database entries that contain the search string are considered as matches. The second method uses approximate string matching algorithm by Levenshtein [Leve66] and uses the closeness of the strings to rank order the matches. The implementation from [Merr06] is used in the code. This criterion does not allow the user to specify any approximate strings for matching. Approximate string matching is used in constituent part based search. A distance of 10 between the query string and the string in the database is considered a match.

## 4.3 Example

Consider a scenario where the designer is searching for an assembly that uses a variant of Beryllium liner. The liner is a custom part in the organization and its geometry is not available to the designer. The designer knows who designed the assembly, the size of the assembly and the number of parts used in assembly. The following search criterion was specified for the search:

- A custom part made of Beryllium and owned by “Chris Harris”
- The assembly bounding box length between 125 and 165 inches, bounding box width between 160 and 190 inches and bounding box height between 300 and 330 inches.
- A target of 7 parts in the assembly.

The search criteria collectively were found to be sufficient the desired rocket motor assembly from the database. This assembly satisfies the designer’s exact requirement and is shown in figure 4.4.

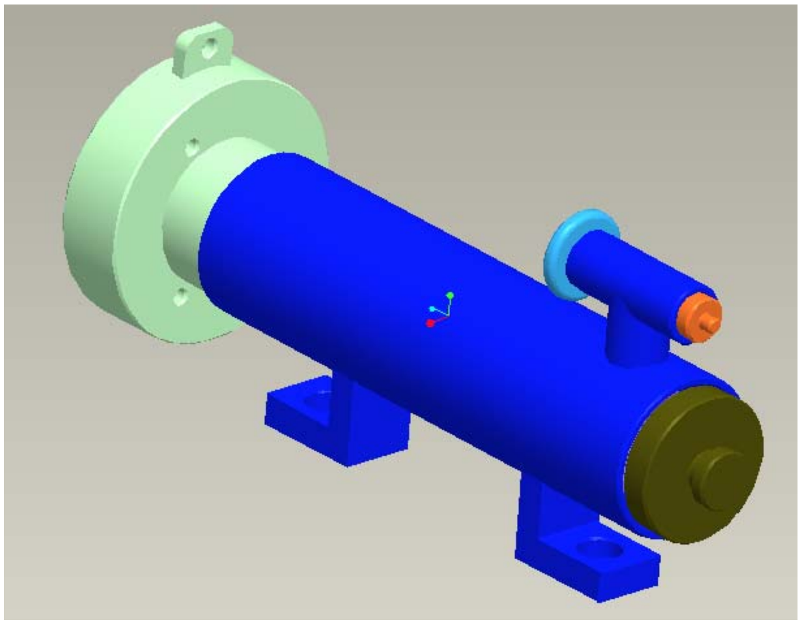


Figure 4.4: Result of the search – Rocket motor assembly

# Chapter 5

# SEARCH BASED ON PART MATING

This chapter is arranged in the following manner: Section 5.1 describes the method to define a query graph to search based on this criteria, Section 5.2 discusses the graph compatibility methods used for searching based on the characteristics and Section 5.3 illustrates the use of this criteria with an example, Section 5.4 describes the initial algorithms for initial processing of graphs in details, Section 5.5 lists the pruning algorithms used before depth first search, Section 5.6 describes the main depth first search algorithm, Section 5.7 describes the auxiliary algorithms for depth first search and Section 5.8 describes the experiments conducted to test the performance of the algorithms.

The mating conditions are the restraints (constraints) imposed on the location of a part with respect to other parts in the assembly. Different set of restraints imposed on the same set of parts can constitute different assemblies. This search criterion uses mating conditions to search for an assembly. The designer specifies the mating condition between parts of a subassembly or an assembly by building a query mating graph. This query mating graph is compared with mating graphs corresponding to the assemblies in the database. The results of the search are all assemblies whose mating graphs are compatible with the query mating graph.

## 5.1 Search Definition

The designer defines an input mating graph to represent a subassembly/assembly. Individual parts are represented as nodes or vertices and edges connect two nodes when mating conditions exist between the two corresponding parts.

Each node 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 Pro/Engineer geometry of the part. The geometry for standard parts is referenced from the library of standard parts in the organization. This attribute can also be left unspecified.
- **Type**: This criterion is defined only for standard parts and specifies the subcategory of the standard part. The available subcategories are: *bolts, nuts, washer, bearings, resting pads in fixtures, mold base, ejector pins, springs, circlips, rivets, retaining rings, hydraulic and pneumatic cylinders, chains, belts, gears, brakes, couplings, engine, actuators, pumps, valves, oil seals, vacuum seals, collars joints, universal joints, solenoids, switches, heating elements* and *limit switches*. 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 the node. This attribute can be given a specific value. If no value is specified, it is

taken to be zero. The graphs need not be necessarily planar, i.e. the degree of any node can be greater than 5. In fact, the degree of a node may be equal to one less than the number of nodes in the query graph. This scenario 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 graph has the following attributes:

- **Type**: This represents the type of mating condition represented by an edge. The search tool supports all mating condition options available in Pro/Engineer. The options available to the designer are: *mate, align, insert, tangent, point on line, point on surface*, and *edge on surface*. 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.

Please note that 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 graph is not a unique graph and many different database graphs might be compatible with the query graph.

An illustration of compatible and incompatible mating graphs is shown below. Figure 5.1 shows an example of a mating graph. This graph will be used as a query graph. The graph has four nodes. Node 1 includes a custom-built part and includes a reference to a file describing part geometry.  Nodes 2 and 3 include standard bolts and hence explicit reference to part geometry files are not needed. Node 4 again includes a custom built part. However geometry is not specified for this part and hence during the search process, parts with different geometries will be able to match this node. In the next step, edges are created between nodes. They represent the mating conditions between parts. Both custom parts are connected to the bolts through mating conditions that mate two faces on the parts. An edge is defined between Nodes 1 and 4 but an exact mating condition is not specified for this edge.  Hence during the search process, this edge will be able to match with many different mating conditions.

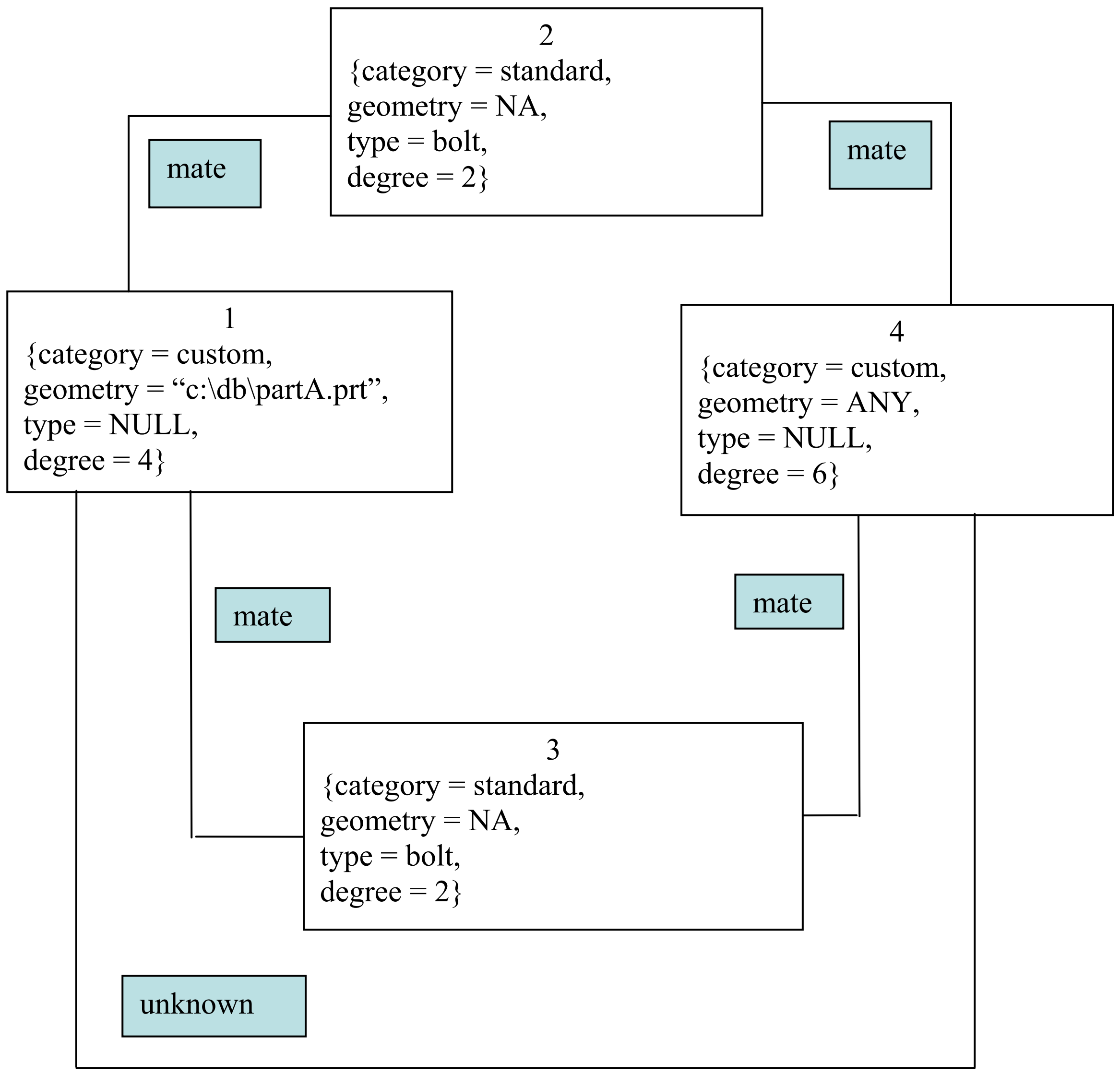
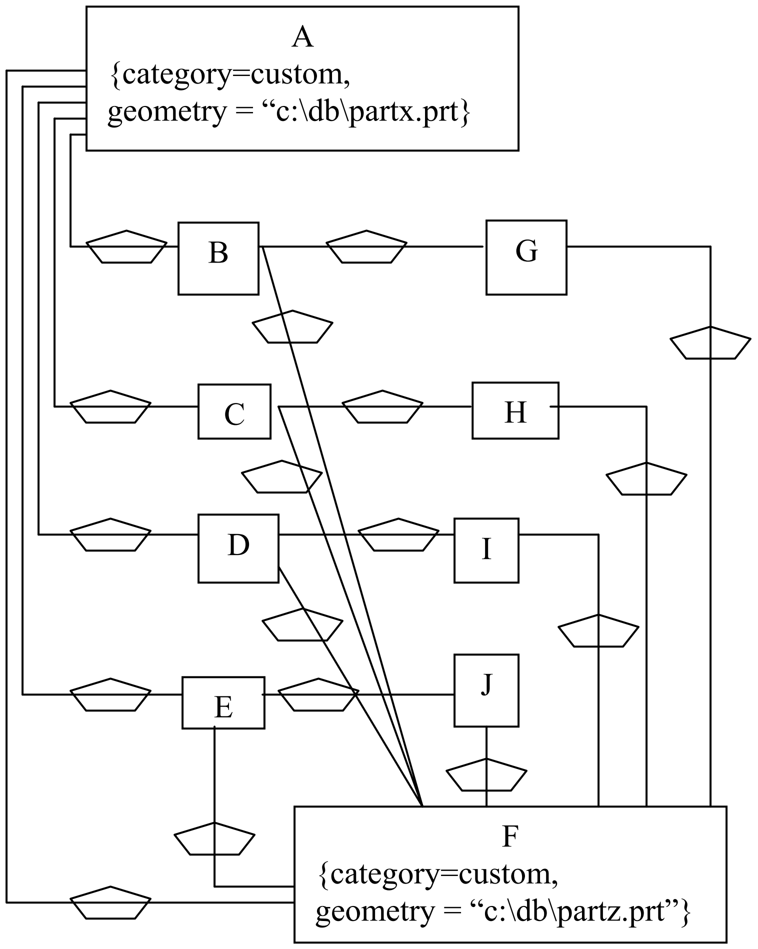


Figure 5.1: The query graph for part mating conditions

Figure 5.2 shows mating graph for an assembly from the database. The graph defined in figure 5.1 is compatible with the graph shown in this figure 5.2. Nodes 1 and 4 in figure 5.1 above match with Nodes A and F in figure 5.2. Nodes 2 and 3 in figure 5.1 match with Nodes B and D in figure 5.2. The mating conditions between standard part and custom parts are also same in the two graphs shown in Figures.



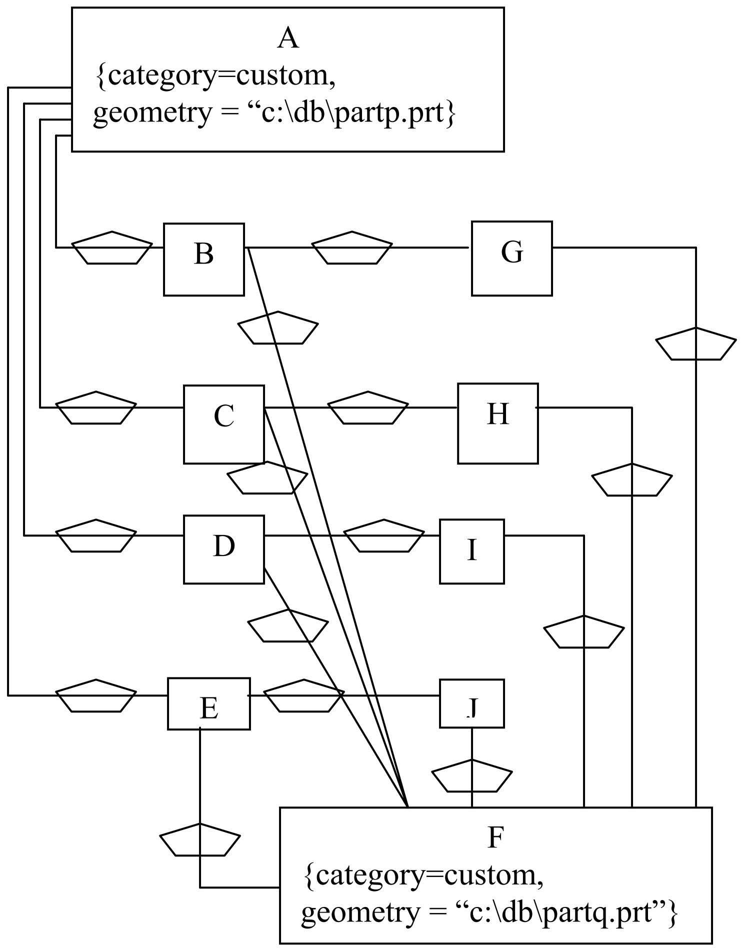
⬠ Represents “mate” mating

B, C, D and E are {category = standard, type = bolt}

G, H, I and J are {category = standard, type = nut}

Figure 5.2: A database mating graph with which the query graph is compatible

Consider a graph shown in figure 5.3 that represents another assembly in the database. In this graph, the node labeled A does not have an edge connecting it to a node representing the custom part. However the query graph shown in figure 5.1 has an edge between the two nodes representing custom parts. Thus, the query graph defined in figure 5.1 is incompatible with the graph shown in figure 5.3.



⬠ Represents “mate” mating

B, C, D and E are {category = standard, type = bolt}

G, H, I and J are {category = standard, type = nut}

Figure 5.3: Another database mating graph with which the query graph is incompatible

The figure 5.4 shows another example of compatible and incompatible graph.

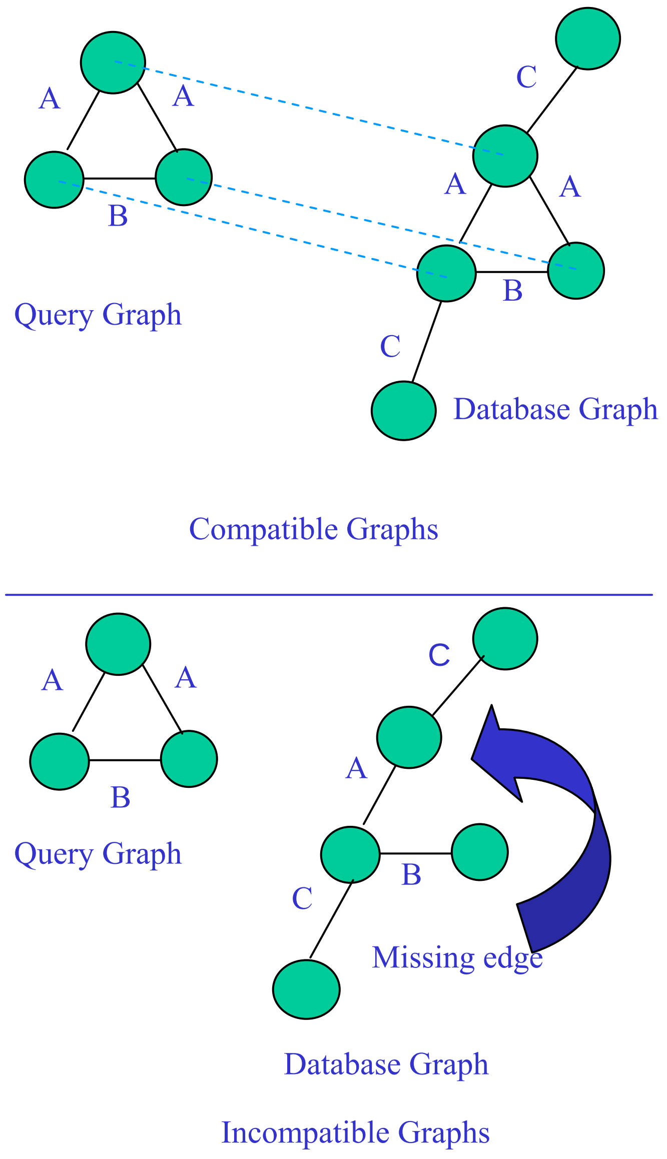


Figure 5.4: Graphical explanation of compatible and incompatible graph

## 5.2 Search Method

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 the 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 graphs. However, since multiple mating conditions are possible between two parts, two nodes may be connected by multiple edges in the database mating graph. Hence, strictly speaking, such a graph should be termed as a multigraph.

The query mating graph needs to be compared with the mating graph of every assembly in the database. However, as already mentioned before, the query graph may not be a fully specified graph due to presence of wildcards in the query graph. Thus, the available subgraph isomorphism techniques cannot be used in their present forms to solve this problem [Boos06, Fort96, Rein77]. Instead, we call this problem a graph compatibility problem.

By adapting existing graph isomorphism techniques, we have developed a depth-first branch and bound algorithm to perform graph compatibility check. The algorithm, described in the following paragraphs, needs to test various combinations of possible node matching. This can be computationally expensive. Thus, in order 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 search criteria defined in Chapters 3 and 4 are computationally very cheap, if applicable they are used first. Only the assemblies that satisfy all the criteria are retained for further tests.

In the second stage, six other conditions are used to prune the list of feasible database matching multigraphs further. Some of these conditions are analogous to the use of vertex invariants in solving the subgraph isomorphism problem [Boos06, Fort96]. They are listed as follows:

- 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 more 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. By matching set, we mean the set of nodes belonging to the database multigraph, which can be possibly matched with a particular query graph node, based on all the node attributes. This set will help us in pruning certain DFS paths later on as well.

Once pruning has been completed, we identify the root node for depth-first search (DFS). As has been explained in [Boos06], if we can label the vertices properly, we can reduce the search space significantly. The most widely used heuristic is to consider the most constrained nodes first. Typically, lower-degree nodes should be examined before high-degree ones as it enables us to chop off a large portion of the trunk before it gets a chance to branch out. 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. Different weights are assigned to individual factors based on empirical results and a final scaled ranking is assigned to every node. The node with the highest rank is selected as the root node and ties are broken arbitrarily. The database graph node that has the lowest degree greater than or equal to the degree of the root query graph node is chosen as the initial node for matching purposes.

Now, coming to the main DFS algorithm, we first check whether the two initial nodes are compatible. Two nodes are compatible if and only if the database multigraph node is a member of the matching set for the query graph node and a one-to-one correspondence exists between all the edges connecting every pair of nodes that have been already matched. If yes, then we explore all the neighbors of the query graph node and try to match it with one of the neighboring nodes of the database graph node. As long as matching (compatible) nodes are found, we proceed in a depth-wise manner till we encounter a leaf node. Then we backtrack to a previous level unmatched node and try to match it with a *feasible* node in the database graph. A *feasible* node is obtained by considering the unmatched neighbors of the database graph node which is compatible to one of the matched, neighboring nodes of the current query graph node. The algorithm and its explanation is given in Section 4 of this Chapter.

The algorithm is terminated when all the query graph nodes have been matched or no match has been found for at least one of them even after exploring all possibilities. A run-time bounding condition is employed. If the matching set for any unmatched query graph node becomes empty, then that path can be safely ignored and other possible paths should be considered.

## 5.3 Example

Consider a scenario where the designer wants to search for a subassembly commonly used in the organization. The subassembly consists of some bolts and a custom part for which the exact geometry is known. The designer looks for an assembly where another custom part was used along with the parts listed above. The designer knows the mating conditions between the parts and defines the query graph (shown in Figure above). The criteria for the search are specified as follows:

- The mating graph as shown in Figure above
- The number of parts as a target value of 10
- Exact geometry match for one of the custom parts

- Two standard parts in the form of a bolt and a nut

The figure 5.5 shows the layout of the flange assembly.

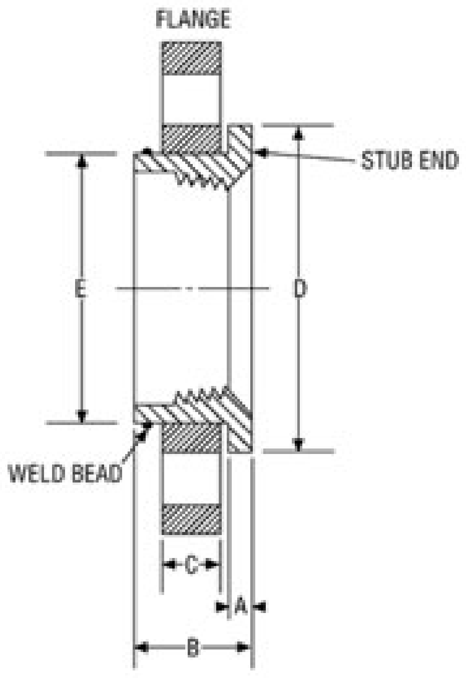


Figure 5.5: Layout of the flange assembly

The figure 5.6 shows a flange which can be adapted by the designer for the new assembly.

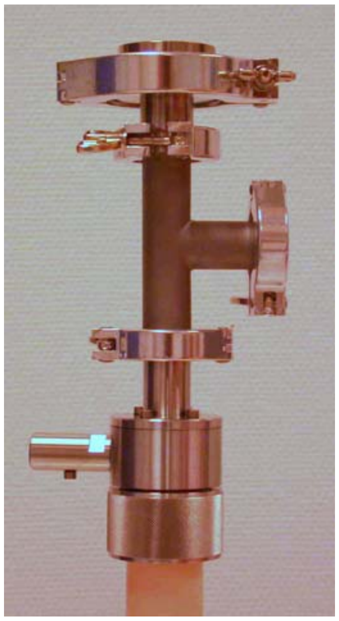


Figure 5.6: A assembly with attached flanges [Nore06]

The assembly shown in Figures 5.7 and 5.8 is an assembly returned by the system that matches the above criteria. In this case, A is the custom part and the bolts shown as B, C, D and E are the standard parts.

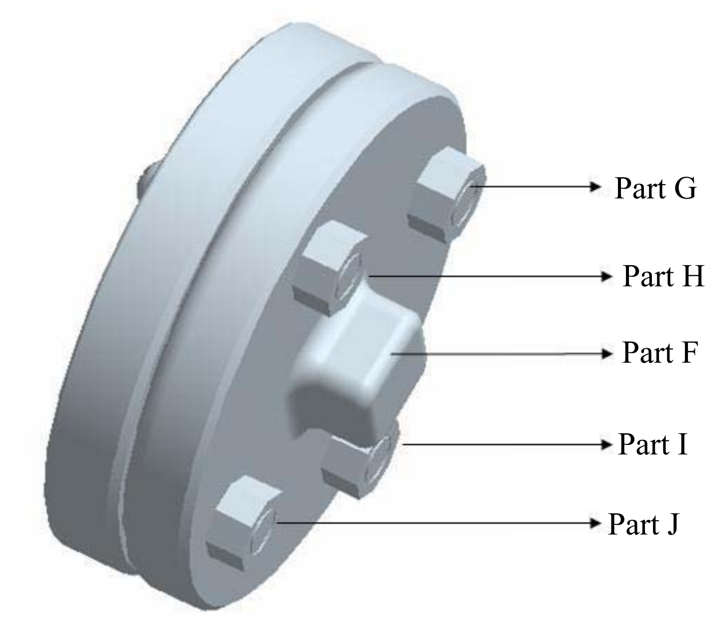


Figure 5.7: A Flange Assembly

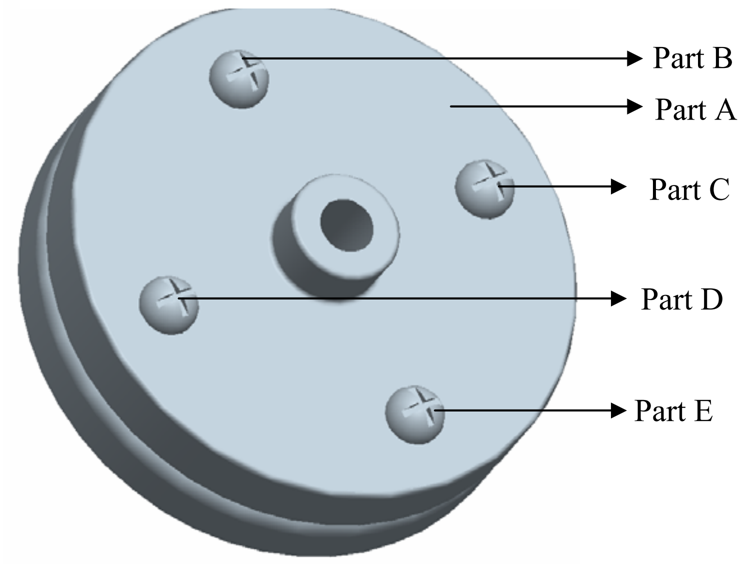


Figure 5.8: Another view of the flange assembly

The figure 5.9 and 5.10 shows another query graph and the result retrieved from the set of assemblies.

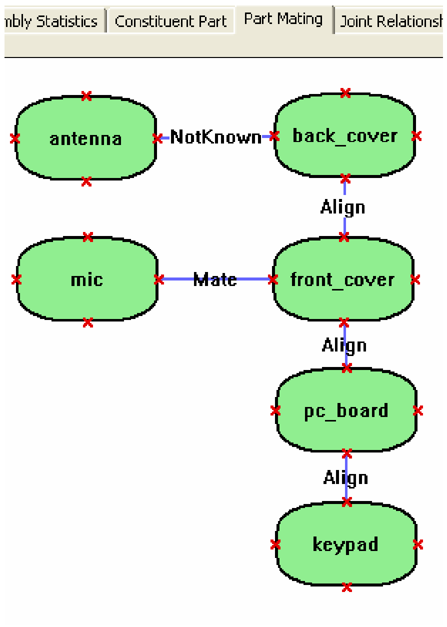


Figure 5.9: A query graph to search for cell phone assembly

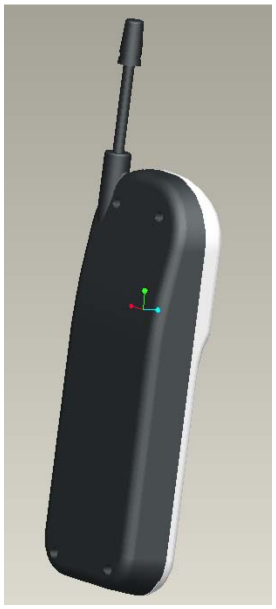


Figure 5.10: An image of the cell phone assembly retrieved from the database of assemblies

## 5.4 Algorithms for initial processing of graph

SearchforCompatibleGraphs is a top level algorithm which calls all other algorithms and controls the entire graph compatibility checking process for all graphs in the database. It takes as input a connected query graph and a list of fully specified connected database multigraphs. It outputs a list of graphs that are compatible with the query graph.

**SearchforCompatibleGraphs(*G1*)**

**Input**

- *G1* = (*V1*, *E1*) represents partially specified connected query graph
- *L1* = the list of fully specified connected database multigraphs
- *G2* = (*V2*, *E2*) represents a member of *L1*

**Output**

- List *L2* of compatible database graphs

**Internal Variables**

- A Boolean variable *CompatiblityCheckStatus*

**Steps**

1) Call AssignPriorityScoreToQueryGraphNodes (*G1*)
2) For each *G2* $\in$ *L1*, call CompatibilityCheck(G1, G2) and set output as *CompatiblityCheckStatus*
3) If *CompatiblityCheckStatus* is TRUE, add *G2* to *L2*

Step 1 calls the function to assign priority score to each node in the query graph. Step 2 recursively calls the functions to check if the query graph is compatible with each database graph. Step 3 adds the compatible database graph to the list of compatible graphs.

The graphs are connected because each node in the graph is connected with some other vertex in the graph. The query is not a multigraph i.e. multiple mating conditions cannot be specified between two nodes in the query graph. The nodes in the query graph are assigned priority score only once during the search over all assemblies in the database. The algorithm iteratively checks whether query graph is compatible with any of the database graphs. The compatible graphs are added to a list of compatible graphs.

The AssignPriorityScoreToQueryGraphNodes algorithm is called by SearchforCompatibleGraphs algorithm. It takes as input the query graph. It assigns a rank to each node from query graph based on the number of nodes attached to each node and other characteristics of the node as specified by the user. This rank is used to identify the root of the depth first search and the most constrained node during each recursion of the depth first search function.

**AssignPriorityScoreToQueryGraphNodes (*G1*)**

**Input**

- *G1*, a query graph

**Output**

- *rank_final*, priority score assigned to each node in *G1*

**Internal Variables**

- *V1* is the set of nodes in *G1*
- $v_i$ is the number of nodes in *V1*

- Degree of a part is the total number of parts attached to the given part in the assembly. It includes the nodes explicitly specified in the query graph and implicitly specified during creation of node. Implicit degree of the node is the number of additional nodes to which the node from query graph is connected but are not shown in the graph. This number is specified while creating the node.
- Category of a part indicates whether a part is of custom category or standard category or if the category is not specified.
- Geometry for a custom part is the path of the part in the database
- Type for a standard part is a type selected from the list of standard parts available in the organization

**Steps**

1) For each $v_i \in V1$, assign *rank_absolute* starting from least possible value of zero and highest possible equal to $v_i$-*2*. If two or more nodes have the same degree, then assign the same *rank_absolute* to all such nodes.
2) Scale all *rank_absolute* between 0 and 1 to assign *rank_relative* to every node. The highest *rank_relative* can at the most be equal to 1 and the lowest *rank_relative* is at least 0.
3) For each $v_i \in V1$, assign *rank_intermediate* based on the following rules:
   a. If part is of custom category, then assign 0.5 points to the node
   b. If part is of standard category, then assign 0.25 points to the node
   c. If part geometry is known, then assign 0.5 points to the node
   d. If part category is known, then assign 0.25 points to the node
   e. *rank_ intermediate* = $\sum$ points assigned to node

4) For each $v_i \in V1$ assign

   $$rank\_final = w1 \times rank\_relative + w2 \times rank\_intermediate,$$

   where $w1 = 1$ and $w2 = 2$

5) Sort the list of nodes in non-ascending order of their *rank_final*

The step 1 is based on the degree of the node. The step 2 scales the *rank_absolute* between 0 and 1 and assigns *rank_relative*. The node with highest degree will have *rank_relative* as 0 and node with lowest degree will have *rank_relative* as 1. Step 3 assigns priority score based on other node invariants i.e. category, geometry and type. Step 4 computes the final priority score value for each node. Finally, in step 5 the nodes are sorted in descending order of their priority score.

The node invariant properties are used to assign priority score to node. According to this algorithm, the node which is most constrained in terms of all invariant properties will have highest priority score. The algorithm assigns highest *rank_absolute* to a node with least degree i.e. the most constrained node. If multiple nodes have the same degree, then they are assigned same priority score. As the highest priority score is always equal to the number of nodes in the query graph, some ranks will not be assigned to any node in case of multiple nodes with the same degree. The priority score is normalized. A node representing a custom made part and for which geometry is specified, is the ideal candidate to act as the root node for DFS as it is the most constrained node. The total number of custom parts for which a geometry match is possible are very less and this reduces the total number of iterations required for compatibility check. Custom parts are given more weightage as compared to standard parts. This is based on the understanding that custom parts will be used in fewer assemblies than standard parts and thus are better suited to act as the root for DFS. Also, more weight is assigned to a custom part with known geometry because it will match with very few parts in database. The highest *rank_intermediate* can have value equal to 1. Final rank is the weighted sum of ranks based on degree and other node variants. The weight for priority score on degree and other node invariants are chosen empirically. Lastly, the node with highest sum total of all ranks is the most constrained node in query graph and will act as the DFS root.

The SearchforCompatibleGraphs algorithm in step 2 calls CompatibilityCheck algorithm. It takes as input the query graph and a fully specified database multigraph. It calls all algorithms to prune and verify the compatibility of the two graphs. It outputs a Boolean variable that indicates the status of compatibility between the two input graphs. True indicates that the graphs are compatible while false indicates that the graphs are incompatible.

**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
```

Step 1 calls the function to check the pruning conditions. This function returns a Boolean value and if the database graph passes the initial pruning step then we proceed to step 2. We have a list of nodes arranged in non-ascending order of priority score. The *matching_set* is created if this function does not return a FALSE value. Step 2 sets the status of the global variable. Step 3 gets the element at the front of the list of nodes from query graph. The front node will have the highest priority score amongst all nodes. Step 4 lists all nodes from database graph that are feasible matches for the selected node in a non-descending order. Step 5 iterates over the list arranged in step 4 till the graphs match or the list is exhausted. At the beginning of each iteration, a list of matched nodes and two lists of nodes from query and database graph respectively, that have been discovered in depth first search (DFS) but not matched to any other node are initiated to empty lists. The global list of feasible matches for all nodes i.e. *matching_sets* is passed as an argument to *Match_Node* function and the copy gets modified during each recursion of the function.

This is the main algorithm to check for graph compatibility. The first step prunes out the database graphs that cannot match with query graph before use of DFS. A priority score is assigned to all nodes in the query graph to find the most constrained node. The most constrained one is used as the root of DFS. The use of the most constrained node as root has been suggested in literature to improve computational time by decreasing number of possible matches to be checked.

The PreliminaryCompatibilityTest algorithm is called by CompatibilityCheck algorithm. This algorithm implements all pruning conditions before actual depth first search is performed on the query graph and the database graph. It takes as input the query graph and the fully specified database multigraph. It outputs a Boolean variable to indicate if the database graph passed the pruning checks. A true value indicates that the database graph is not pruned and depth first search needs to be performed while a false value indicates that database graph is pruned and depth first search need not be performed.

## Section 5.5 Pruning Algorithms

**PreliminaryCompatibilityTest (*G1, G2*)**

**Input**

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

**Output**

- Boolean *preliminary_compatibility_status* as return value to indicate whether graphs can be compatible according to preliminary check

**Internal Variables**

- $v_1$ represents the number of nodes in *G1*
- $v_2$ represents the number of nodes in *G2*

- $e_1$ represents the number of edges in *G1*
- $e_2$ represents the number of edges in *G2*
- *A* is a set of integers: each $a \in A$ represents the degree of $v_i \in V1$
- *B* is a set of integers: each $b \in B$ represents the degree of $v_j \in V2$

**Steps**

1) If $v_1 > v_2$, set *preliminary_compatibility_status* as FALSE and return
2) If $e_1 > e_2$, set *preliminary_compatibility_status* as FALSE and return
3) If number of standard parts in the query graph > number of standard parts in the database graph, set *preliminary_compatibility_status* as FALSE and return
4) If number of custom parts in the query graph > number of custom parts in the database graph, set *preliminary_compatibility_status* as FALSE and return
5) Call CheckOnetoOneMappingofNodes(*G1*, *G2*) and set *preliminary_compatibility_status* as the output.
6) Return *preliminary_compatibility_status*

Step 1 checks that number of vertices in database graph is equal to or greater than number of vertices in query graph. Step 2 checks that number of edges in database graph is equal to or greater than number of edges in query graph. Step 3 and Step 4 check that the database graph has at least equal number of standard and custom parts as those specified in the query graph. Step 5 calls the function to check one to one mapping of nodes. This function ensures that every node from query graph has at least one feasible match among the nodes of database graph.

The database graph represents an assembly in the database. The query graph and the database graph are related by injective function because every $v_j \in V2$ should map to one and only one $v_i \in V1$. For a database graph to be compatible with the query, it should at least have the same number of nodes and edges as the query graph. This is the primary requirement for graphs to match and is checked in step 1 and 2. Based on the injective function, it follows in step 3 that number of standard parts in database assembly should be at least equal to or greater than number of standard parts in query graph. The same condition is applicable for custom parts and is checked in step 4. The fifth condition checks whether injective function exists only on the basis of degree. If for every *a* there does not exist a distinct *b*, than at least one node from query graph will not have a node with equal or higher degree from database graph. A complete injective function cannot exist and thus the two graphs cannot match.

CheckOnetoOneMappingofNodes algorithm is called by PreliminaryCompatibilityTest algorithm. This algorithm checks that for every node from query graph at least one distinct match exists in database graph. It takes as input the query graph and a fully specified database multigraph. It has two outputs. The first output is a Boolean that indicates if a match exists for every node from the query graph. If a match exists than it creates sets of all possible matches for every node from query graph. It acts as a pruning condition and its output is used in a bounding condition during the depth first search.

**CheckOnetoOneMappingofNodes(*G1*, *G2*)**

**Input**

- *G1,* a query graph

- *G2*, a database graph

**Output**

- Boolean value *mapping_exists*. The Boolean is TRUE if one to one mapping exists between nodes of query graph and database graph and FALSE if it does not exist
- *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

**Internal Variables**

- *degreee_a* represents the total number of neighbors for any node *a*
- *category_a* represents the category of any node *a*. The possible values are custom, standard or not specified
- *geometry_a* represents the geometry of any node *a*. The possible values are path to the geometry or not specified
- *type_a* represents the type of standard part that node *a* represents. The possible values are any entity from the list of standard parts or not specified.

**Steps**

1) Populate *QueryDegreeArray* with degree of nodes in query graphs and *DatabaseDegreeArray* with degree of nodes in database graph
2) Sort *QueryDegreeArray* and *DatabaseDegreeArray* in non-descending order using Bubble sort
3) If for $\vee$ $a \in$ *QueryDegreeArray* there does not exist a distinct $b \in$ *DatabaseDegreeArray* : $a \leq b$, set *mapping_exists* as FALSE and return
4) For every $p \in V_1(G_1)$, do the following

    a. Pop the first element from the list of nodes in the database and set it to *q*. When the list of nodes in the database ends, go to step 4.
    b. For every *p* and *q*, do the following
        I. *degree_p* ≤ *degree_q* go to 4-b-II else go to step 4-a
        II. *category_p* is not specified or *category_p* = *category_q* go to step 4-b-III else go to step 4-a
        III. if *category_p* = custom then either *geometry_p* is not specified or *geometry_p* = *geometry_q* go to step 4-b-IV else go to step 4-a.
        IV. if *category_p* = standard then either *type_p* is not specified or *type_p* = *type_q* go to step 4-c else go to step 4-a.
    c. Add *q* to the *matching_set* of *p*

5) Set *mapping_exists* as TRUE and return

Step 1 populates two arrays with the degree of nodes of query graph and database graph. Step 2 sorts the two arrays using bubble sort in non-descending order. Step 3 runs two for loops over the two arrays to check one to one mapping of elements. Step 4 creates *matching_set* for all nodes in query graph. Step 4 cycles over all the elements in the query node. Step 4-a cycles over all the elements in the database node. Step 4-b compares the two nodes with each other. Step 4-b-I compares the degree, step 4-b-II compares the category, step 4-b-III compares the geometry for custom parts and step 4-b-IV compares the type for standard parts. If at any of the step in the in 4-b, the nodes do not match, then the algorithm returns to 4-a and iterates with the next node from the nodes in database graph. Step 4-c adds the *q* to the set of feasible matches for node *p*.

The main step in this algorithm is to find injective relationship between elements in two arrays. The first two steps populate and sort the arrays. The third step loops over the two arrays to find if an injective relationship exists. The fourth step populates the matching sets i.e. a list of nodes for every node from query graph that the query node can match. These matching sets are used in the both bounding conditions during the DFS.

## Section 5.6 Depth First Search algorithm

MatchNode is the depth first search recursive function. This algorithm is initially called by CompatibilityCheck algorithm in step 5-d. It then recursively calls itself to create a search tree. It takes as input a node from query graph, a node from database graph, a list of pair of nodes that have already been matched, a list of nodes from query graph and a list of nodes from database that have been discovered in the depth first search but not yet matched and a list of possible matches for each node in query graph. It outputs a Boolean value to indicate whether the query graph and database graph is compatible.

**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 depth first search (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.
```

Step 1 calls the function to check whether the two input nodes can match. This function checks if the two nodes match and if they match, then algorithm proceeds to step 2. Step 2 checks if all unmatched from the query graph have at least one feasible match. If they do not have at least one feasible match, then the function returns FALSE. Step 3 updates the list of nodes that have been matched by appending this pair of nodes. Step 4 deletes the list of feasible matches of node *p* from the *matching_set* and also deletes *q* from list of feasible matches of all unmatched nodes from query graph. Step 5 checks whether the query graph and database graph are found to be compatible using termination condition. The condition checks if the number of pairs of matched nodes is equal to the number of nodes in query graph. If the termination condition is satisfied then the global output is set to TRUE and the recursive function performs a global exit. Step 6 lists all neighbors of the node *p* from query graph. Step 7 checks whether the list of neighbors of *p* is empty. If the list is empty, then *p* is leaf node and the function proceeds with other unmatched nodes from step 14. If the node is not a leaf node, the algorithm goes to step 8. Bubble sorting is used as the size of the array is very small. The implementation is borrowed from [Lamo06]. Step 8 sorts the list obtained in step 6 in non-descending order of the degree of nodes. Step 9 places all nodes that are connected to input node by an unknown edge type at the end of the list. If the elements of the sorted list are not already a part of the list of unmatched nodes, step 10 appends them at the beginning of the list of unmatched nodes from query graph. Step 11 lists all neighbors of input node from database graph that are not present in the list of matched nodes. Step 12 sorts the list obtained in step 11 in non-descending order of the degree of nodes. Step 13 appends each element from sorted list obtained in step 12 to the list of unmatched nodes of the database graph starting from the beginning of the list.

Step 14 iterates over all elements in the list of active from query graph. Step 14-a lists all neighboring nodes that are also members of the list of matched nodes for the node selected from in step 14. Step 14-b iterates over all elements of the list found in step 14-a. Step 14-b-i pops the first element in the list found in 14-a. Step 14-b-ii finds the node that matches with node popped in step 14-b-i. Step 14-b-iii lists all neighbors of node found in the 14-b-ii. Step14-b-iv iterates over the list found in 14-b-iii by popping each element in the list. Step-14-b-iv-i checks if edge between nodes found in 14-b-i and 14-1 and the edge between nodes found 14-b-iii 14-b-iv is compatible. If the nodes are compatible then algorithm goes to step 14-b-iv-ii. If the edges are compatible then algorithm proceeds to next step. Step 14-b-iv-i-I removes the node found in 14-a from the list of active nodes from query graph. Step 14-b-iv-i-II removes the node found in 14-b-iv from the list active nodes from database graph. Step 14-b-iv-i-III recursively continues the algorithm with the nodes removed from the list of active nodes, the updated lists of unmatched nodes from query and database graph, the updated list of matched nodes and the updated *matching_set*. Step 14-b-iv-ii pops the next element from the list found in step 14-b-iii and returns the control to step 14-b-iv-i. Step 14-b-v is reached if the list found in step 14-b-iii is empty. It pops the next element from the list found in step 14-a and takes the control back to step 14-b-ii.

This algorithm is the main depth first search (DFS). The input to the algorithm consists of a node each from query graph and database graph, two lists consisting of nodes from query and database graph that have not been matched and a list of pairs of matched nodes. The algorithm first checks whether the two nodes match with each other by calling function CheckNodeConsistency. If the nodes match with each other, the algorithm checks two bounding conditions. It ensures that at least one match is available for every unmatched node from the query graph. The algorithm then updates a copy of the list of matched nodes and a copy of the feasible matches for unmatched nodes. This update accounts for the two nodes that have been matched in this step. The algorithm then checks for terminating condition. If the size of the list of matched nodes is equal to the size of the list nodes in query graph, it implies that a match has been found for all nodes from the query graph. Thus the algorithm sets a global output and terminates all recursions.

The algorithm then lists all neighbors of node from query graph and sorts them in the ascending order of degree of nodes. If the list is empty, it implies that a leaf node has been reached. In such case the algorithm iterates with the next available node from the list of active nodes of the query graph. In addition, for this list of query graph nodes, the nodes having an unknown edge type are placed at the back of the list. This is because unknown edge type is a wild card entry and can match to edge of any type. The constraints are lesser as compared to edge that has a specific type and more constrained nodes are given priority. The elements in the sorted list are appended to the list of unmatched nodes at the beginning if they are not already a part of the list. As they are appended at beginning and the nodes are later selected from the beginning of the list, the search always proceeds in depth first method. Also, the check to ensure that elements are not already a part of the list accounts for the fact that some nodes can be reached from multiple paths and should not be appended more than once. In the next step the algorithm makes a list of all neighbors of input node from database graph, sorts it in ascending order of degree and appends the elements to the list of unmatched nodes from database graph with the same rules as described above.

The algorithm then iterates over the list of active nodes from query graph. For each element, it finds a list of neighbors that have been matched from the list of matched nodes. It then finds the matching nodes from database graph and list of its unmatched neighbors. The algorithm then tries to iteratively match the node from query graph to all such neighbors by using recursion. If a match is not found, then algorithm returns a match failure status. The algorithm then retracts back to an instance where an alternative path is available for search. If all paths have been explored and a match has not been found then the recursive algorithm terminates with a match failure status.

## Section 5.7 Auxiliary Algorithms for Depth First Search

The CheckMatchNodeAvailability algorithm is called by MatchNode algorithm It takes as input a node from query graph, a node from database graph, a list of node pairs that have already been matched and a list of possible matches for each node from query graph. It checks whether the node from database graph is the only possible match to a unmatched node from the query graph except the node currently being matched. This algorithm is used to implement bounding condition during the depth first search.

**CheckMatchNodeAvailability (*p, q, match_nodes_list, 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 depth first search (DFS)
- *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**

- *matched_node_available*, a Boolean

**Steps**

1) Initialize *matched_node_available* to TRUE.
2) While *matching_set* for all nodes in query graph are not explored and *matched_node_available* is not FALSE, do the following
    a. If $q$ is the only element $\in$ *matching_set* of any node $v_i \in V_1(G_1)$ except $p$, set *matched_node_available* as FALSE and go to step 3, else go to *matching_set* of next node in query graph
3) Return *matched_node_available*

Step 1 initializes the output of the function to TRUE. Step 2 iterates over the list of feasible matches from the *matching_set* for all nodes of query graph. Step 3 checks if $q$ is the only element in the list of feasible matches from *matching_set* for all other node except $p$. If $q$ is found to be the only member of a list of feasible matches of any other node, the output is set to FALSE and function terminates.

This function ensures that the node from database being matched is not the only possible match for any unmatched node except $p$ from query graph. If the database node being matched is the only possible match for any unmatched query node, then the query node will not have a match and the graphs cannot be compatible. Thus, this path is not feasible and an alternative path needs to be explored. In such case, the function returns false. If all database nodes have another possible match, then the algorithm can proceed for matching other nodes in the query graph.

The CheckNodeConsistency algorithm compares node from query graph and database graph. It compares the two nodes and checks if the node from database graph satisfies any cyclic relation that exists for the node from query graph. Its input is a node from query graph, a node from database graph and a list of pair of all nodes that have been matched. It outputs a Boolean that indicates whether the two nodes match.

**CheckNodeConsistency(*p, q, matched_nodes_list*)**

**Input**

- *p*, a node from query graph
- *q*, a node from database graph
- *matched_nodes_list*, the list of pair of nodes that have been already matched during depth first search (DFS)

**Output**

- *node_consistency_result* , a Boolean as return value to indicate whether the two nodes matched

**Internal Variables**

- *degreee_a*, the total number of neighbors for any node *a*
- *category_a*, the category of any node *a*. The possible values are custom, standard or not specified
- *geometry_a*, the geometry of any node *a*. The possible values are path to the geometry or not specified
- *type_a*, the type of standard part that node *a* represents. The possible values are any entity from the list of standard parts or not specified.

- *pair(x, y)*, the pair of nodes in list of matched nodes where *x* is from query graph and *y* is from database graph
- *edge(a, b)*, the edge between node *a* and node *b*

**Steps**

1) If
   a. *degree_p* $\leq$ *degree_q* and
   b. *category_p* is not specified or *category_p* = *category_q* and
   c. if *category_p* = custom then either *geometry_p* is not specified or *geometry_p* = *geometry_q* and
   d. if *category_p* = standard then either *type_p* is not specified or *type_p* = *type_q*

   then go to step 2 else set *node_consistency_result* as FALSE and return
2) Find a list $P'$ of all the neighboring nodes of *p* that are already in *matched_nodes_list*
3) For every $x \in P'$, do the following:
   a. Find the pair *(x, y)* in *matched_nodes_list*
   b. If *edge(y, q)* is not compatible to *edge(x, p)*, than set *node_consistency_result* as FALSE and return
4) Set *node_consistency_result* as TRUE and return

Step 1 checks whether the degree, category, geometry and type match for the input nodes. Except degree, all inputs are optional. Geometry can only be specified if part is of custom category. Type can only be specified if part is of standard category. Steps 2 and 3 are included for checking cyclic relations in the query graph. Step 2 lists all neighbors from the list of matched nodes that have an unexplored edge with the input query graph node. Step 3 iterates over the list found in step 2. Step 3-a finds the pair of nodes from the list of matched nodes that contain node obtained in step 2. The first element of the pair is a node from the query graph and second element is a node from database graph. Step 3-b verifies if an edge exists between the input database graph node and second element of the pair obtained in 3-a that is compatible with the edge between input query graph node and the first element of the pair. If such edge does not exist, then the two input nodes do not match.

If the input node from query graph has an unexplored edge with a node from the query graph that is already in the match list, then a cyclic condition exists in the query graph. If the node from query graph has a cyclic relation, then its matching node from database should also have a cyclic relation. Such cyclic relations are checked in step 2 and 3.

## 5.8 Computational Experiments

A database of 200 database graphs was created randomly. The number of nodes in each graph varies between 10 and 100. We conducted three experiments to test the computational speed and robustness of the graph compatibility algorithm. The database graph used to prepare the query for all experiments consists of 24 nodes.

The first experiment uses varying number of nodes in the query graph. This experiment qualitatively estimates the time complexity of the algorithm in terms of the number of nodes in query graph. The graph in figure 5.11 shows the plot for number of nodes in the query graph against the computation time for depth first search. The number of nodes in query graph varies from 5 to 17. The query graph used in this experiment was fully defined i.e. it did not have any wild cards. The query graph was prepared from one of the graphs from the database. As can be observed from the graph, the time required for depth first search 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 and querying and updating of the list of nodes being currently processed both in the query graph and the database graph. This increases the amount of time required for depth first search operation.

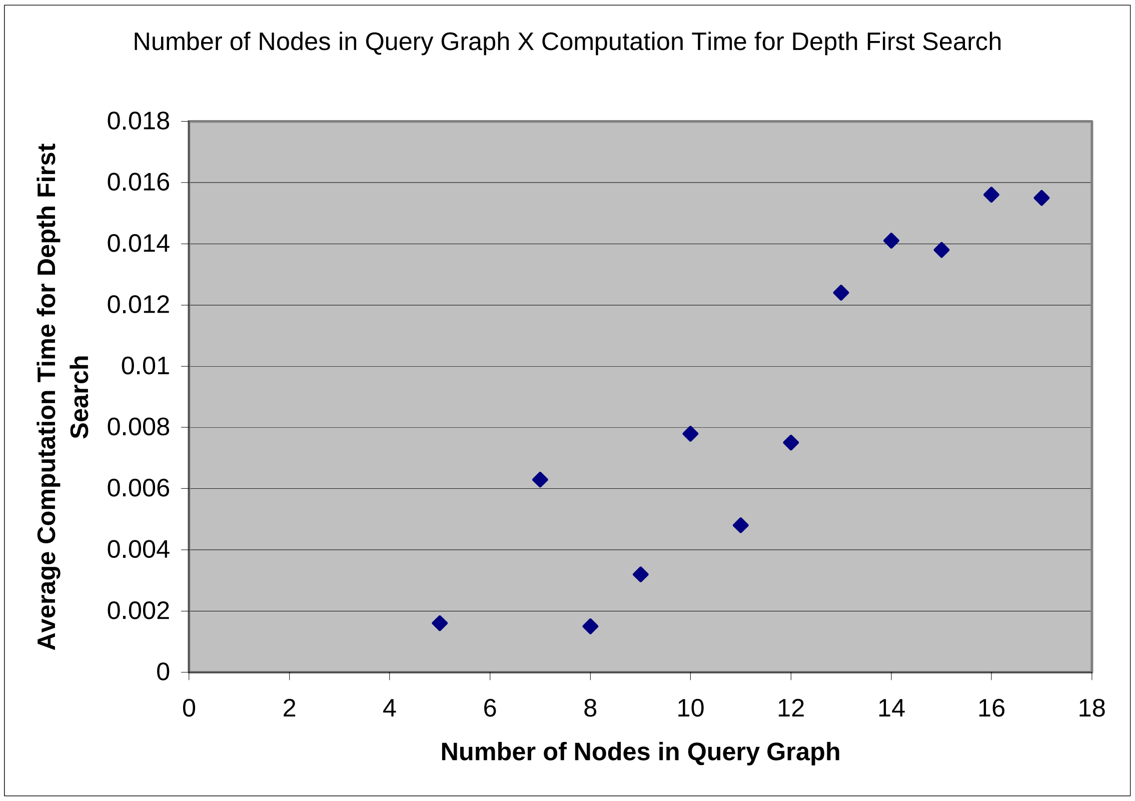


Figure 5.11: Number of Nodes in Query Graph
X Computation Time for Depth First Search

The second experiment used different degrees of nodes in the query graph, keeping the total number of nodes constant (11). This experiment tests the robustness of the algorithm as we increase the number of edges in the graphs. The graph in figure 5.12 shows the plot for number of nodes in the query graph against the computation time for depth first search. The edges in query graph vary from 10 to 15. The query graph used in this experiment was fully defined i.e. it did not have any wild cards. As can be observed from the graph, there is no considerable change in time required for depth first search. The maximum variation in the timing of depth first search is of the order of 0.01 s 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 depth first search significantly. Thus, the horizontal nature of the graph is easily explained from a theoretical standpoint.

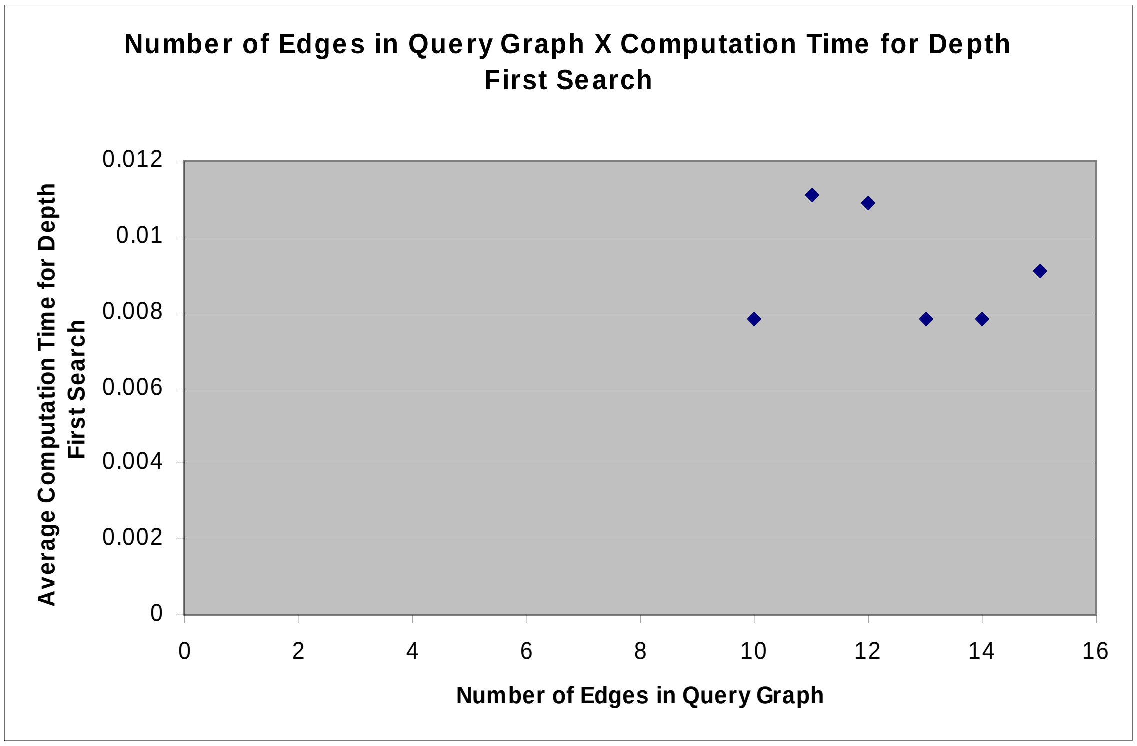


Figure 5.12: Number of Edges in Query Graph
X Computation Time for Depth First Search

The third experiment used query graphs defined with varying amount of information given about the node and edge attributes, keeping both the total number of nodes and edges fixed at 10 and 9 respectively. 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. If all optional inputs have been defined for all nodes and edges in a query graph, then the graph is termed as fully specified. This experiment measures the robustness of the algorithm with varying percentage of wild card entries in a query graph. The percent of wild card entries in the query graph is 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 below shows a plot of computational time against percentage of wild card entries in the query graph.

The graph in figure 5.13 shows the plot for percentage of wild card entries in the query graph against the computation time for depth first search. The query finds one matching assembly up to 31% wild cards. The number of results increases thereafter producing 15 results out of a database of 200 graphs. In the final query graph, 79% wild cards were used. As can be observed from the graph, the time required for depth first search increases with the increase in number of matching results. When the amount of wild card entries for nodes is increased, multiple possible matches (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 is going to increase. If the category is also not specified, it can result in further increase in the size of the matching sets. Moreover, every edge has an optional input (attribute) in form of the type of mating condition that it represents. If the edge attribute is left unspecified, then multiple paths can be explored in DFS. As the number of possible matches and explored edges for every possible match are increasing 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.

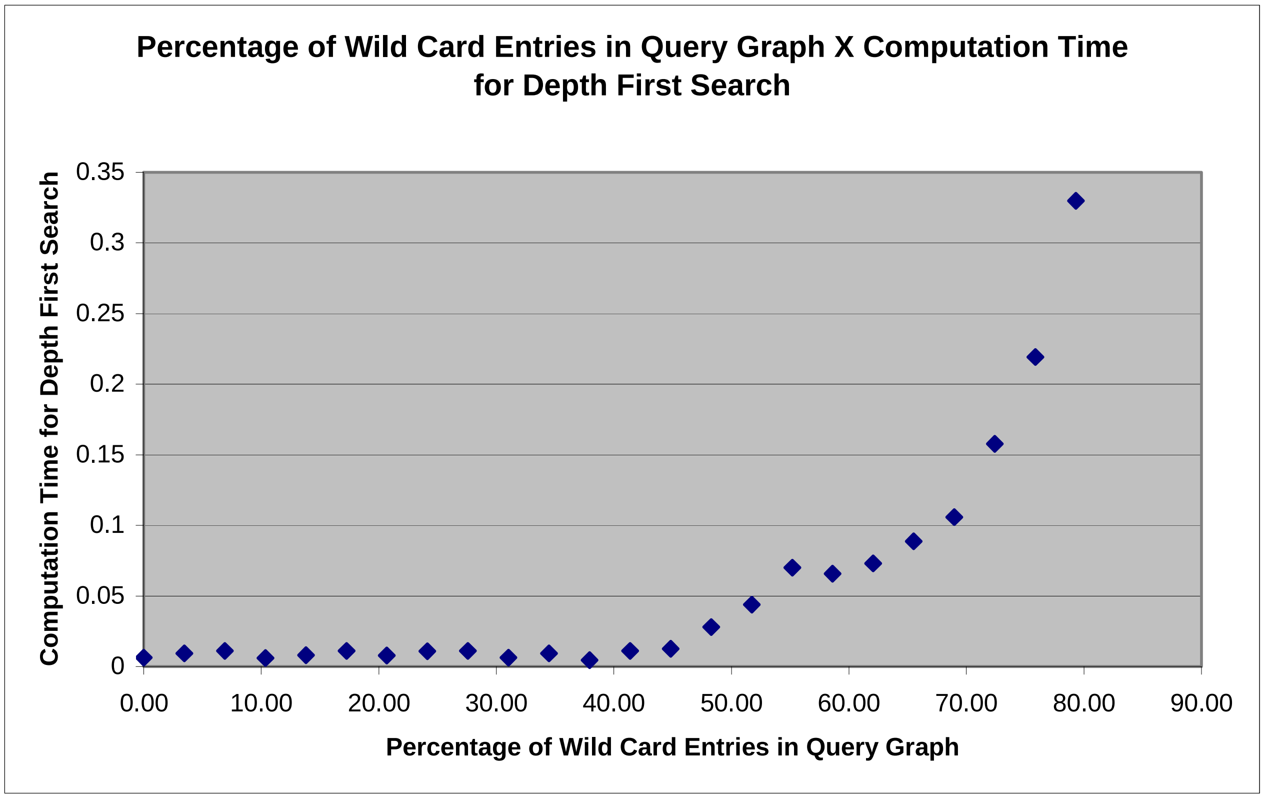


Figure 5.13: Percentage of Wild Card Entries in Query
Graph X Computation Time for Depth First Search

It can be inferred from these experiments that graph compatibility algorithm is computationally fast, efficient and robust for query graphs with varying number of nodes, varying number of edges of graphs and varying amount of optional criteria specified in the query graph.

To create the test database of 200 graphs a random graph generator program was created. This can generate a graph with number of nodes between 10 and 100. The random graphs can consist of any number of parts selected at random from a database of 200 custom parts and 25 standard parts. The mating condition is randomly selected from the available list of mating conditions in Pro/Engineer Wildfire Educational Edition.

Query graph for the experiment was created from one of the graphs generated for the performance testing.

# Chapter 6

# SEARCH BASED ON JOINT RELATIONS

This chapter is arranged in the following manner: Section 6.1 describes the relationship between joints that can be used to define search in this criteria, Section 6.2 discusses the methods used for searching based on the characteristics and Section 6.3 illustrates the use of this criteria with an example.

Consider a scenario where a robot designer is searching for an assembly consisting of two revolute joints at right angles to each other to mimic the motion of a human arm as shown in the figure 6.1. The main characteristic of this assembly is the joints defined between these parts. A typical search for such an assembly would be a manual search. Another possible way to search for this assembly is to specify the type of joint and their orientations with respect to each other. The joints in articulated assembly are important to define the function of the assembly. We propose a set of criteria to search based on joints and their interrelationships.

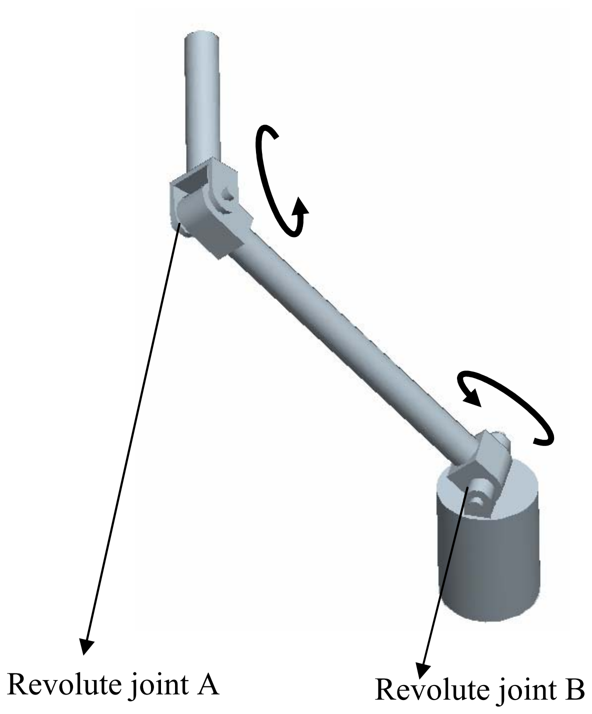


Figure 6.1: A robot arm with two revolute joints allowing perpendicular motion and attached on one link

## 6.1 Search Definition

In the search tool, the designer specifies the type of joint and the orientation between the joints. The designer can select joint of any type that is available in Pro/Engineer. The available choices are listed in Section 2 of Chapter 3. The possible orientations between joints are as follows while the figure 6.2 shows the definition of a query:

- *Parallel*
- *Perpendicular*
- *Angle* (This is specified as range value for angle between the two joints)
- *Unknown relationship*
- *No relationship*

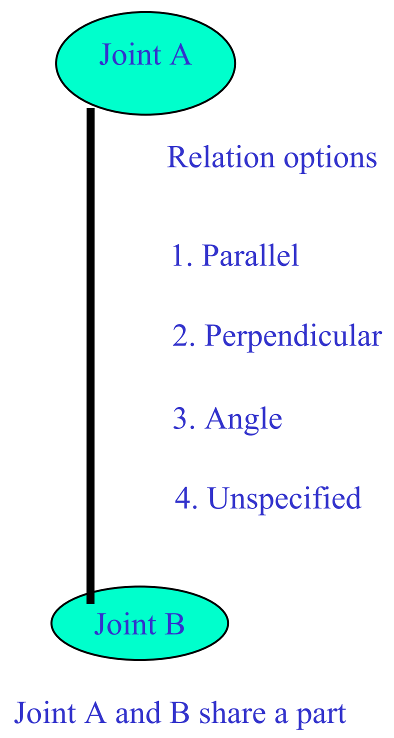
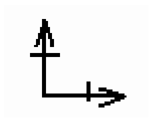
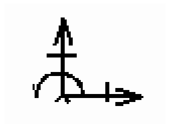
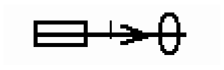
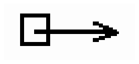
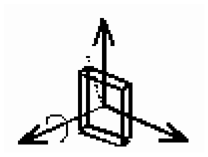
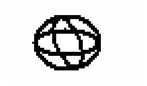
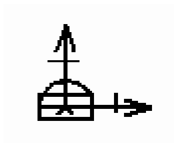
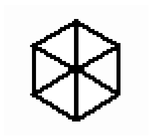


Figure 6.2: A query definition

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 a relation cannot exist between two joints, then the designer can choose *No relationship*. The designer will provide a set of joints that are the defining characteristics for the assembly and the relationships among them. No relation between joints can be defined if one of the two joints is of the type of *ball*, *cam-follower*, *slot-follower, weld* or *gear-pair*. In case of *planar* joint the angle is assumed to be defined with respect to (w.r.t.) the plane of joint. In case of *gimbal* joint, the angle is assumed to be defined w.r.t. the rotor axis. The angle for *bearing* joint is defined w.r.t. the axis of the bearing. The angle for a *U joint* (i.e. a Universal joint) is defined w.r.t. the axis perpendicular to both axes about which rotation is possible. The angles are assumed to be defined at the zero position of the joints in Pro/Engineer. The axis of each type of joint is shown in the table below. [Proe04]

| **Joint Type** | **Degrees of Freedom** | | **Description** |
|---|---|---|---|
| | **Rotational** | **Translational** | |
| Pin | 1 | 0 | Sometimes known as a revolute joint. |
| U-joint  | 2 | 0 | |
| Gimbal  | 3 | 0 | Unlike ball joints, gimbal joints have distinct axes. A common example of a gimbal joint is a gyroscope.<br><br>Axis 1 is the outer gimbal axis, axis 2 is the inner gimbal axis, and axis 3 is the rotor axis. |

| | | | |
|---|---|---|---|
| Cylindrical  | 1 | 1 | |
| Slider  | 0 | 1 | |
| Planar  | 1 | 2 | Bodies connected by a planar joint move in a plane with respect to each other. |
| Ball  | 3 | 0 | Rotational degrees of freedom are without respect to a fixed axis. |
| Bearing  | 3 | 1 | The first axis allows both rotation and translation. |
| Weld | 0 | 0 | This joint allows no relative movement between bodies. It welds two bodies together. |

|  |  |  |  |
|---|---|---|---|
|  |  |  | Pro/MECHANICA displays axes for this joint only when you edit it. In this case, it shows two alignment axes on each body. |

The 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 body 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 manually and stored. The system searches for an assembly that has all the query doubles in the list of database doubles.

Consider the robot assembly shown in Figure above. The list of doubles for this assembly is given as follows:

$$D_1 = \{\{revolute, revolute\}, perpendicular\}$$
$$D_2 = \{\{revolute, otherjoints\}, unspecified\}$$
$$E_1 = \{D_1, D_2, ..., D_n\}$$

*$D_1$* is the first double. The assembly contains other doubles which have not been specified. *$E_1$* is the list of doubles representing the entire assembly. The two joints in *$D_1$* define the range of workspace for the robot. They are oriented perpendicular to each other. A possible way to search for this assembly is by only using the double *$D_1$* in the list of query doubles. Such a query list will be able to locate the robot assembly as a possible match from a list of assemblies.

## 6.2 Search Method

The input is received as a list of doubles. For an assembly from the database to satisfy the search criteria, it should contain all the joints with relationship between them as defined in each of the query doubles. Each double from the list of query doubles is compared to a double from the list representing the joint-pairs in the database assembly. This is done by exact string matching. The relation between query doubles and database doubles is injective. A distinct match for every input double must be found in the list of doubles present in a particular database assembly, in order to match it with the query. The figure 6.3 shows the method of search.

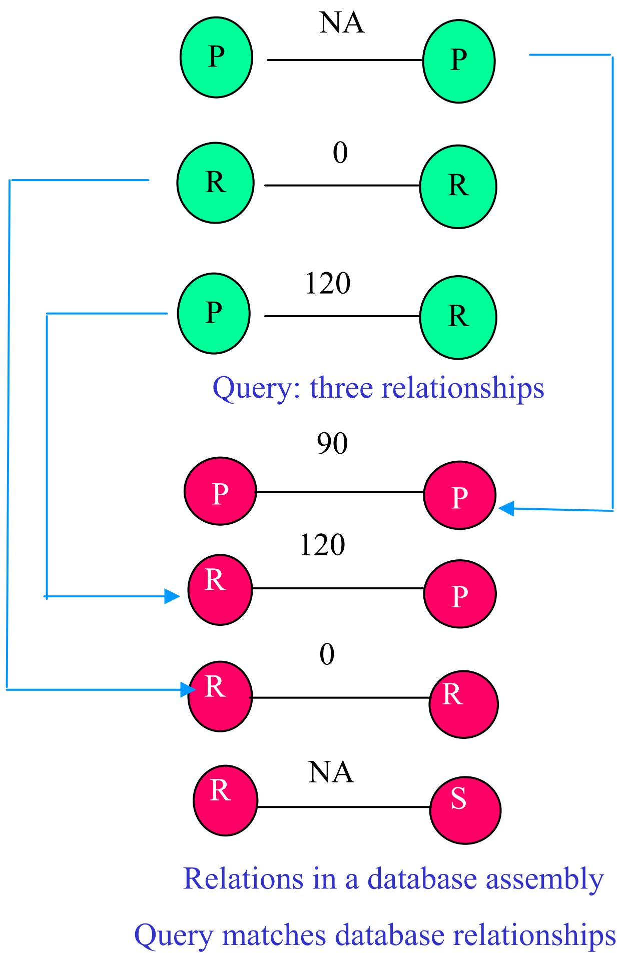

NA
P
P
0
R
R
120
P
R
Query: three relationships
90
P
P
120
R
P
0
R
R
NA
R
S
Relations in a database assembly
Query matches database relationships

Figure 6.3: A query matching with a database assembly

## 6.3 Example

Consider a scenario where the designer wants to search for a spatial mechanism that allows a base to have translational motions along three axes, 120 degrees apart from each other and lying in one plane. Another translational motion is possible along an axis which is perpendicular to the earlier plane. This motion can be achieved by using three ball joints and one slider joint. The designer provides a list of doubles to search for such a mechanism. The designer has a restriction on the total number of joints in the assembly, and the size and total number of parts in the assembly. The criteria for search are specified as follows:

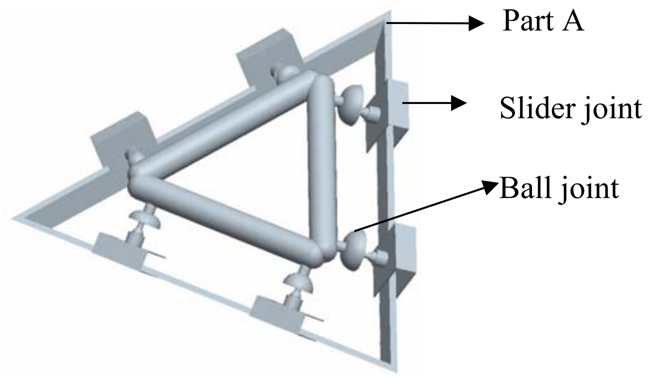
- The list representing doubles as shown below

$$\begin{aligned}D_1 &= \{\{ball, ball\}, Norelation\}\\ D_2 &= \{\{ball, ball\}, Norelation\}\\ D_3 &= \{\{ball, slider\}, Norelation\}\\ E_1 &= \{D_1, D_2, D_3\}\end{aligned}$$

- The total number of joints in the assembly (should not exceed 20)
- The number of ball joints in the assembly (should not exceed 8)
- The number of slider joints in the assembly (should not to exceed 6)
- The bounding sphere volume of the assembly (should be between 2 and 3 inches)
- The total number of parts in the assembly (a target value of 13 is specified)

With these criteria, the system is able to retrieve the Stewart platform shown in figure 6.4. This mechanism comes closest to the designer's requirement. The platform uses 6 ball and 6 slider joints. The designer can get the required motion by editing the geometry of Part A. The list of doubles of the Stewart platform is shown below:

$$\begin{aligned}D_1 &= \{\{ball, ball\}, Norelation\}\\ D_2 &= \{\{ball, ball\}, Norelation\}\\ D_3 &= \{\{ball, ball\}, Norelation\}\\ D_4 &= \{\{ball, slider\}, Norelation\}\\ D_5 &= \{\{ball, slider\}, Norelation\}\\ D_6 &= \{\{ball, slider\}, Norelation\}\\ E_2 &= \{D_1, D_2, D_3, D_4, D_5, D_6\}\end{aligned}$$



The mechanism has 6 slider and 6 ball joints

Figure 6.4: A match for search by joint relationship – Stewart platform

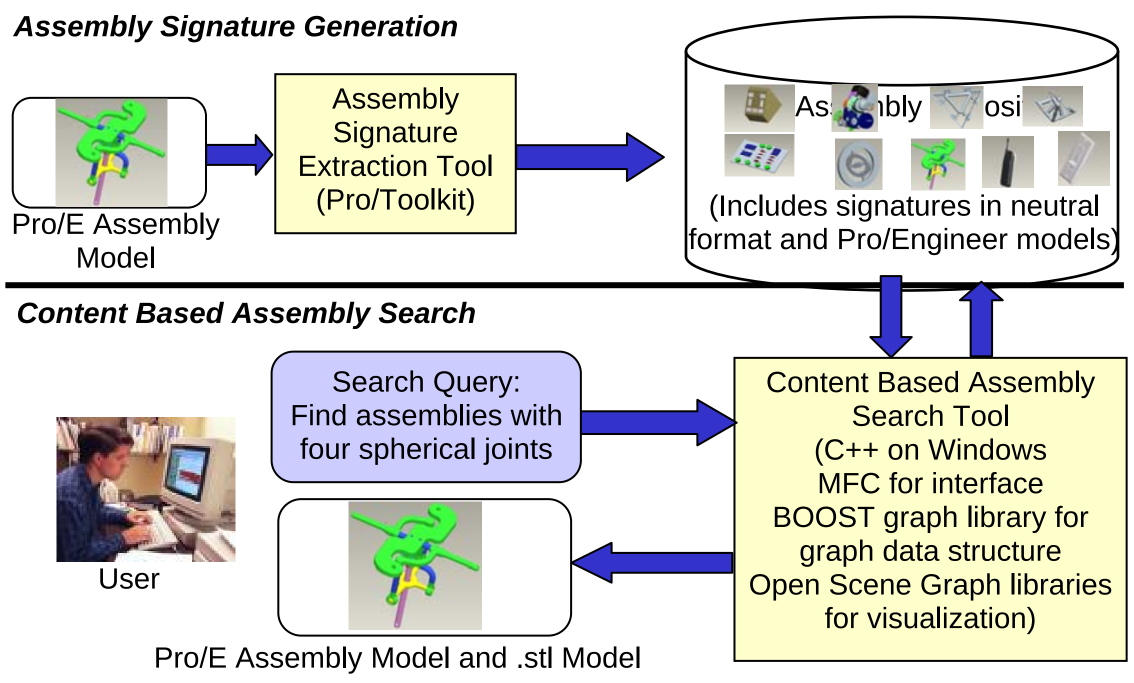
# Chapter 7

# IMPLEMENTATION

This chapter is arranged in the following manner: Section 7.1 describes the system architecture, Section 7.2 lists the libraries that have been used in the implementation, Section 7.3 describes the standards used in the assembly design, Section 7.4 describes the signature file format, Section 7.5 describes the query file format, Section 7.6 describes the process used to extract the signature of assemblies, Section 7.7 describes the output of the system and Section 7.8 describes the assembly viewer.

## 7.1 System Architecture

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 version was used as the integrated development environment (IDE) to build the software. Assembly search software is an event driven software. Figure 7.1 shows the framework of assembly search system.



Assembly Signature Generation
Assembly Signature Extraction Tool (Pro/Toolkit)
Pro/E Assembly Model
Assembly Repository
(Includes signatures in neutral format and Pro/Engineer models)
Content Based Assembly Search
Search Query: Find assemblies with four spherical joints
Content Based Assembly Search Tool (C++ on Windows MFC for interface BOOST graph library for graph data structure Open Scene Graph libraries for visualization)
User
Pro/E Assembly Model and .stl Model

Figure 7.1: The architecture of the assembly search system

The algorithm requires user to specify a top-level directory. It then iteratively searches for the signature files in the directory and all its sub-directories. A user cannot define a new query or load an existing query without specifying the search directory. A list of all assemblies in the specified folder is prepared and is used to search for possible matches. The search sequence is selected depending on the criteria specified for search. The assembly statistics are used as the first criteria. It is followed by joint relation and constituent parts. If an assembly passes all checks for these criteria, then part mating based search is performed. If any criterion is not specified, then it is omitted from the search.  The sequence of the use each criterion during search is decided on the computational time for each search criteria. Criteria are ranked based on time for computation. The criterion that takes least time is used as the first criteria and is followed by the criterion that takes comparatively higher time for search. Assembly statistics is purely a string matching based search and thus is the fastest. Joint relations based search and constituent part based search are string matching based searches which require iteration over a list. Thus they follow assembly statistics based search. An assembly is expected to have fewer numbers of joints than the constituent parts because all parts may not be part of articulated joints. Thus, joints based search will require lesser iteration and is used before constituent part based search. Part mating based search is the most time consuming of the criteria and is thus used last and is only used for assemblies that pass the check for all three previous criteria.

## 7.2 Libraries used

MFC library provides the graphical user interface (GUI) to the user to define and edit the query and select the search folder. The software stores the location of last search in registry and loads the same each time when it is started. The event handler function and variables of various components of UI provide the data that is stored in the data structures built using C++.

The data structure provided by BOOST graph library (BGL) is used to store the query graph and database graphs. The data structure is in the form of an adjacency list. The graph is in the form of an undirected graph. The parts in the assembly are represented by the nodes and mating conditions are represented by the edges. The list of nodes and list of edges is stored in a vector data structure. Each node and edge in this list can have attributes. The attributes of nodes and edges are stored in a structure.

Many inbuilt functions from BGL have been used in this implementation. The query graph required for search is a single component graph. To ensure that a single component graph is available the implementation of a function to find the number of connected components in graph from BGL has been used. The software makes use of the data structures from standard template library (STL). The list data structure from STL has been used to store the joint relations and the list of parts besides the extensive use of list and vector in part mating along with BGL.

Assembly viewer makes use of Open Scene Graph, a high level, object oriented library written over OpenGL. This library allows viewing capability of the assemblies with individual .stl files. The library has inbuilt functions to implement the zooming, panning and rotation capabilities required in a viewer. It also has inbuilt function to apply a homogenous transformation to representation of any solid body.

## 7.3 Conventions used in Assembly Design

The software provides the user an option to select characteristics of part or assembly like part material, the designer from pull down menus. The available options may change in an organization as the team of designers changes or additional material may be introduced or some material may be discarded. This standard is maintained by an organization and needs to be used in the assembly search software. The list of standards are stored in and read from a XML file. The file stores a list of assembly designers, standard parts, standards followed while drafting, the list of standard materials for parts, the type of joints available in Pro/Engineer Wildfire, the type of relations that can be defined in joints and the type of mating conditions available in Pro/Engineer Wildfire. This file can be edited without requiring recompiling of the entire application. Thus, an administrator with the proper authority can add to or delete from this list and extend the scope of search software. The values in this file are read using an XML parser and stored by the system at start up. The file is stored in the following format.

```
- <assemblysearchstringlists>
- <names>
+ <list nodetypes="assembly_owner_names">
+ <list nodetypes="assembly_standard_names">
+ <list nodetypes="standard_part_types">
+ <list nodetypes="part_material">
+ <list nodetypes="type_of_joints">
- <list nodetypes="type_of_joint_relation">
  <name ID="61">Parallel</name>
  <name ID="62">Perpendicular</name>
  <name ID="63">Angle</name>
    </list>
+ <list nodetypes="type_of_mating_condition">
    </names>
    </assemblysearchstringlists>
```

The functions to read the XML file have been borrowed from Arne Gfell. These functions are a part of his research work with Dr. Gupta.

## 7.4 Signature File Format

The search software compares the query with the signature of an assembly stored in the database. The signature of an assembly from database consists of three files. The first file has an extension sig which is an acronym for signature and contains all information about assembly statistics, and constituent part based search. In particular, the signature file contains the length, width and height of the bounding box of the assembly, the diameter of the bounding sphere of the assembly, the number of parts and fasteners in the assembly, the number of different types of joints in the assembly, the standard followed in the drafting, the name of the assembly designer and the details of the constituent parts. The details of the constituent part include the category of the part, the part name, the location where the part has been stored, the material of the part, the owner of the part, the file name, the part number assigned to the part and the mass of the part. The second file has an extension dot and is in the format used by graphviz to represent assemblies. Graphviz is graph layout software maintained by AT&T systems. The file contains a numbered list of parts in the assembly. The list of parts is followed by a list of edges in the graph. This file is read by the code and is stored in a data structure. The file contains the path of the part, the category of the part, and the mating condition between parts in the assembly. The third file has an extension jot and stores the relation between different joints in the assembly. The file contains information about all such joints in the assembly that have any relation between them.

The format of the dot file is as follows:

```
graph
{
//node
LINK_SEG2 ; // C:\Assembly_Search_demo\Database\C-clamp\link_seg2.prt
Custom inhouse
LINK_SEG1 ; // C:\Assembly_Search_demo\Database\C-clamp\link_seg1.prt
Custom inhouse
INTERCHANGE ; // C:\Assembly_Search_demo\Database\C-
clamp\interchange.prt Custom inhouse
//end node

//edge
LINK_SEG2 -- LINK_SEG1 [label= Align ];
LINK_SEG2 -- INTERCHANGE [label= Align ];
//end edge
}
```

The format of the jot file is as follows:

```
Slider -- Perpendicular -- Slider – 0
```

The format of the sig file is as follows:

```
14.190688
3.0000000
10.000000
22.950001
4 0
1
0
0
0
0
0
0
0
0
```

```
0

0

0

25

35

ISO

JohnHarris

Custom base_roller

C:\Assembly_Search_demo\Database\stewart_platform3\base_roller.prt

Aluminum JohnHarris base_roller.prt BR 3

Custom roll_joint

C:\Assembly_Search_demo\Database\stewart_platform3\roll_joint.prt

Bronze KenTucker roll_joint.prt RJ 3

Custom joint

C:\Assembly_Search_demo\Database\stewart_platform3\joint.prt Iron

JamesBond joint.prt JT 2

Custom  cap  C:\Assembly_Search_demo\Database\stewart_platform3\cap.prt

Iron JamesBond cap.prt CP 1
```

The architecture of the software has been designed to easily adapt to a change in the format of the signature. It can be adapted to read the information from a standard assembly signature format like the Open Assembly Model (OAM) proposed by the National Institute of Standards and Research (NIST).

## 7.5 Query File Format

The search software has the capability to store the query which has been defined. When a search is stored a query file (qry), a joint relations file (jot) and query graph file (dot) along with the XML representation of the graph are saved in a folder chosen by the user. The user can load the query from the directory.  A user can load an existing query from the database or define a completely new query. The results of each search are stored in a file on the computer. The capacity to store a query and store the search results can be used for iteratively refining the search. A new query can be defined to find a large set of assemblies. If the query is saved, it can be loaded from the computer and modified to search within the existing results and locate the desired assembly or set of assemblies from the database.

## 7.6 Signature Extraction

This search software system supports CAD data from Pro/Engineer Wildfire educational edition. All assemblies used in the search to create a database were created using Pro/Engineer Wildfire Education Edition. Pro/Toolkit provided with Pro/Engineer was used to extract the data from assembly files. Pro/Toolkit is C language application programming interface (API) provided with Pro/Engineer. An asynchronous program was written using this API to extract the signature of the assembly from the database. The program can extract information about the parts, their names, mating conditions between parts, the tree structure of the assembly, the material of a part and the mass properties of the part and the assembly. The mass properties include the mass and density of the parts and bounding sphere diameter of the assembly. The parameters of the bounding box were extracted using Open Scene Graph which provides a function to calculate these parameters. The assembly viewer requires the assembly signature in form of an XML file and the individual part files in form of .stl files. The Pro/Toolkit program written for this application exports the XML file and .stl files in the format required by the assembly viewer. The .stl file size needs to be adjusted so that the details in small features of the part are not lost and the size is not very large for a computer with a limited memory to read and display. In order to ensure this, experiments were conducted with the size of triangles in a .stl file based on the bounding sphere diameter of the part. It was visually observed that a chord length of size equal to a ratio of bounding sphere diameter to 1000 gives the best resolution and speed for loading the parts in the assembly viewer. The program in Pro/Toolkit is written accordingly to export the .stl files. If the size of the .stl file expands beyond 5MB, the file is re-exported with larger chord length. In all cases the angle value for exporting the .stl file is maintained at 1 degree to ensure that finer details of the part are visible in the .stl file. The program also exports relative positions of all parts in the assembly with respect to a single coordinate axis. The relative position is in the form of homogenous coordinates. The assembly viewer software applies the transformation to individual .stl files and shows each part in its final position in the assembly.

We could not function to extract information about joints and the relations between joints in an assembly in Pro/Toolkit. This information has been extracted manually. The information about the number of different types of joints is recorded in the signature file and the type of relation between different joints in the assembly is recorded in the jot file. In addition, the following information is appended manually to the signature of the assembly: standards followed in drafting, the name of designers for the constituent part and assembly, the number of fasteners in the assembly, the names of conformance standards used in the assembly design, the rotationally symmetric nature of the part and the percentage of sheet metal parts in the assembly.

The API program requires the user to select a folder where the Pro/Engineer assemblies are stored. In 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 software does not require any CAD software as it searches only based on the assembly signature. Since a separate viewer is provided to browse the assemblies and the results of search, this software is not dependent on any single CAD system.

The data stored in the signature of the assembly can be extracted from any CAD software. Most 3D CAD software provide an API using which a program can be written to extract data from the CAD files. Examples of other such API programs are UG/Open API for Unigraphics and CATIA CAA for CATIA. Once such data is extracted from CAD, the assembly search software can be used to search assemblies. Thus this software can be used for searching assembly designed in any CAD system. It can also be used to search in a database that contains assemblies designed using multiple CAD software.

## 7.7 System Output

The software displays the results as thumbnails in a pop up window. The window shows the jpeg image of the assemblies that matches with the query. The user can visually browse the results in the window. The user can see the name and location of each matching assembly file which is printed below each image. The figure 7.2 shows the output of the system. This window does not provide the user with any additional viewing capabilities like zooming in or out or rotate, panning. It also does not provide the user access to the hierarchical tree structure of the assembly. Additional software has been provided that has all the capabilities and allows the user to browse the results or the assemblies in the database. The user can double click on an image of any matching assembly shown in the pop up window to launch the assembly viewer. Double click on separate images results in multiple sessions of the assembly viewer.

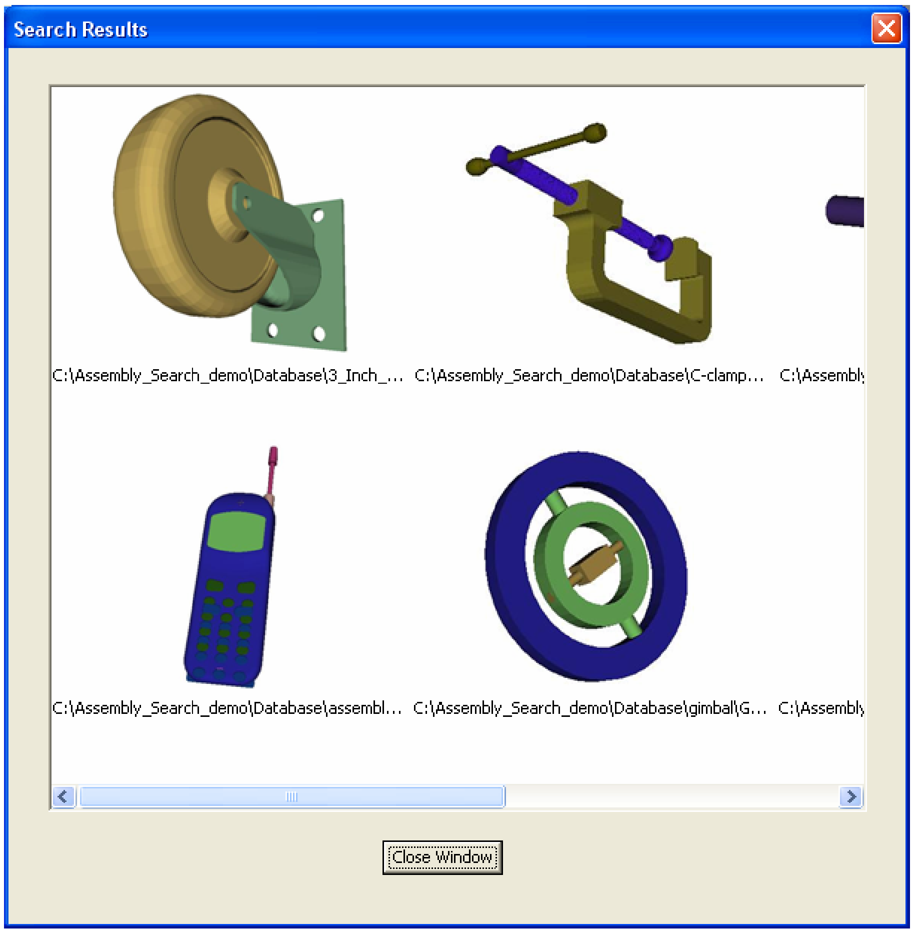

Search Results
C:\Assembly_Search_demo\Database\3_Inch_...
C:\Assembly_Search_demo\Database\C-clamp...
C:\Assembly
C:\Assembly_Search_demo\Database\assembl...
C:\Assembly_Search_demo\Database\gimbal\G...
C:\Assembly
Close Window

Figure 7.2: Output window of the assembly search software

## 7.8 Assembly Viewer

The results are displayed using the Assembly Viewer software as shown in the figure 7.3. Assembly Viewer provides an OpenGL based representation of the assembly files. To implement OpenGL based representation, Assembly Viewer uses Open Scene Graph library which is a higher level library built on top of OpenGL. The viewer has the capability to hide parts in the assembly. It has rotate and pan capabilities of a 3D graphics viewer. The viewer shows the tree structure of the assembly. The selection of a part for hiding can be done either on the graphics screen or by clicking on a part in the assembly tree structure. This viewer is used with assembly search as a tool to view the matching assemblies in details. It can also be used an independent tool to view the assemblies in the database instead of viewing them in the CAD system.

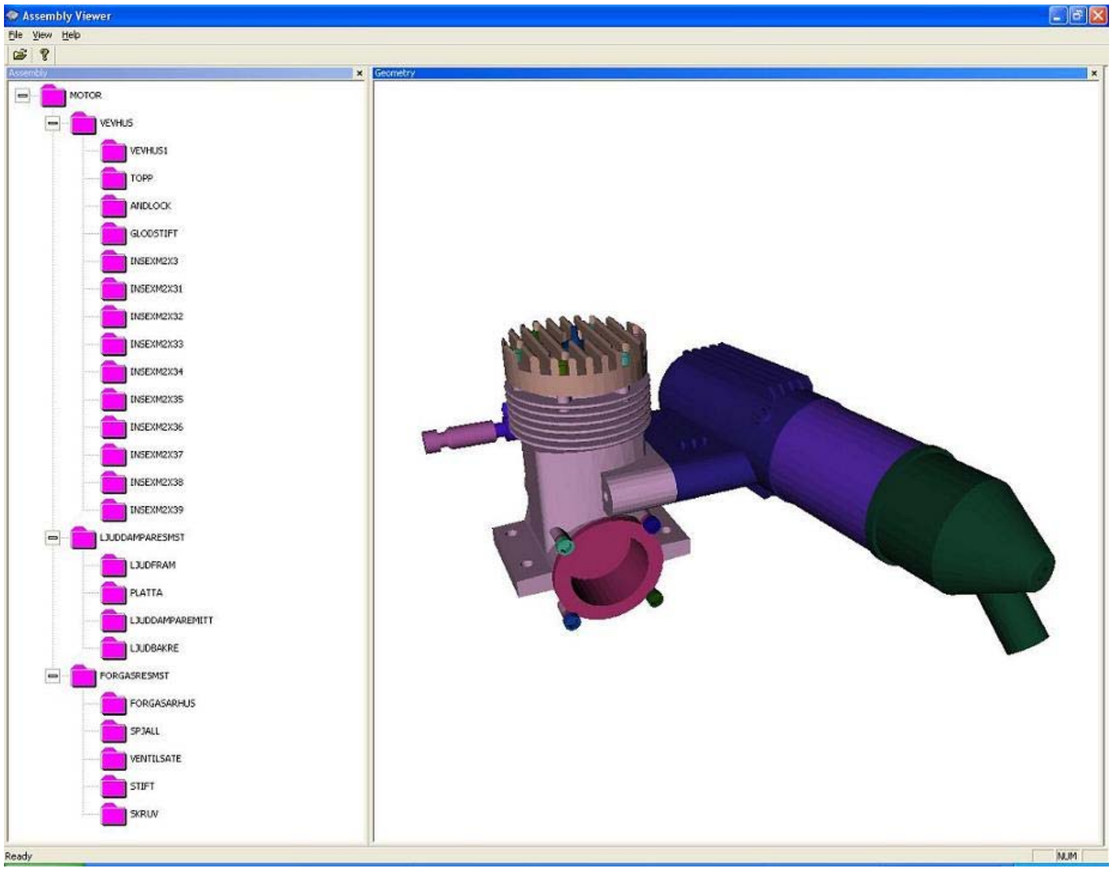


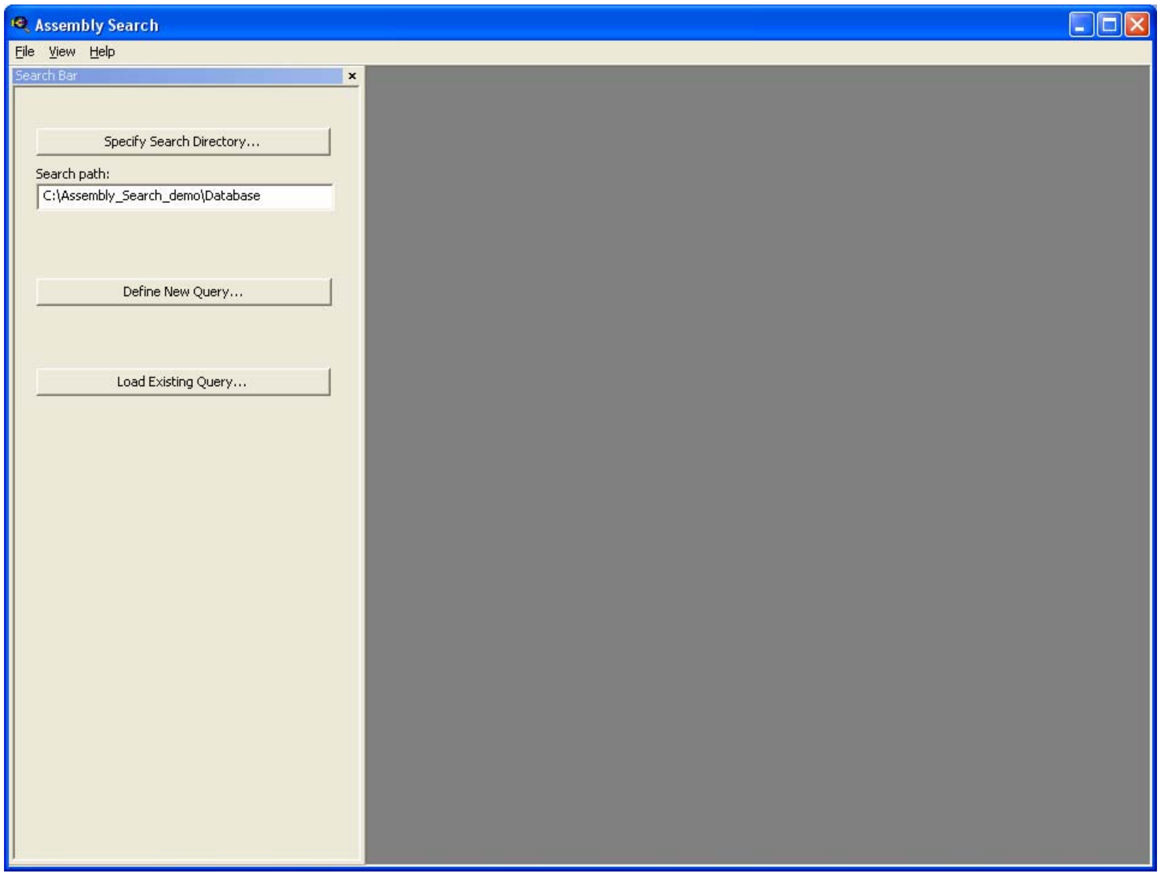
Figure 7.3: Assembly viewer with tree structure on the left and geometry view on the right

## 7.9 Statistics of assembly database

The database consists of 50 assemblies. The numbers of parts vary between 2 and 107. The largest assembly has a bounding sphere radius equal to 169 inches and bounding box length equal to 91, width equal to 115 and height equal to 113 inches. The highest numbers of joints in any assembly are 6. The smallest assembly has a bounding sphere radius of 0.5 inches and bounding box length of 0.3 inches, width of 0.4 inches and height of 0.3 inches.

## 7.10 Interface to Define Search

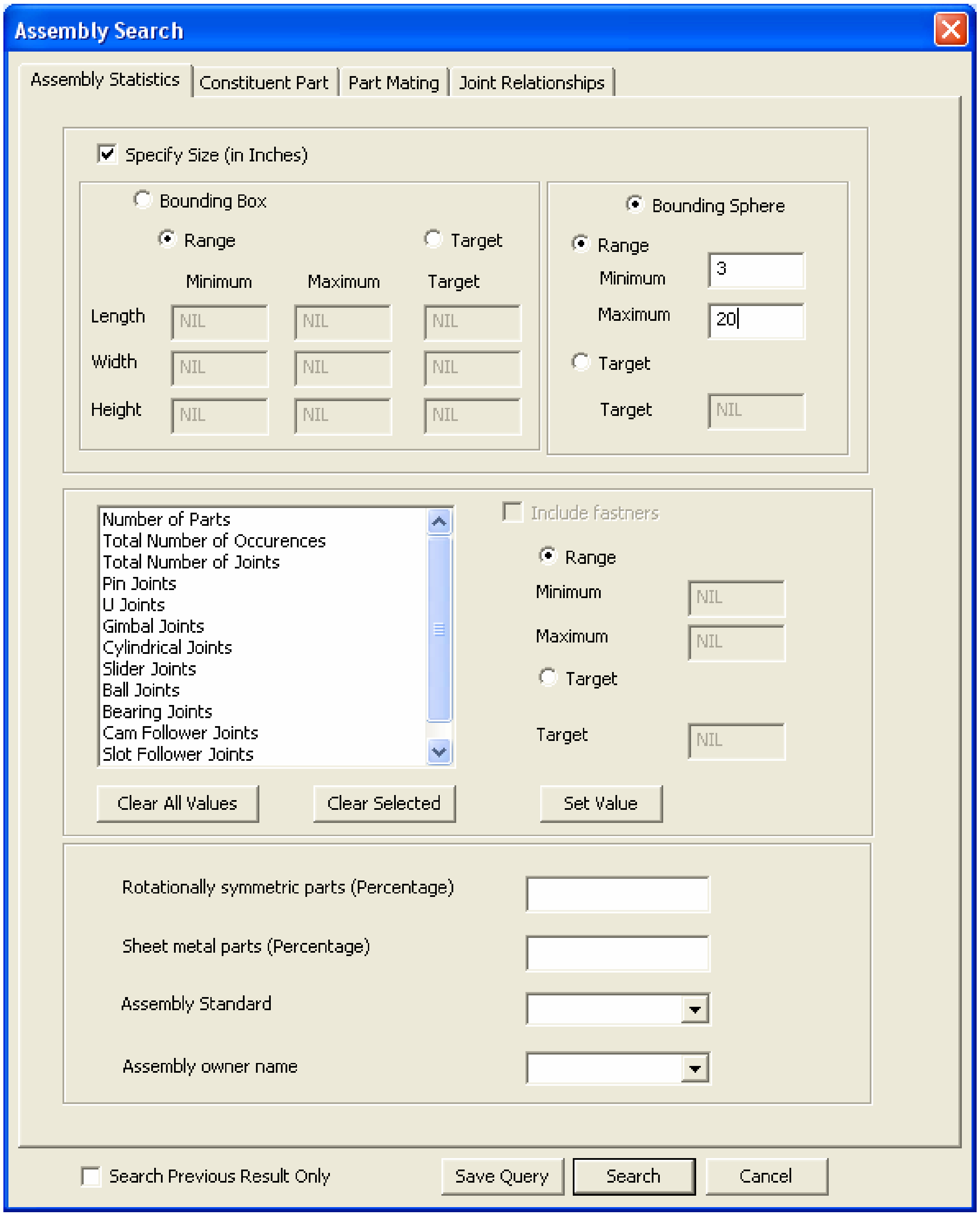
Figure 7.4 shows the main window to define search path.



Search path: C:\Assembly_Search_demo\Database

Figure 7.4: Main window to define search path and define queries

Figure 7.5 shows the interface to define search based on assembly statistics.


Assembly Search

Assembly Statistics | Constituent Part | Part Mating | Joint Relationships

[✓] Specify Size (in Inches)

( ) Bounding Box
[● Range] ( ) Target

| | Minimum | Maximum | Target |
|---|---|---|---|
| Length | NIL | NIL | NIL |
| Width | NIL | NIL | NIL |
| Height | NIL | NIL | NIL |

[● Bounding Sphere]
[● Range]
Minimum: 3
Maximum: 20
( ) Target
Target: NIL

Number of Parts
Total Number of Occurences
Total Number of Joints
Pin Joints
U Joints
Gimbal Joints
Cylindrical Joints
Slider Joints
Ball Joints
Bearing Joints
Cam Follower Joints
Slot Follower Joints

[ ] Include fastners
[● Range]
Minimum: NIL
Maximum: NIL
( ) Target
Target: NIL

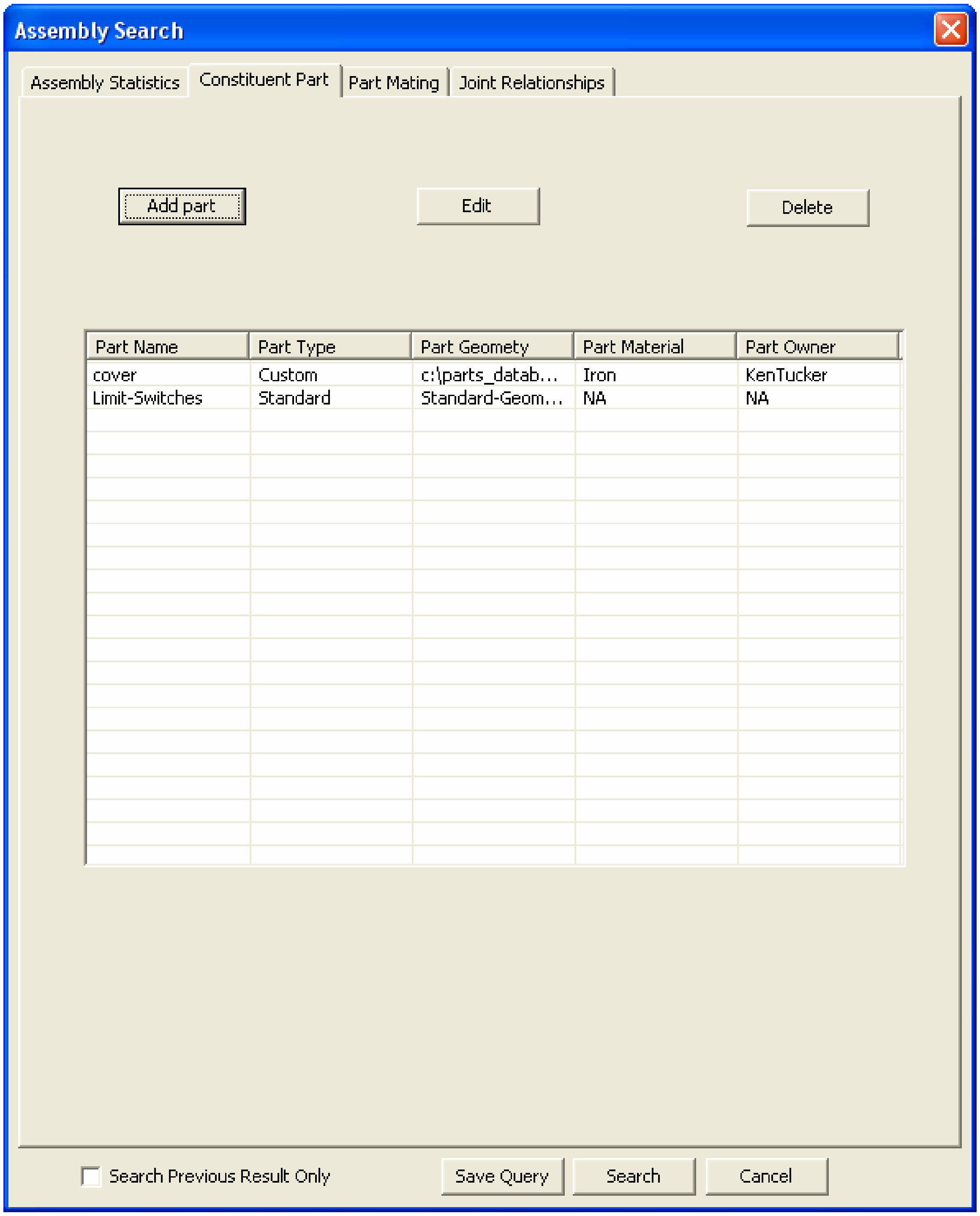
Clear All Values | Clear Selected | Set Value

Rotationally symmetric parts (Percentage)
Sheet metal parts (Percentage)
Assembly Standard
Assembly owner name

[ ] Search Previous Result Only
Save Query | Search | Cancel

Figure 7.5: Interface to define search criteria based on assembly statistics

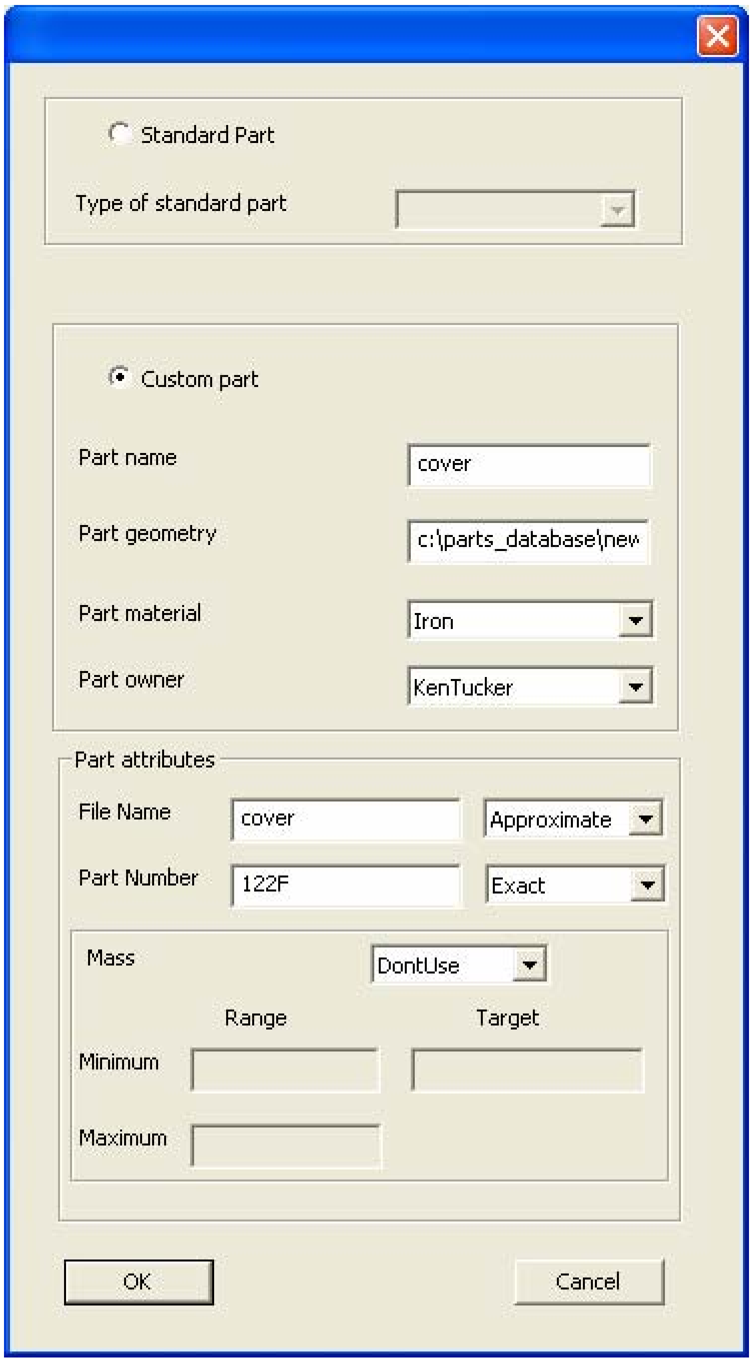
Figure 7.6 shows the interface to define search based on constituent parts.



| Part Name | Part Type | Part Geometry | Part Material | Part Owner |
|---|---|---|---|---|
| cover | Custom | c:\parts_datab... | Iron | KenTucker |
| Limit-Switches | Standard | Standard-Geom... | NA | NA |

Figure 7.6: Interface to define constituent part based search

Figure 7.7 shows the interface to define a constituent part.



( ) Standard Part

Type of standard part

(●) Custom part

Part name: cover

Part geometry: c:\parts_database\new

Part material: Iron

Part owner: KenTucker

Part attributes

File Name: cover — Approximate

Part Number: 122F — Exact

Mass: DontUse

Figure 7.7: Dialog to define a constituent part

Figure 7.8 shows the interface to define search based on mating conditions between parts.

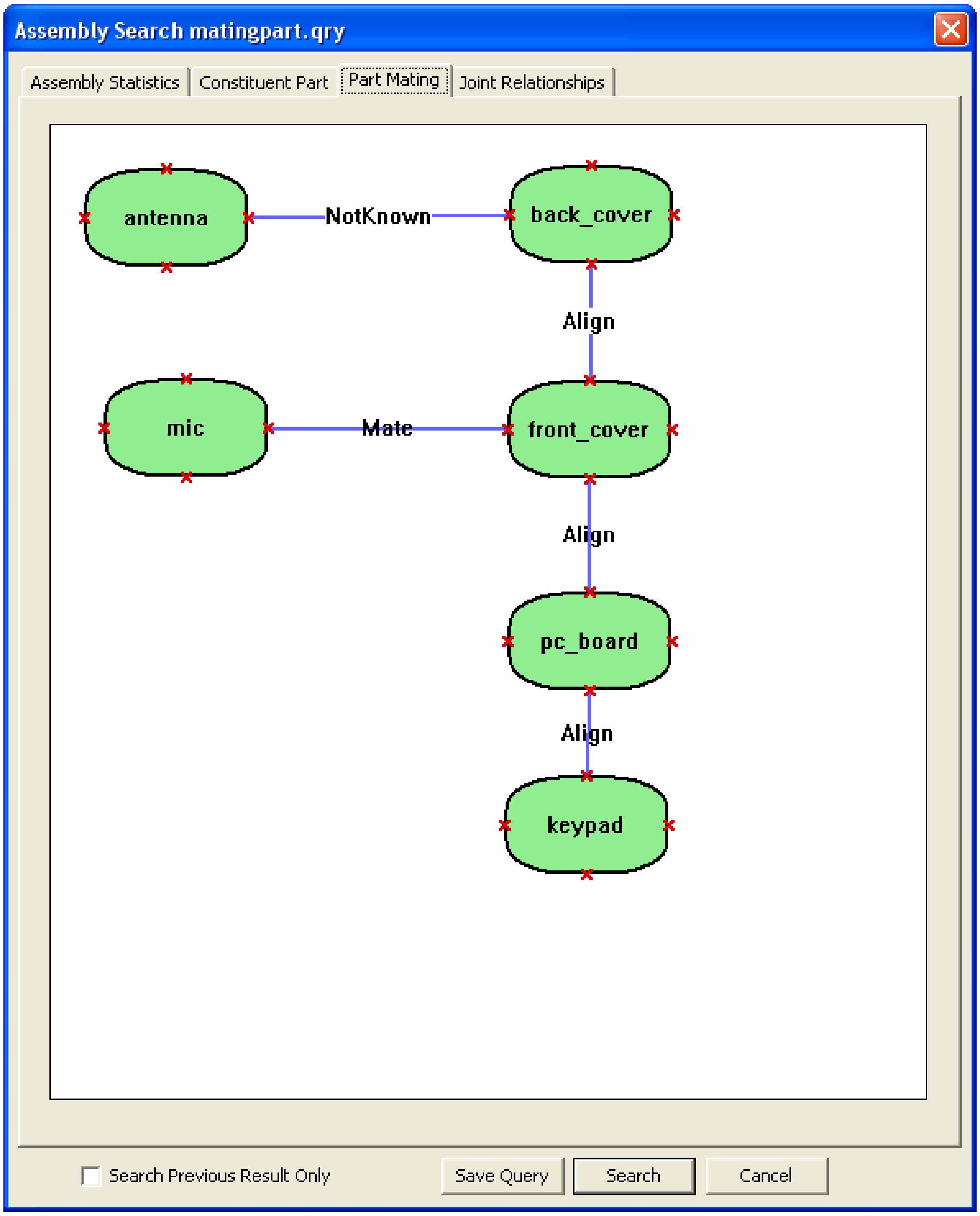
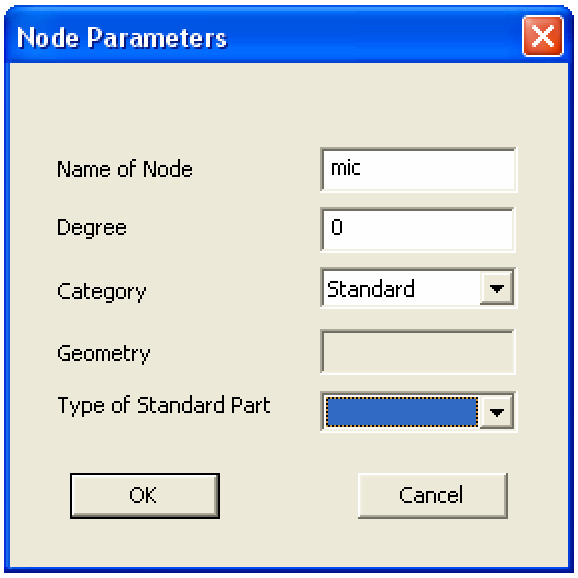


Figure 7.8: Interface to define query graph for part mating conditions based search

Figure 7.9 shows the interface to add a node to the query graph.


Node Parameters

Name of Node: mic

Degree: 0

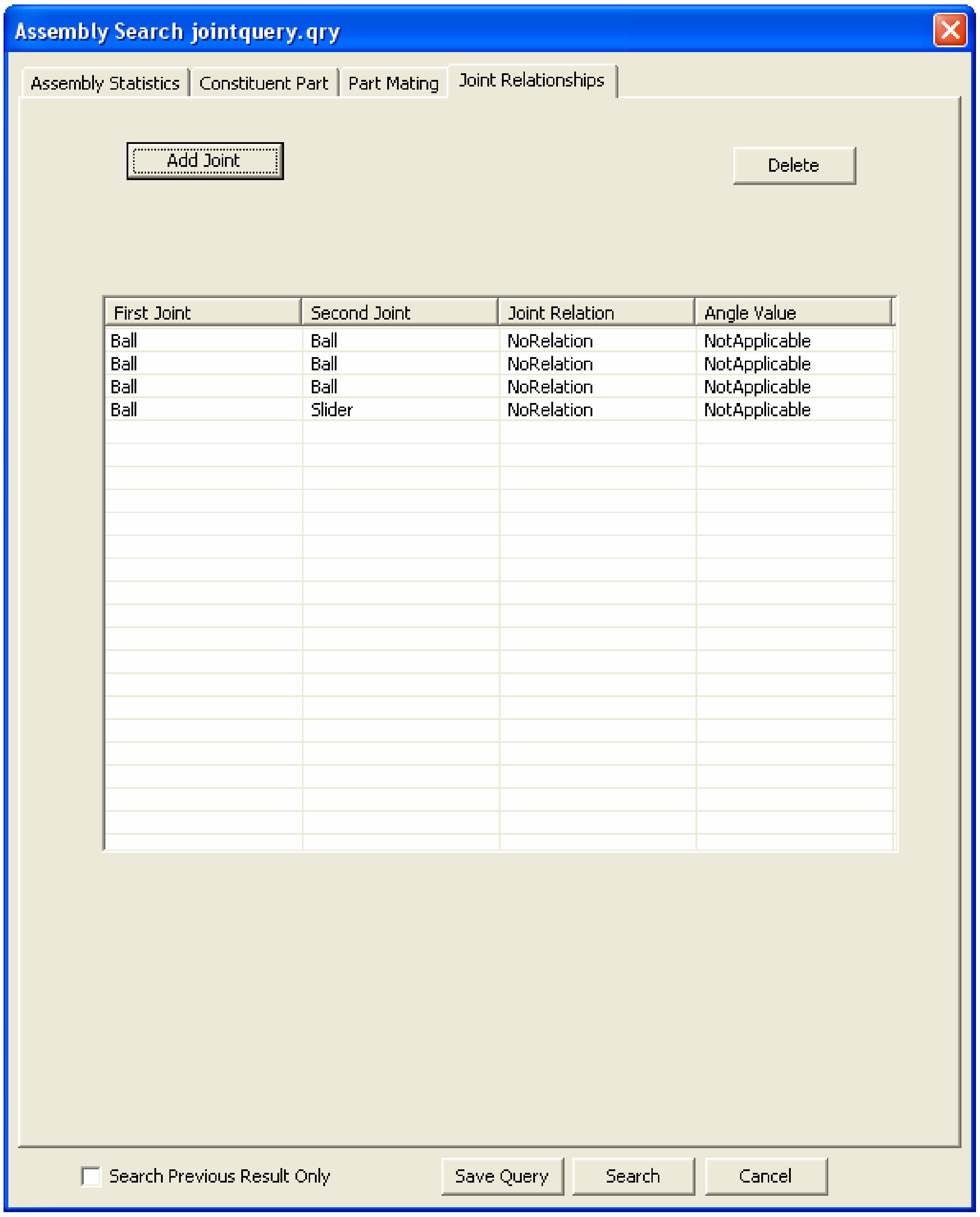
Category: Standard

Geometry:

Type of Standard Part:

Figure 7.9: Dialog to define node in the query graph for search based on part mating conditions

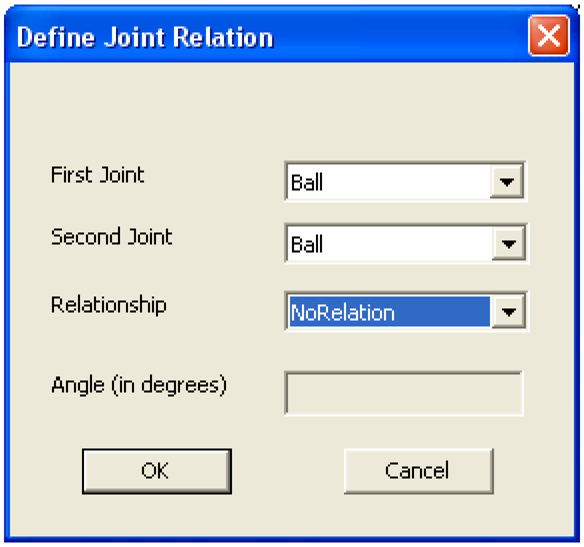
Figure 7.10 shows the interface to define search based on joint relationship.



| First Joint | Second Joint | Joint Relation | Angle Value |
|---|---|---|---|
| Ball | Ball | NoRelation | NotApplicable |
| Ball | Ball | NoRelation | NotApplicable |
| Ball | Ball | NoRelation | NotApplicable |
| Ball | Slider | NoRelation | NotApplicable |

Figure 7.10: Interface to define search based on joint relationships

Figure 7.11 shows the interface to add a joint relationship to the query.


Define Joint Relation

First Joint: Ball

Second Joint: Ball

Relationship: NoRelation

Angle (in degrees):

Figure 7.11: Dialog to define a joint relationship

**Chapter 8**

# CONCLUSION AND FUTURE WORK

This chapter is arranged in the following manner: Section 8.1 describes the contributions of this research, Section 8.2 lists the anticipated benefits from this research and Section 8.3 describes the future work in this field of research.

## 8.1 Research Contributions

**Content based assembly search:** This research work describes a comprehensive framework for performing content-based search for mechanical assemblies. The search definition templates support a wide variety of search queries that can be posed to the system. Search definition templates described in this thesis spans nearly all aspects of the assembly model. Hence, it provides a designer with a very expressive search definition capability. A variety of search options are provided to allow users to define from very narrow to very broad searches based on their needs.

**Iterative search with refined criteria for faster computations:** Searches can be iteratively refined to better direct the search. The search has been categorized to enable the user to define search criteria that can be used to search for multiple assemblies over a large database. One can first use computationally faster search criterion to narrow down the search and then subsequently use computationally expensive search criterion. This search criterion can then be used to search only on the results obtained in the previous search results. The number of assemblies for the computationally expensive search criteria can thus be reduced to get results in lesser time as compared to applying the computationally expensive criteria over the entire database of assemblies. Search results can be browsed through a combination of images and a convenient visual interface of the assembly viewer.

**Pruning search space for lesser computation:** Conservative pruning used by search algorithms ensures efficient search performance and at the same time does not exclude results that might be of interest to the user. In case of search criteria defined for multiple criteria, the system ranks the search criteria. The ranking gives highest priority to a search criteria that will reduce the search space. Thus, progressively the search space is reduced and the more computationally expensive search criteria are used for less number of assemblies from the specified database.

**Solution of the graph compatibility problem:** A problem similar to subgraph isomorphism, termed in this thesis as graph compatibility, is solved for nodes and edges with attributes. This research work deals with partially specified query graph and uses depth first search combined with six pruning criteria to determine compatibility of the query graph with graphs in the database. We have shown that this approach works well with small query graph.

## 8.2 Anticipated Benefits

**Reduced design time with better searching:** We expect that the system described in this thesis will serve three purposes. First, it will allow designers to reuse existing assemblies by giving them a means to identify assemblies with the desired characteristics. A large part of designer's time is lost in redesigning solutions for similar problems or for searching the database for earlier design attempts. This research is a further step on geometry based search and text based search as it allows content based search. The search time for assemblies can be reduced using this system. This system allows defining a search based on all characteristics of assemblies defined in a CAD system.

**Access to design knowledge in legacy designs:** It will provide designers an access to the DFMA knowledge contained in the assembly database, and hence transfer best practices to new designs. This capability can also be used by new designers to learn the design principles followed in the organization. These capabilities will eventually lead to further cutting down the design time required for assemblies.

**CAD independent cost effective search:** This research also attempts to separate search functionality and the proprietary CAD data. Thus, this system can be used by organizations that use multiple CAD systems. In addition, the capability to search assemblies independent of any CAD leaves the more costly CAD software free for design work.

## 8.3 Future Work

**CAD API programs for extracting assembly signature:** The current implementation works only with the assembly characteristics available in Pro/Engineer CAD system to build the signature of the assembly. The search tool can be extended to search assemblies from other CAD systems. This would require the API programs for each CAD system to extract the signature of the assembly from the CAD software.

**Inclusion of function based search:** An assembly can be defined as a collection of parts to fulfill a function. The function of the assembly is thus the primary characteristic of an assembly. However, a function of an assembly is not always explicitly stored in CAD files. Often, the function of an assembly cannot be inferred from its geometric characteristics. Hence, the designer cannot search for a design fulfilling a particular function. Further research needs to be done to extend this work to support queries based on functions.

**Extension of search for assemblies beyond mechanical domain:** The proposed search method works only on the basis of the form related characteristics of the assembly. This search is thus not applicable to assemblies that have characteristics from other domains of engineering like electromechanical or electrochemical engineering. Further research needs to done identify characteristics from these domains and algorithms to search based on these characteristics need to be identified.

**Ensuring availability of data for creation of assembly signature:** The search also works on the assumption that the designer explicitly defines joints and mating conditions or such relations can be implicitly 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. Further research needs to be done to infer the data about mating conditions and joints in an assembly where the designer has not specified it.

# BIBLIOGRAPHY

[Anan96] R. Anantha, G.A. Kramer, and R.H. Crawford. Assembly modelling by geometric constraint satisfaction. *Computer Aided Design,* Vol. 28, No. 9, pp. 707-722, 1996.

[Besp03] D. Bespalov, W.C. Regli, and A. Shokoufandeh. Reeb Graph Based Shape Retrieval for CAD. In Proceedings of *23rd ASME DETC Computers And Information In Engineering (CIE) Conference*, Chicago, Illinois, 2003.

[Bohm05] M.R. Bohm, J.P. Vucovich, and R.B. Stone. Capturing Creativity: Using a Design Repository to Drive Concept Innovation. Proceedings of *ASME Design Engineering Technical Conferences*, Long Beach, California, September 2005.

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

[Boot94] G. Boothroyd. Product design for manufacture and assembly. *Computer Aided Design*, Vol. 26, No. 9, pp. 505-520, 1994.

[Brun00] G. Brunetti, and B. Golob. A feature-based approach towards an integrated product model including conceptual design information. *Computer Aided Design*, Vol. 32, No. 14, pp. 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*, Vol. 3, No. 2, pp. 109-118, 2003.

[Card04] A. Cardone, S.K. Gupta, and M.V. Karnik. Identifying similar parts for assisting cost estimation of prismatic machined parts. In Proceedings of the *ASME Design for Manufacturing Conference*, Salt Lake City, Utah, 2004.

[Card05] A. Cardone, *A Feature-Based Shape Similarity Assessment Framework.* Ph.D. Thesis, University of Maryland, College Park, Maryland 2005.

[Card06] A. Cardone, and S. K. Gupta. Shape Similarity Assessment Based on Face Alignment using Attributed Applied Vectors. *CAD Conference*, Phuket Island, Thailand, June 2006.

[Chak05a] T. Chakraborty, S. Venkataraman, and M. Sohoni. A fast 3D Shape Search Technique For 3D Cax/PDM Repositories. Technical Paper, *Society Of Manufacturing Engineers*, August 2005.

[Chak05b] T. Chakraborty. Shape-Based Clustering Of Enterprise CAD Databases. *Computer Aided Design and Applications*, Vol. 2, No. 1-4, pp. 145-154, 2005.

[Cici00] V. Cicirello, and W.C. Regli. Managing Digital Libraries for Computer-Aided Design. *Computer Aided Design*, Vol. 32, No. 2, pp. 119-132, 2000.

[Cici01] V. Cicirello, and W.C. Regli. Machining feature-based comparisons of mechanical parts. In Proceedings of the *International Conference on Shape Modeling & Applications*, 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*, Vol. 26, No. 10, October 2004.

[Corn03] J. Corney, H. Rea, D. Clark, J. Pritchard, R. MacLeod, and M. Breaks. Coarse Filters for Shape Matching. *IEEE Computer Graphics and Applications*, Vol. 22, No. 3, pp. 65-74, 2003.

[DeFa87] T. De Fazio, and D. Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, Vol. 3, No. 6, pp. 640-658, 1987.

[Desh05] A.S. Deshmukh, S.K. Gupta, M.V. Karnik, and R. Sriram. A system for performing content-based searches on a database of mechanical assemblies. In *ASME International Mechanical Engineering Congress & Exposition*, Orlando, Florida, November 2005.

[ElMe03] M. El-Mehalawi, and R. A. Miller. A Database System Of Mechanical Components Based On Geometric And Topological Similarity, Part II: Indexing, Retrieval, Matching, And Similarity Assessment. *Computer-Aided Design*, Vol. 35, No. 1, pp. 95-105, 2003.

[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*, Vol. 1876, pp. 427-436, 2000.

[Funk03] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A Search Engine For 3D Models. *ACM Transactions on Graphics*, Vol. 22, No. 1, pp. 83-105, 2003.

[Gupt01] S.K. Gupta, C.J. Paredis, R. Sinha, and P.F. Brown. Intelligent assembly modeling and simulation. *Assembly Automation*, Vol. 21, No. 3, pp. 215-235, 2001.

[Gupt06] S.K. Gupta, A. Cardone, and A. Deshmukh. Content-Based Search Techniques for Searching CAD Databases. *CAD Conference*, Phuket Island, Thailand, June 2006.

[Hila01] M. Hilaga, Y. Shinagawa, T. Kohmura, and T.L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In Proceedings of the *28th Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, California, August 2001.

[Home91] L.S. Homem de Mello, and A.C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 2, pp. 228-240, 1991.

[Iyer04] N. Iyer, S. Jayanti, K. Lou, Y. Kalyanaraman, and K. Ramani. A Multi-Scale Hierarchical 3D Shape Representation For Similar Shape Retrieval. In *Proceedings Of Tools and Methods for Competitive Engineering Conference*, Lausanne, Switzerland, April 2004.

[Karn05a] M.V. Karnik, S.K. Gupta, and E.B. Magrab Geometric Algorithms for Containment Analysis of Rotational Parts. *Computer Aided Design*, Vol. 37, No. 2, pp. 213-230, 2005.

[Karn05b] M.V. Karnik, D.K. Anand, E. Eick, S.K. Gupta, and R. Kavetsky. Integrated visual and geometric search tools for locating desired parts in a part database. *CAD Conference*, Bangkok, Thailand, June 2005.

[Karn05c] M.V. Karnik, S.K. Gupta, D.K. Anand, F.J. Valenta, and I.A. Wexler. Design Navigator system: A case study in improving product development through improved information management. In *ASME Computers and Information in Engineering Conference*, Long Beach, California, September 2005.

[Khos89] P Khosla, and R Mattikalli. Determining the assembly sequence from a 3D model. *Journal of Mechanical Working Technology*, Vol. 20, pp. 153-162, 1989.

[Ko03] K.H. Ko, T. Maekawa, and N.M. Patrikalakis. An Algorithm for Optimal Free-Form Object Matching. *Computer Aided Design*, Vol. 35, No. 10, pp. 913-923, 2003.

[Ko05] K.H. Ko, T. Maekawa, and N.M. Patrikalakis. Algorithms for Optimal Partial Matching Of Free-Form Objects With Scaling Effects. *Graphical Models*, Vol. 67, No. 2, pp. 120-148, 2005.

[Kope05] J.B. Kopena, C.D. Cera, and W.C. Regli. Conceptual Design Knowledge Management and the Semantic Web. *ASME 2005 International Design*

*Engineering Technical Conference and Computers and Information in Engineering Conference*, Long Beach, California, September, 2005.

[Lamo06] M. Lamont, http://linux.wku.edu/~lamonml/index.html

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

[Lee93] S Lee, G. Kim, and G. Bekey. Combining assembly planning with redesign: An approach for more effective DFA. In Proceedings of the *IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993.

[Leve66] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Phys. Dokl.*, Vol. 6, pp. 707-701, 1966.

[Li04] Z. Li, M. Liu, and K. Ramani. Review of product information retrieval: representation & indexing. In Proceedings of the *ASME Design Engineering Technical Conferences*, Salt Lake City, Utah, 2004.

[Lou04] K. Lou, S. Prabhakar, and K. Ramani. Content Based Three Dimensional Engineering Shape Search. In *Proceedings Of 20th International Conference On Data Engineering*, Boston, Massachusetts, 2004.

[Merr06] http://www.merriampark.com/ld.htm

[Mess98] B.T. Messmer, and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition* Vol. 32, Issue 12, pp. 1979-1998, 1999.

[McWh01a] D. McWherter, M. Peabody, A. Shokoufandeh, and W.C. Regli. Transformation invariant similarity assessment of solid models. In

Proceedings of the *ASME Design Engineering Technical Conference*, Pittsburgh, Pennsylvania, 2001.

[McWh01b] D. McWherter, M. Peabody, A. Shokoufandeh, and W.C. Regli. Solid Model Databases: Techniques And Empirical Results. *Journal Of Computer And Information Science In Engineering*, Vol. 1, No. 4, pp. 300-310, 2001.

[Min02] P. Min, J. Chen, and T. Funkhouser. A 2D Sketch Interface for a 3D Model Search Engine. *SIGGRAPH 2002 Technical Sketches*, San Antonio, Texas, July 2002.

[Moll93] E. Molloy, H. Yang, and J. Browne. Feature-based modelling in design for assembly. *International Journal of Computer Integrated Manufacturing*, Vol. 6, No. 1-2, pp. 119-125, 1993.

[Nand05] J. Nanda, T.W. Simpson, S.B. Shooter and R.B. Stone. A Unified Information Model for Product Family Design Management. *ASME 2005 International Design Engineering Technical Conference and Computers and Information in Engineering Conference*, Long Beach, California, September 2005.

[Noor02] A. Noort, G.F.M. Hoek, and W.F. Bronsvoort. Integrating part and assembly modeling. *Computer-Aided Design*, Vol. 34, No. 12, pp. 899 - 912, 2002.

[Nore06] http://www.norecs.com/images/flangeT4_web.jpg accessed on November 17, 2006

[Osad01] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3D models with shape distributions. In Proceedings of the *International Conference on Shape Modeling and Applications*, Genova, Italy, 2001.

[Osad02] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape Distributions. *ACM Transactions On Graphics*, Vol. 21, No. 4, pp. 807-832, 2002.

[Pu05] J. T. Pu, and K. Ramani. A 2D Sketch Based User Interface for 3D CAD Model Retrieval. *Journal of Computer Aided Design and Application*, Vol. 2, pp. 717-727, 2005.

[Proe04] Pro/Engineer Wildfire documentation, 2004.

[Pros06] http://www.prosthetics-orthotics.net/Hip7.JPG accessed on November 17, 2006

[Rach06] S. Rachuri, Y.H. Han, S. Foufou, S.C. Feng, U, Roy, F. Wang. R.D. Sriram, and K.W. Lyons. A Model for Capturing Product Assembly Information. *Journal of Computing and Information Science in Engineering*. Vol. 6, pp. 11-21, 2006.

[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*, Vol. 1, No. 3, pp. 245-256, 2001.

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

[Saak04] A. Saaksvuori and A. Immonen. *Product Lifecycle Management*. Springer, 2000.

[Shaf05] J. Shaffer, J.B. Kopena, and W.C. Regli. Web Service Interfaces for Design Repositories. *ASME 2005 International Design Engineering Technical Conference and Computers and Information in Engineering Conference*, Long Beach, California, September, 2005.

[Shah93] J.J. Shah, and M.T. Rogers. Assembly modeling as an extension of feature-based design. *Research in Engineering Design*, Vol. 5, No. 3-4, pp. 218-237, 1993.

[Sung02] R. Sung, H.J. Rea, J.R. Corney, D.E.R. Clark, J. Pritchard, M.L. Breaks, and R.A. MacLeod. Assessing the effectiveness of filters for shape matching. In Proceedings of the *ASME International Mechanical Engineering Congress & Exposition*, New Orleans, Louisiana, 2002.

[Ullm76] J.R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the Association for Computing Machinery*, Vol. 23, No. I, pp. 31-42, 1976.

[Wang90] N. Wang, and T.G. Ozsoy. Representation of Assemblies for Automatic Tolerance Chain Generation. *Engineering with Computers*. Vol. 6, Number 2, pp. 121-126, 1990.

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