Computer-Aided Design 44 (2012) 554–574

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

A flexible assembly retrieval approach for model reuse

Xiang Chen [a], Shuming Gao [a,*], Song Guo [a], Jing Bai [a,b]

[a] State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou, PR China

[b] Dept. of Computer Science and Engineering, Beifang University of Nationalities, Yinchuan, PR China

ARTICLE INFO

Article history:
Received 12 June 2011
Accepted 8 February 2012

Keywords:
Flexible assembly retrieval
Multilevel assembly descriptor
Partial retrieval
Assembling semantics
Assembly model matching
Assembly indexing mechanism
Design reuse

ABSTRACT

Nowadays, growing quantities of product models are created in industries. Usually, these models contain abundant design knowledge, either explicit or implicit, in various disciplines. As an approach to taking full advantage of the design knowledge embedded, model reuse plays an increasingly important part in complex product design and innovative design, in which enormous time and cost can be saved. While model retrieval is a natural and promising way to help designers find the right models for quick and accurate reuse, the retrieval technology for assemblies is yet to reach maturity since the previous text-based or low-level content-based assembly retrieval could not fully support the needs of users.

In this paper, a new assembly retrieval approach is presented, based on which, users can input flexible queries, either rough or precise, to retrieve efficiently the whole or partial assemblies they want from the product library. First, a multilevel assembly descriptor supporting various searching requirements is elaborated, which collects different levels of information in assembly models. Then, the corresponding matching and similarity assessment methods with well-balanced efficiency and discriminability are given to evaluate the differences between assembly models. Moreover, an indexing mechanism for accelerating assembly retrieval, especially the partial retrieval, is presented to filter the unmatchable models quickly. Finally, an assembly retrieval prototype system is implemented, and the experimental results are analyzed to verify the advantages of the flexible assembly retrieval approach.

1. Introduction

In the information age today, the amount and value of information is increasing rapidly due to the widespread use of information technology. Specifically, as the computer-aided design (CAD) systems enjoy ever greater popularity in modern industries, a vast number of 3D digital models are generated and stored in the internet or enterprise repositories. These models always contain plenty of embedded knowledge worthy of utilization. Therefore, it will be a huge waste if the models, together with the embedded knowledge, could not be discovered and exploited to help practical works. Model retrieval technology, capable of searching out the models similar or relevant to a user input query from a huge library, is apparently conductive to the reuse of the abundant models and knowledge available worldwide.

In fact, model retrieval can be a dominant technology bringing about remarkable changes in the world. Various daily works can benefit from the model retrieval technology, such as design, study, exhibition and consultation. For example, designers always need to find models reusable for their design tasks at hand. The functional decomposition, geometrical implementation or even manufacturing cost estimation of a new design can all be accelerated by studying or reusing the design intents from similar models. Sometimes, when designers have just a rough idea in their minds, they can even provide a rough query model and use model retrieval to find some relevant models to inspire their designs. Since design is crucial to the success of a product in the market, modern industries can achieve huge gains with the increase of design efficiency granted by model retrieval. As Llewelyn [1] point out, the design work takes up only about 15%–20% of the total cost, but the quality of the design can decide 70%–80% of the total cost. Meanwhile, in statistics, model and knowledge reuse takes a considerable part in design activities. According to Gunn [2], only 20% of parts require completely new designs, while 40% of them are obtained by direct reuse and the other 40% through a modification of the existing designs. Therefore, it is urgent for modern enterprises and companies to possess model retrieval technologies which fulfill designers' various requirements. Well developed and utilized model retrieval systems can enormously help enterprises and companies shorten the product lifecycles, which ultimately determines their successes in the global market.

Given that manual browsing is tedious and error-prone, there have emerged several kinds of tools for search so far, e.g. text-based search. However, as mentioned in [3], the text-based search has its limitations and is not always a good way to search engineering

* Corresponding author. Tel.: +86 571 88206681x514.
E-mail addresses: xchen@cad.zju.edu.cn (X. Chen), smgao@cad.zju.edu.cn (S. Gao), guosong@cad.zju.edu.cn (S. Guo), baijing@cad.zju.edu.cn, baijing.nun@gmail.com (J. Bai).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2012.02.001

models. After the text-based search, methods for searching models based on contents have drawn considerable attention, and lead to numerous corresponding studies in the past decades [4,5]. Specifically, in the engineering domain, shape-based 3D model retrieval has been well explored [6].

The shape-based methods mainly focus on analyzing the visual appearances of 3D models and matching the extracted shape descriptors between them. Although this paradigm may be effective in some part searching works, it cannot be directly used in the assembly retrieval. With respect to the assembly retrieval, which is the focus of this paper, it searches and reuses complex mechanical assembly models consisting of many sub-level parts or components. In the assembly retrieval, it is the product structure and the relationships between components that are really important, rather than the overall shapes of assembly models. In other words, two assemblies may be very relevant when their overall visual appearances are quite different. Therefore, the current shape-based methods are unable to tackle this problem well. This limitation of shape-based methods can also cause problems in some specific part searchings [7].

The search method in [3] begins to address the assembly retrieval in a way other than the shape-based methods. Besides the overall assembly statistics (e.g. the number of parts), the mutual relationships and structures existing in assemblies are also considered in this method. However, this method is yet to satisfy practical needs due to the following problems:

(a) The hierarchy in product structure is not considered. In high-level design such as top-down design or innovative design, the rough query with vague components and main relationships needs to be supported by an assembly retrieval system. This cannot be achieved without the hierarchical information.
(b) The semantics of assembly interfaces are not explored. The assembly interfaces between components in an assembly can exhibit many different forms under the same kinematical essentials. Many similar assemblies may be missed in the search if this information is not utilized.
(c) The indexing mechanism is absent. The efficiency will be a problem in case of an extremely large library of assemblies.

In this paper, a new assembly retrieval approach is presented with a view to overcoming the above-mentioned problems. In particular, a multilevel assembly descriptor capturing the main characteristics of assembly models is designed first. In this descriptor, the hierarchical assembly structure and the assembling semantics are extracted and maintained. Then graph-matching based algorithms are used to compare the multilevel assembly descriptors and calculate the corresponding similarities between assembly models. Finally, an indexing mechanism which supports sub-graph matching is given to accelerate the assembly retrieval. In the approach, the information arrangement and corresponding calculation during search is seamlessly integrated into a unified framework. Moreover, in the process of calculation, it is flexible to utilize different portions of the information for different application requirements. As a result, flexible queries can be well supported by our assembly retrieval approach. For example, while a conceptual designer inputs a rough query with abstract components and the kinematic-pair relationships defined between them to search out the assembly models similar in high-level concept, another detailed designer can input a more concrete query with additional information like shapes, layouts or geometric-matings defined. Sometimes the different levels of information can even be mixed together as users want or are accustomed to. And for sure, queries for general retrieval and partial retrieval (i.e. the query model can be a portion of the target model in the assembly structure) are all supported.

The rest of the paper is organized as follows. In the second section, we give a brief review of related works about CAD assembly retrieval. In Section 3 an overview of the flexible assembly retrieval approach is provided. Section 4 defines the multilevel assembly descriptor at length and illustrates the corresponding reasons and thoughts behind it. In Section 5 the details of matching and similarity assessment algorithms are given, while Section 6 describes the indexing and filtering mechanisms. Section 7 introduces the implementation details of the prototype system and analyzes the experimental results. Finally, we conclude the paper and present further work directions.

2. Related works

There are quite a few works dedicated to the content-based model retrieval. The works here we survey are those most closely related to mechanical engineering domain.

As is mentioned above, model retrieval works in engineering can be grouped into three main sub-aspects: the sketch retrieval, the part retrieval and the assembly retrieval.

The sketch retrieval methods mainly process digital images or vector drawings. Surveys [8,9] analyze many relevant works in this area, and some latest typical works include [10–17]. Fonseca et al. present a sketch-based retrieval framework in [17] which could search vector drawings from both CAD and clip-art. The presented approach use hierarchical topology and geometry information of vector drawings to establish the indexing structure, hence it could support both multilevel retrieval and partial retrieval to some extent.

The part retrieval methods search engineering part models of various formats, including mesh models, *B*-rep models, and feature-based models. Tangelder and Veltkamp [5] give an exhaustive review about content-based shape retrieval methods, while Iyer et al. [6] survey the 3d shape searching works specifically relevant to CAD and engineering problems. Moreover, Cardone et al. [18] make a comprehensive comparison of the similarity assessment methods focusing on CAD models. Recently, research in this area is still carried on, and [19–26] are some latest works about it. Li et al. in [24] present a part retrieval method based on the FDAG (feature dependency directed acyclic graph) of part models. The multilevel simplification and sub-part segmentation are executed on the FDAG of part models to generate the general shape descriptors and partial shape descriptors respectively. Based on the descriptors, both multilevel retrieval and partial retrieval of part models could be well supported. In [25], Bai et al. present a multimode partial retrieval method to search part models. In their work, the extended feature tree is first generated from part models based on the semantic features and the relationships between them. Then the hierarchical descriptors of local reusable subparts are calculated according to some design semantics and heuristics. Finally, the multimode partial retrieval could be effectively executed between the query and the reusable subparts in the model library.

Other than the sketch retrieval and part retrieval, the assembly retrieval methods put emphasis on finding useful assembly models promptly and accurately according to users' requirements.

In the past decades, many works have been dedicated to case-based design, in which the "case retrieval" is a crucial step. Aamodt and Plaza give an overview and discuss the general problem of the case-based reasoning approach in [27]. The works [28,29] show the application of case-based reasoning approach in design for assembly, in which topics like how to represent a design case, how to modify or retrieve a case have been discussed. Hu et al. [30] give a case representation and retrieval strategy by using a hierarchical decomposition tree of the product case based on object-oriented technology. In [31], Wu et al. present a platform for rapid design of mechanical products, while a case library is built up and utilized to help shorten the design cycle. Chao and Liu in [32] also propose a case retrieval method, in which the welding

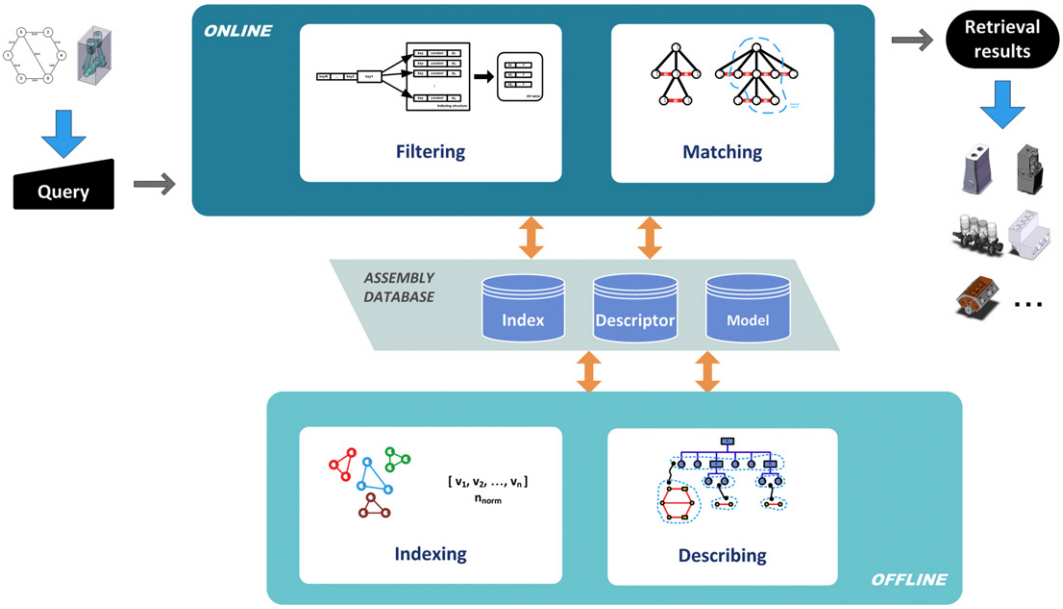


Fig. 1. The flexible assembly retrieval approach.

and assembly processing information is mainly used. More works about case-based design could be found in the two comprehensive reviews [33,34]. The general problem about the retrieval in case-based design is that the case retrieval often relies on some high-level knowledge, like properties of function and design concept, but human intervention could not be totally avoided to get the necessary knowledge. Usually it would be tedious to manually establish a useful design category for case retrieval. Meanwhile, plenty of information like implicit design knowledge kept in today's CAD models is not well utilized.

Regli and Cicirello describe the establishment of an engineering knowledge repository in [35], and in the work, the assembly search problem is also mentioned. As they describe it, the users can create some primitives and define various properties and the relationships between them to start a searching. However, the details about how to do assembly retrieval are not expanded then.

After that, Deshmukh et al. present comprehensive works on assembly retrieval in [3,36,37]. The main idea of these works is mixing multiple assembly retrieval methods into their system, e.g. searching with the statistics and annotations of an assembly such as the sizes, the number of parts or the names of designers. The users can select a method or use several methods together to get reasonable results from the search. A typical one of these retrieval methods is the use of mating graph, which can effectively support the partial retrieval of the assembly. In this method, users can input a segment of the mating graph as a query and find the assemblies whose mating graph contains the query. (The Ullmann algorithm [38] is adopted to handle the sub-graph isomorphism problem.) However, this cannot support high-level queries well, since it requires the users to know in advance some segments of the most detailed implementations in the desired model, while the design activities often work on some abstract components and the high-level relations among these components. The orientation relationships between assembly joints are also used in their approach, which can be beneficial in some cases. However, this information is inadequate to support some high-level queries, e.g. a kinematics-graph of an assembly. Meanwhile, the joints information needs to be available in the library models beforehand in the work, which most of the assembly models do not normally provide. Moreover, the mainstream exact sub-graph isomorphism algorithms do not run in polynomial time, and there are no indexing mechanisms described in the works dedicated to the acceleration of the partial retrieval. Once the model library gets bigger (200 assemblies are used in the experiments of the work), the efficiency of the algorithm may become a bottleneck.

Recently, more and more attention is paid to the ontology-based technology. Gaag et al. [39] present a function-based method for retrieving design solutions. In their approach, existing solutions are described in terms of functions which are realized in an ontology model, and then the semantic search and reuse can be conducted based on the built structure. However, the establishment of ontology structure is nontrivial. Many design documents are needed to extract the necessary information, and the built structure is often restricted to some specific applications. Similarly, there are some other retrieval related works focusing on some specific application domains, such as [40,41]. Kim et al. describe an assembly model retrieval method based on the Web Services technology in [42], but their focus is on the data exchange between cooperators in collaborative design, rather than search relevant assemblies from a large product database.

Today's industrial applications tend to have more requirements on model retrieval for multilevel, partial and semantic characteristics. It can be seen that recent research has also noticed the trends and presented quite a few approaches to address it in sketch retrieval and part retrieval. Besides the sketch retrieval and part retrieval mentioned in the works above, we believe that today's assembly retrieval should also possess multilevel, partial and semantic characteristics, which are the very objectives of the work in this paper.

3. Overview of the flexible assembly retrieval approach

Fig. 1 shows the overview of our flexible assembly retrieval approach. It could be seen that our approach contains three main parts, i.e. the online processing, the offline processing and the assembly database. Here we give a brief description of each part respectively:

a. *Assembly database*

The assembly database stores all the necessary data involved in both online processing and offline processing, including all the assembly models collected from internet repositories, the multilevel assembly descriptors generated from these models for matching and an indexing structure established on these models for filtering. Moreover, the correlations between these data are also maintained.

b. *Online processing*

The online processing starts when a user inputs a query and ends with exporting the assembly models similar to the query. This processing requires high efficiency to respond promptly to the users; meanwhile good distinguishability is also needed to provide accurate retrieval results in a reasonable rank order. Two main steps are contained in this processing:

Filtering

In this step, a fast filtering is carried out based on the indexing structure stored in the assembly database. Several discriminating keys are first calculated from the input query, and then the keys are used to search in the indexing structure to rapidly exclude abundant assemblies in the library that are unable to match the input query appropriately.

Matching

After rough filtering, the remaining assembly models still need to be compared with the input query in a more subtle way. The graph-matching based algorithm is adopted in this step to conduct the calculation, with the help of the multilevel assembly descriptors stored in the assembly database. Based on the matching results, discriminatory similarities between the input query and the assemblies in the database are calculated. Finally, the corresponding assemblies in the assembly database are sorted by similarities and retrieved for the users.

c. *Offline processing*

The offline processing handles all the models in the assembly database and pre-calculates all the relevant data needed by online processing. These pre-calculated data are then stored into the assembly database. By this prior processing, the operations and calculations in real-time search can be minimized to provide fluent human–computer interaction. There are also two main steps in this processing:

Describing

In this step, all models in the assembly database are parsed and analyzed to generate their corresponding descriptors. These assembly descriptors contain multilevels of data, including the topological structures which depict the whole product structure, the assembling semantics which indicate the mutual relationships between product components, and the geometric information which reveals the product layout and component shapes.

Indexing

After the multilevel assembly descriptors are generated, an efficient assembly indexing structure is established to support the filtering step. Here the topological structures with a portion of the assembling semantics in the assembly descriptors are used to calculate the discriminating keys, and then the keys are inserted into an indexing structure called NB-Tree. Moreover, a subdivision mechanism is used here to enable the support of partial retrieval from the indexing structure.

In general, the presented assembly retrieval approach utilizes more semantic information than previous content-based retrieval approach in achieving better discriminability. Meanwhile, different levels of information are arranged into a unified calculation framework to facilitate the support of different search styles. Besides that, prompt partial retrieval of assembly models can be better supported through the specific indexing strategies. All the details about these characteristics and main steps of the approach will be given in the following sections.

4. A multilevel assembly descriptor

In order to effectively achieve flexible assembly retrieval, a reasonable and comprehensive assembly descriptor involving various levels of data from high-level information such as kinematical properties to low-level information like geometric shape is a prerequisite.

In the work, we present a multilevel assembly descriptor for both queries and database models involved in assembly retrieval. The information in the presented assembly descriptor is multilevel in three aspects. First, it includes topological structure, assembling semantics and geometrical information; second, the topological structure is hierarchical; third, the assembling semantics has multiple layers. The details about them are given below.

4.1. Topological structure

The presented assembly descriptor is based on a topological structure which can be one of the most distinguishable characteristics between different assemblies, i.e. the *hierarchical assembly structure*.

In general, the assembly descriptor used in the work is represented upon a hierarchical data structure [43], which is the foundation of today's mainstream assembly representation. In the hierarchical assembly structure, the "part-of" relationship represents the connections between an assembly and its components, while the assembly interface represents the connections among all the components of an assembly. It is important that a component of an assembly can still be a subassembly, which in turn has its own subcomponents. Those leaf components without children are called parts, which are the elementary components in an assembly model. As a result, a treelike hierarchical structure is formed based on the assembly, subassemblies, components and the relationships between them.

Fig. 2 shows an engine model and its corresponding hierarchical assembly structure. The engine is composed of six components, i.e. one main block, one crank shaft, two pistons and two connecting rods. The connecting rod itself is a subassembly, which is again composed of two subcomponents, i.e. the upper rod and the end cap. As shown in the figure, the part-of relationships (blue edges with arrows) form a two-level treelike structure, which illustrates how the whole engine model is composed from many components, including subassemblies and parts. Moreover, the assembly interfaces (red edges) form three graph structures, corresponding to the top engine assembly and the two connecting rod subassemblies respectively. The graph structure is the assembling-graph illustrating how the components are connected together. In fact, these connections imply the cooperation manners between the components which finally make the assembly work.

Some previous works use connections among all parts in an assembly model as the topological descriptor to search assemblies. Instead of that, we adopt the hierarchical descriptor here for the following three reasons:

(a) Hierarchical representation contains more implicit design intents than a flat graph, and it is better supported by modern CAD systems. The product construction sequence, component proximity, relative importance and some other information can all be more clearly seen in a hierarchical representation.
(b) Assembly retrieval for high-level design such as top-down design or innovative design can be better supported. In these design works, the high-level structure, consisting of major components or subsystems of the desired product with the desired interfaces, relationships and constraints among them [44,45], is crucial, and the components defined can even be vague and incomplete in geometric details. Therefore, when users provide a query for assembly retrieval in design works,

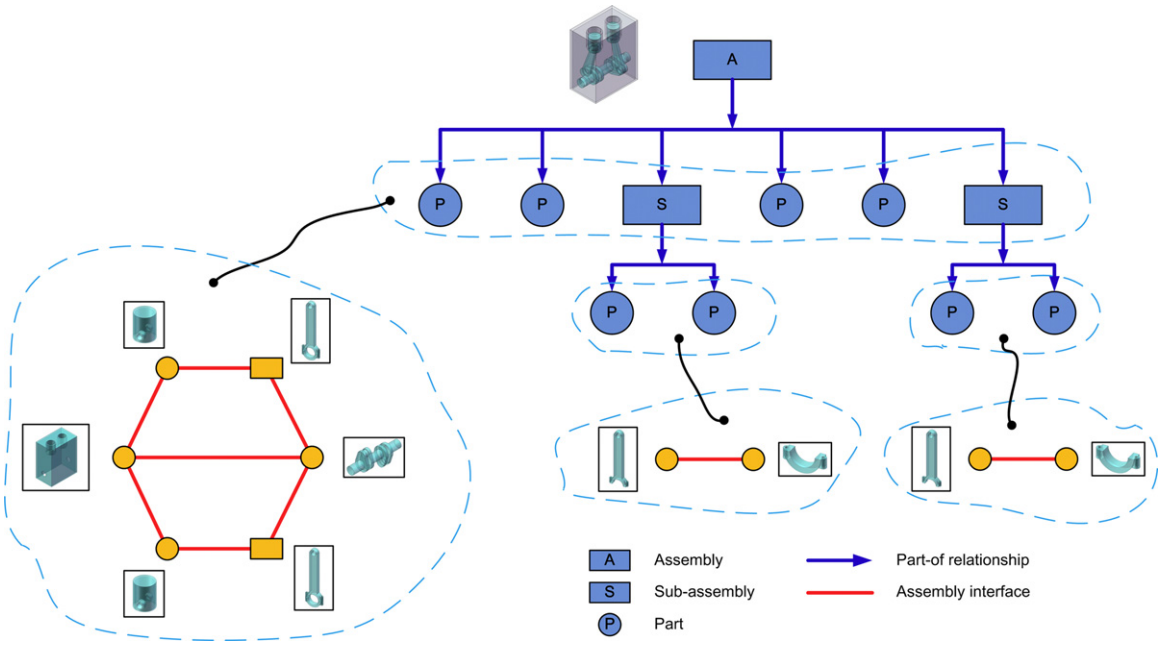
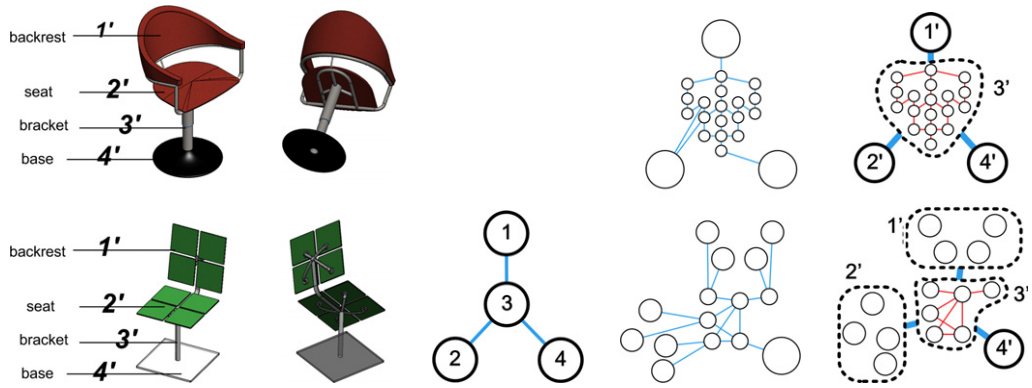


Fig. 2. The hierarchical assembly structure of an engine model.



(a) Chair models with main components. (b) Back view of chair models. (c) Query structure. (d) Non-hierarchical structure. (e) Hierarchical structure.

Fig. 3. High-level structure preservation of the hierarchical representation.

they firstly tend to think of the model structure as simple as possible (in high-level), that is, the trivial details will be simply ignored. On the other hand, the hierarchical representation for an assembly can just maintain this kind of high-level structure, i.e. main components and main interfaces between those components without interference from plenty of part details. Fig. 3 shows this idea and illustrates what benefits the hierarchical representation could bring. Two chairs with their front view (Fig. 3(a)) and back view (Fig. 3(b)) are shown. In high-level design, a structure shown in Fig. 3(c) may be input as a query with four high-level components (corresponding to the backrest, the seat, the bracket and the base annotated in Fig. 3(a) respectively) and their interfaces defined. However, the chair models are composed of much more parts other than the four main components. For example, the bracket is composed of 18 subparts in the above chair model and is composed of 6 subparts in the bottom chair model. Therefore, the non-hierarchical representations of these chairs tend to be quite messy (Fig. 3(d)) and are unable to match the query structure appropriately. In contrast, the hierarchical representations of these chairs (Fig. 3(e)) can match the query perfectly since the top-level components and relationships are maintained. The red edges in Fig. 3(e) are the second-level relationships between second-level components, which could be seen as the inner-relationships of top-level component 3′ (the bracket).

(c) Partial-retrieval could be better supported since the hierarchical representation has many meaningful subassemblies defined in it.

4.2. Assembling semantics

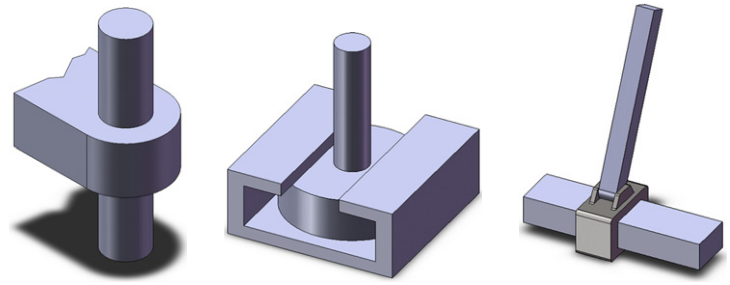
The assembling semantics describes the essential connotations of the interactive relationships between components in an assembly, which are fundamental to the function and behavior of the assembly.

4.2.1. Semantic assembly interface

In the presented assembly descriptor, the assembly interface is also defined as multilevel. Table 1 shows the semantic assembly interface representation used in our assembly descriptor.

Table 1
Semantic assembly interface.

Function Layer	**Degree of Freedom**						
	Translation			Rotation			Composition
Implementation Layer	**Kinematic Pair**						**Interface Part**
	Revolute	Prismatic	Screw	Cylindrical	Spherical	Planar	Essential
	Gear	Universal	Point on Surface	Point on Planar Curve	Surface	Fixed	Accessorial
Geometry Layer	**Geometric Mating**						
	Coincident	Concentric	Distance	Tangent	Perpendicular	Parallel	
	Point on Line		Point on Surface		Edge on Surface		Lock



(a) Cylinder. (b) Slotted cylinder. (c) Slider with pin.

Fig. 4. Different implementations of the same DOFs.

There are three major levels in this descriptor: the function layer, the implementation layer, and the geometry layer. Information in each layer is more abstract and intensive than in the layers below it.

Function layer:

This layer represents the design abstraction of an assembly interface. Ambrósio and Eberhard [46] describe three major stages for the essential engineering design tasks in the systematic development process of rigid body mechanisms, in which the determination of the degree-of-freedom (DOF) is put in the first step. Meanwhile, Molian [47] and Chiou and Sridhar [48] have also mentioned that the DOF is the intrinsic property of a mechanism. As a result, what we put here in the first layer is the statistical information about the relatively independent DOF between connected components of an assembly interface, i.e. counts of translational, rotational and composite DOFs. Composite DOF is a specific DOF as the result of constraining multiple DOFs together, e.g. the 1-composite DOF in the screw joint is the effect of imposing a fixed ratio between 1-translational DOF and 1-rotational DOF. Actually, there can be different ways for implementing the same DOF requirements, i.e. the same design abstraction (Fig. 4 shows three different implementations of 1-translational DOF and 1-rotational DOF).

Implementation layer:

Information in this layer describes the ways selected by designers for implementing the design objective of the interface. The counts of various kinds of kinematic pairs and interface parts are stored in this layer. Generally, kinematic characteristics in most assembly interfaces are composed of several elemental kinematic-pairs from the ones we selected, which include six lower-pairs (Fig. 5 shows the sample geometrical forms of them.) and some typical higher pairs.

Interface parts are specific parts which are only meaningful for implementing assembly interfaces and do not participate in any other main functions. These parts are further divided here into two subtypes, i.e. the essential and the accessorial. The essential interface parts are crucial for kinematic implementation (e.g. the black parts of the swivel joint in Fig. 6(a)), while the accessorial ones do not affect essential kinematic property of the assembly interface and could thus be ignored (e.g. the bearing in Fig. 6(b)) in a way.

Geometry layer:

This layer contains various kinds of geometric-matings typically used in today's assembly modeling. (The information in the table is established based on SolidWorks.) Counts of these geometric-matings expose information about the detailed shape of an assembly interface.

4.2.2. Multiple interpretations

Since the assembling graphs in the hierarchical assembly structure (see Fig. 2) are composed of components (nodes) and assembly interfaces (edges), the assembling graph can actually be interpreted as multiple graphs (Fig. 7) with different levels of information according to the definition of the semantic assembly interface.

Due to the possibility of multiple interpretations, users could import a quite abstract assembling graph as the query, such as the DOF graph (Fig. 7(a)) or the kinematics graph (Fig. 7(b)). Moreover, if there is a module for designing the schematic diagram of mechanism, the diagram could be directly converted to a kinematics graph and used as a query for assembly retrieval, which is shown in Fig. 8.

The geometric-mating graph (Fig. 7(c)) is used in previous work [3] as a kind of descriptor for assembly retrieval. However, the method may not be very effective when users want to use high-level knowledge in assembly retrieval. For example, the two assemblies in Fig. 9 have different geometric-matings, but they are just two geometric forms of a prismatic-pair. If geometric-mating graph is used alone, the two assemblies are not similar at all, though they have the same kinematical semantics. Once the users input a graph with only geometric-mating information, either of the two assemblies cannot be retrieved. On the other hand, if the users input a graph with high-level kinematical information, the two assemblies in Fig. 9 can all be retrieved.

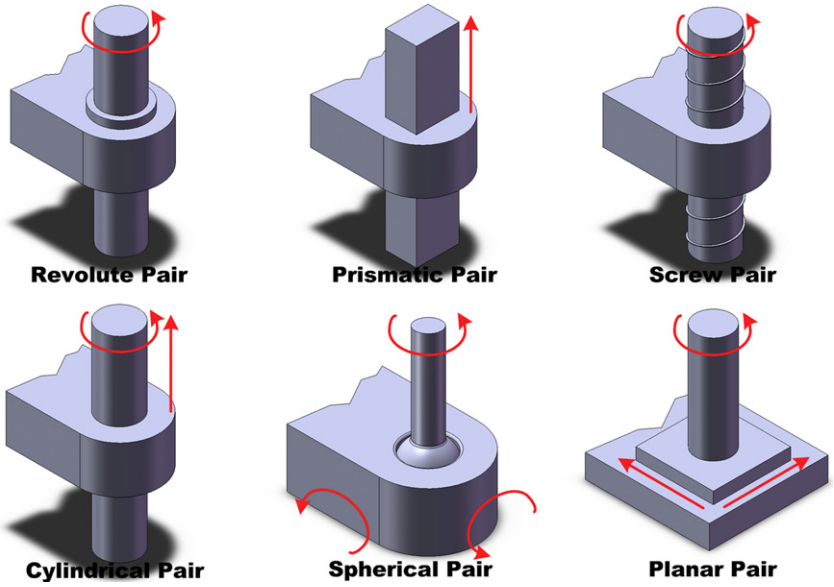


Fig. 5. Sample geometrical forms of lower kinematic-pairs.

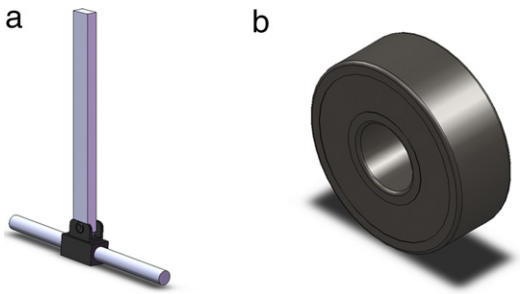


Fig. 6. Samples of interface parts. (a) swivel joint; (b) bearing.

In general, with the three different levels of information, the assembly descriptor could take full advantage of rich design knowledge embedded in assembly models and could support more flexible queries. (These information could even be mixed together as users want, see Fig. 7(d).) As a result, not only the assemblies similar in low-level, but the assemblies similar in high-level can also be retrieved.

4.3. Geometrical information

The topological structure with the assembling semantics in multilevel assembly descriptor gives the skeleton of an assembly, which represents the global characteristics and dominates the similarity between two different assembly models. On the other hand, the geometrical information is like the corresponding muscles of an assembly model, which represents the local characteristics and acts as supplementary information.

Each node in the topological structure represents a component of the assembly model. Geometrical shape information is calculated for each component of an assembly model and stored in its corresponding node as properties. Here in the multilevel assembly descriptor, different types of components need different calculations for the geometrical information (Fig. 10). If the component is a part, shape distribution vector [49,50] of the part shape is computed; if the component is an assembly or a subassembly, the shape distribution vector of the bounding-box (or convex-hull) of the component is computed. In addition, the geometrical center of each component is also calculated and stored into the nodes of

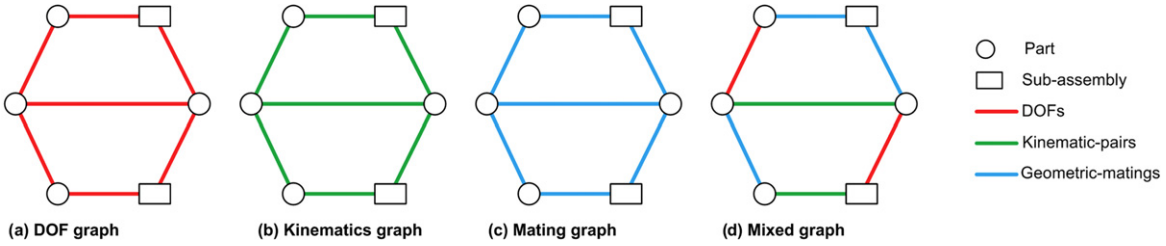


Fig. 7. Multilevel interpretations of assembling graphs.

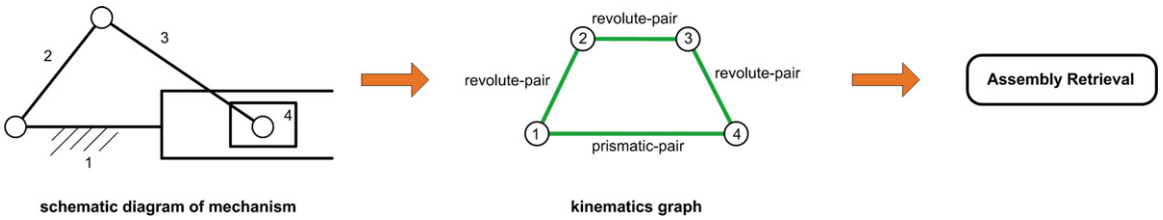


Fig. 8. Schematic diagram of mechanism as query.

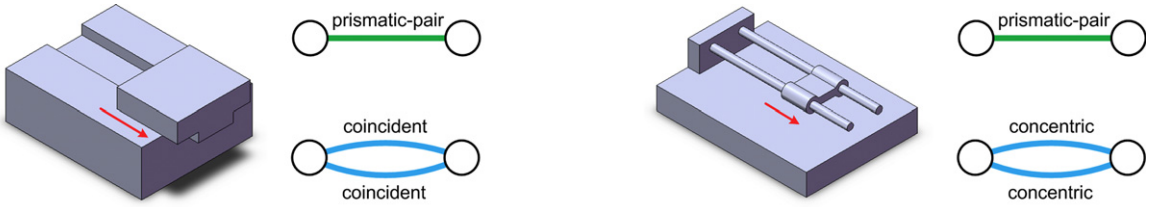


Fig. 9. Two geometric forms of prismatic-pair.

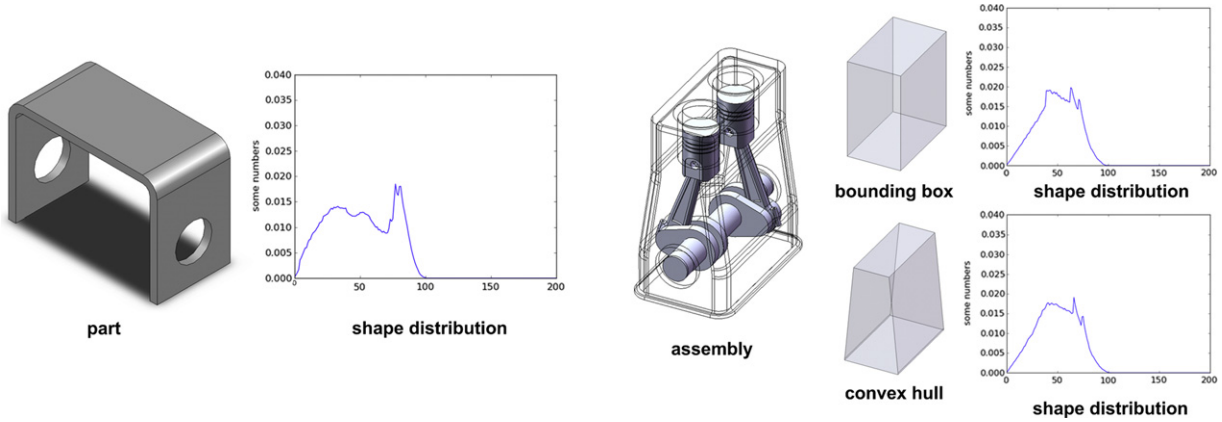


Fig. 10. Shape distributions of part and assembly.

the descriptor. The geometrical information collected here is further re-organized together for the comparison of assembly models, which is discussed in the section of assembly matching.

4.4. Other useful information

Besides the information described above, there is some other product information which can be quite helpful for searching assemblies, including but not limited to:

Functions—the specific processes, actions or tasks that a system or component is able to perform.

Loads—the forces, deformations or accelerations applied to a structure or its components.

Environmental conditions—the environment in which the product is expected to work such as the temperature or safety factors.

The semantic information can reduce the search space and accelerate the assembly retrieval indubitably. Unfortunately, most of the existing assembly models we get from the internet repositories do not have the accompanying design documents or specifications which are necessary for extracting the semantic information. Therefore, we could not use the information in the work currently but just reserve the corresponding positions in descriptors. How to extract and utilize the information will be considered in the future. In fact, once the information is present or extracted, it can be incorporated into our multilevel assembly descriptor seamlessly as suitable properties in the hierarchical assembly structure.

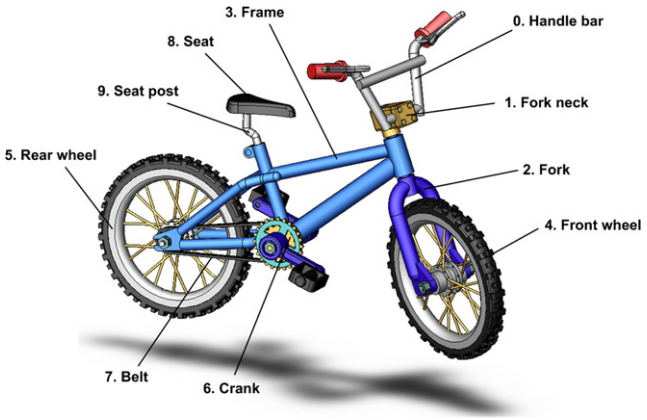
4.5. Generation of multilevel assembly descriptors

The hierarchical assembly structure information can be extracted from CAD assembly files. The part-of relationships are parsed to construct the hierarchical tree. On the other hand, the assembly interface relationships are established based on the geometric-matings defined between components in an assembly. Once there are one or more geometric-matings between two components, an assembly interface is created between them.

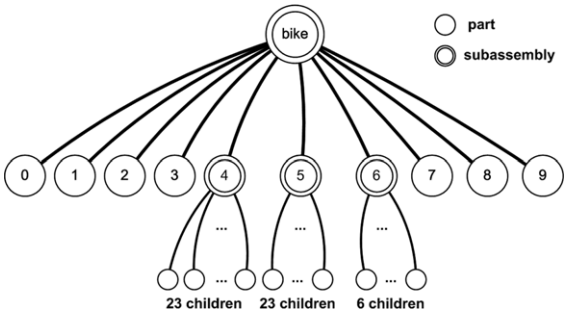
Multilevel information in the semantic assembly interface is generated from bottom-level to top-level. After the construction of the hierarchical assembly structure, the information on geometric-matings in an assembly interface is already extracted and stored. Then, the kinematic-pairs and DOFs are derived from the information about geometric-matings based on the automatic identification approach of kinematic-pairs mentioned in report [51]. The general idea is firstly mapping each geometric-mating in an assembly interface to the elementary type of translational and rotational freedom, and then all the elementary freedoms in an assembly interface are reduced based on the reducing rules of freedom presented in [52]. Finally the reduced freedom of an assembly interface is mapped to the corresponding kinematic-pair. (Tables in the section of Appendix show the relevant definitions, mapping and reducing rules.) With this method, some typical kinematic-pairs can be identified automatically, and the gear-pair is identified by another method mentioned in [51]. Other complex kinematic-pairs defined in the descriptor are labeled manually in the work if they appear in the assembly models. The interface-part is also manually labeled currently; the reasonable automatic identification of the interface-part should be addressed in the future because of its non-ignorable role in the graph-based structure. Fortunately, most of the models used in the current work can be automatically parsed without human intervention.

Geometrical information is generated on each component of the assembly model. Shape distribution information is calculated by the method presented in [49] for components' shapes; Geometrical center of each sub-component's shape is also calculated and is used when comparing their parent assembly's layout information in the matching step.

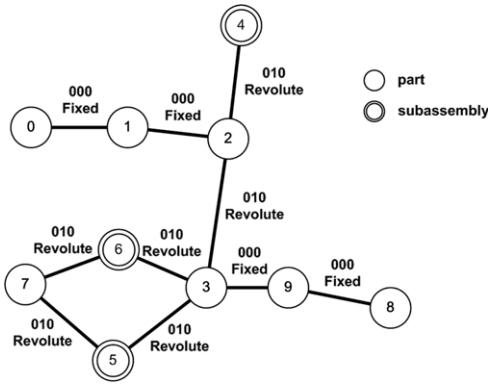
Fig. 11 shows an example about the generation of multilevel assembly descriptors. The hierarchical assembly structure is shown in Fig. 11(b), while the first-level assembling graph of the bike assembly is shown in Fig. 11(c). Fig. 11(d) shows the typical information generated and stored in edges and nodes. The number labeled on the graph edge is the DOF information, e.g. '010' means zero translational-dof, one revolutional-dof and zero composite-dof.



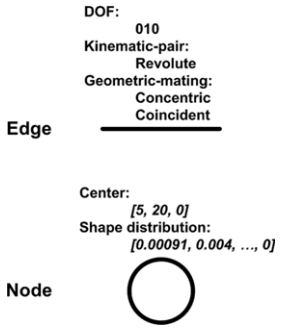
(a) A bike assembly.



(b) Hierarchical assembly structure.



(c) First-level assembling graph of the multilevel assembly descriptor.



(d) Information in edges and nodes.

Fig. 11. Sample of the multilevel assembly descriptor generation.

Finally, the storing of multilevel assembly descriptors is based on its hierarchy structure. Each unique subassembly in an assembly is stored separately as a single item in the database, and its corresponding descriptor has just one sub-level topology expanded. Correlations are established between these subassemblies for reconstructing the whole hierarchy structure during the matching process. This strategy can minimize the storing size and the comparing time, and the partial retrieval of subassemblies can be achieved more conveniently at the same time.

5. Assembly matching

During the assembly retrieval, the query descriptor needs to be compared with the assembly descriptors in the database to get the similarities between them. Since the multilevel assembly descriptor contains abundant information, the matching process is divided into the following two main steps.

5.1. Primary comparing

The topological structure of the multilevel assembly descriptor is firstly compared, and an effective and efficient graph matching algorithm is needed. Here we do a hierarchical graph matching based on the VF2 sub-graph isomorphism algorithm [53] for its exactness and high performance. An exhaustive survey of the graph matching algorithms can be found in [54].

Fig. 12 shows an example of the hierarchical graph matching process. At first, the sub-graph isomorphism algorithm is executed on the top-level assembling graphs of the two assembly descriptors to be compared. Then, if any node in the top-level assembling graph of the query is a subassembly, the sub-graph isomorphism algorithm is executed recursively between the graph under this node and the graph under the matched node in the descriptor of the database assembly. Finally, if a corresponding node in the descriptor of the database assembly has been found for each node in that of the query, then a successful match between the multilevel assembly descriptors is found. As shown in the process, partial retrieval can also be well supported by the matching algorithm.

In the process, the sub-graph isomorphism algorithm uses not only the topology of assembling graph, but also some assembling semantics information to help the search space pruning in the algorithm. The degree-of-freedom information (mentioned in 4.2.1) in the corresponding edges are required to be equal between two assembling graph; On the other hand, there is a value stored in each node as a property representing the count of child nodes under this node (for part, the value is 0), and this value in the node of the query graph is required to be less than or equal to the value of the corresponding node in the graph of the database model (otherwise, the recursion cannot be carried on appropriately).

A successful match M is a mapping from descriptor $D1$ to descriptor $D2$, in which each node n in $D1$ is mapped to a node $M(n)$ in $D2$ and each edge e in $D1$ mapped to an edge $M(e)$ in $D2$. This injective mapping is not bidirectional since it is regarded as validate in the work that the query descriptor topology is a subpart of the assembly descriptor topology in the database but not vice versa.

5.2. Secondary refining

After a successful match M is found, the similarity between two descriptors needs to be calculated based on the match M

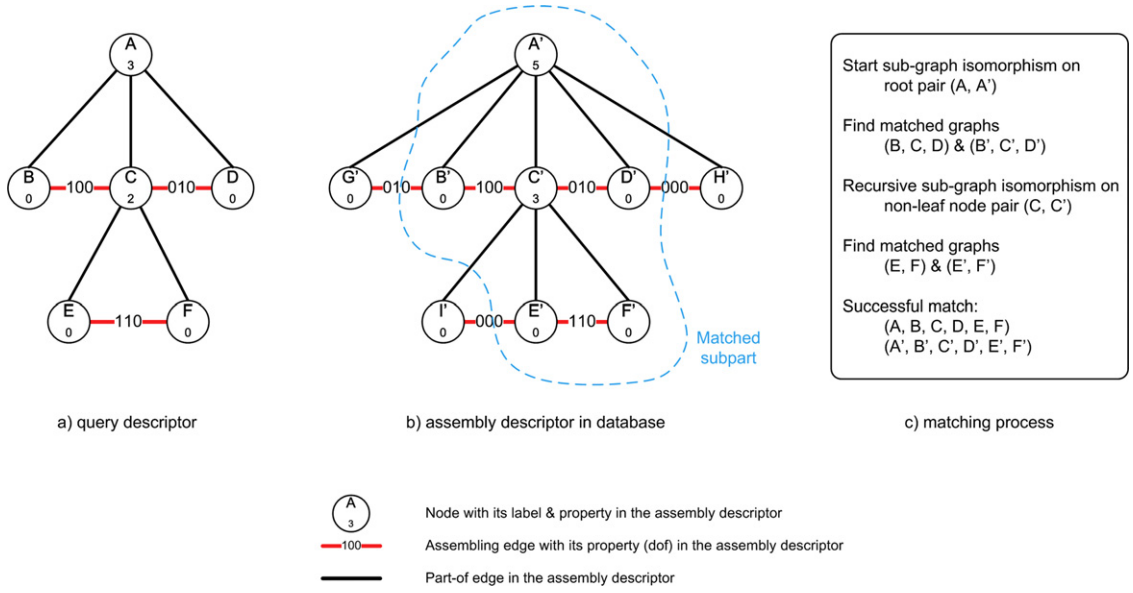


Fig. 12. Hierarchical graph matching between assembly descriptors.

and the properties in corresponding nodes and edges of the two descriptors. According to the assembly interfaces of the assembly descriptor, nodes in the descriptor can be divided into different isolated assembling graphs. For example, the query descriptor in Fig. 12 has two assembling graphs, i.e. (B, C, D) and (E, F), which are mapped to assembling sub-graphs (B', C', D') and (E', F') respectively. The similarity is calculated as the weighted sum of the similarities of the matched assembling graph-pairs. A heuristic rule about the weight setting is that the weight of the high-level assembling graph is larger than the weight of the low-level assembling graph (here the word "level" refers to that of the hierarchical structure).

$$\text{Similarity}(M) = \sum_{g \in \text{Descriptor}} \omega_g * \text{Similarity}(g, M(g))$$

$$\sum_{g \in \text{Descriptor}} \omega_g = 1.$$

For each matched assembling graph-pair $(g, M(g))$, the calculation of similarity can be divided into two parts, i.e. the assembling similarity and the geometrical similarity.

5.2.1. Assembling similarity

The assembling similarity mainly describes how similar two assembling graphs are in their topological structure. For each matched edge-pair in a matched assembling graph-pair, the similarity between the two assembly interfaces represented by the matched edge-pair is calculated, and their weighted sum is the assembling similarity between the two assembling graphs in the matched pair.

$$\text{Similarity}(g, M(g)) = \sum_{e \in g} \omega_e * \text{Similarity}(e, M(e))$$

$$\sum_{e \in g} \omega_e = 1.$$

The calculation of the similarity between two assembly interfaces is based on the definition of the semantic assembly interface (4.2.1). There are four categories of values stored in an assembly interface as shown in Table 1. The first category DOF is used in the sub-graph isomorphism algorithm as described above, and the other three categories of values are used here to calculate the similarity between two assembly interfaces. Each category has its own weight and heuristically the category in the implementation layer has greater weight than the category in the geometry layer of the semantic assembly interface. The similarity between two assembly interfaces is the weighted sum of the similarity between two corresponding categories.

$$\text{Similarity}(e, M(e)) = \sum_{c \in \text{Category}(e)} \omega_c * \text{Similarity}(e_c, M(e)_c)$$

$$\text{Categories}(e) = \{\text{kinematic-pair, interface-part, geometric-mating}\}$$

$$\sum_{c \in \text{Categories}(e)} \omega_c = 1.$$

The similarity between two corresponding categories is then calculated as one minus the distance between them. This distance is the weighted sum of the distance between corresponding values in each type slot (e.g. type "prismatic-pair" in category "kinematic-pair"). The weight of each type slot is defined uniformly among all types in a category. Moreover, the value of a type-slot is a count corresponding to the type, e.g. the count of prismatic-pairs presented in an assembly interface.

$$\text{Similarity}(e_c, M(e)_c) = 1 - \text{Dist}(c)$$

$$\text{Dist}(c) = \omega_t * \sum_{t \in \text{Types}(c)} \text{Dist}(t)$$

$$\text{Types}(c) = \{\text{all types in category } c\}$$

$$\omega_t = \frac{1}{\text{Cardinality}(\text{Types}(c))}$$

$$\text{Dist}(t) = \frac{|(e_t - M(e)_t)|}{\text{Max}(e_t, M(e)_t)}.$$

Fig. 13 shows the main steps in the calculation of assembling similarity. In general, this calculation mainly uses the properties of the topological edges; hence the result can reflect the similarity between two different assembling structures.

5.2.2. Geometrical similarity

Other than the assembling similarity, the geometrical similarity mainly describes how similar two assembling graphs are in their geometrical properties. There are two kinds of information involved in the calculation of the geometrical similarity: layout and shape.

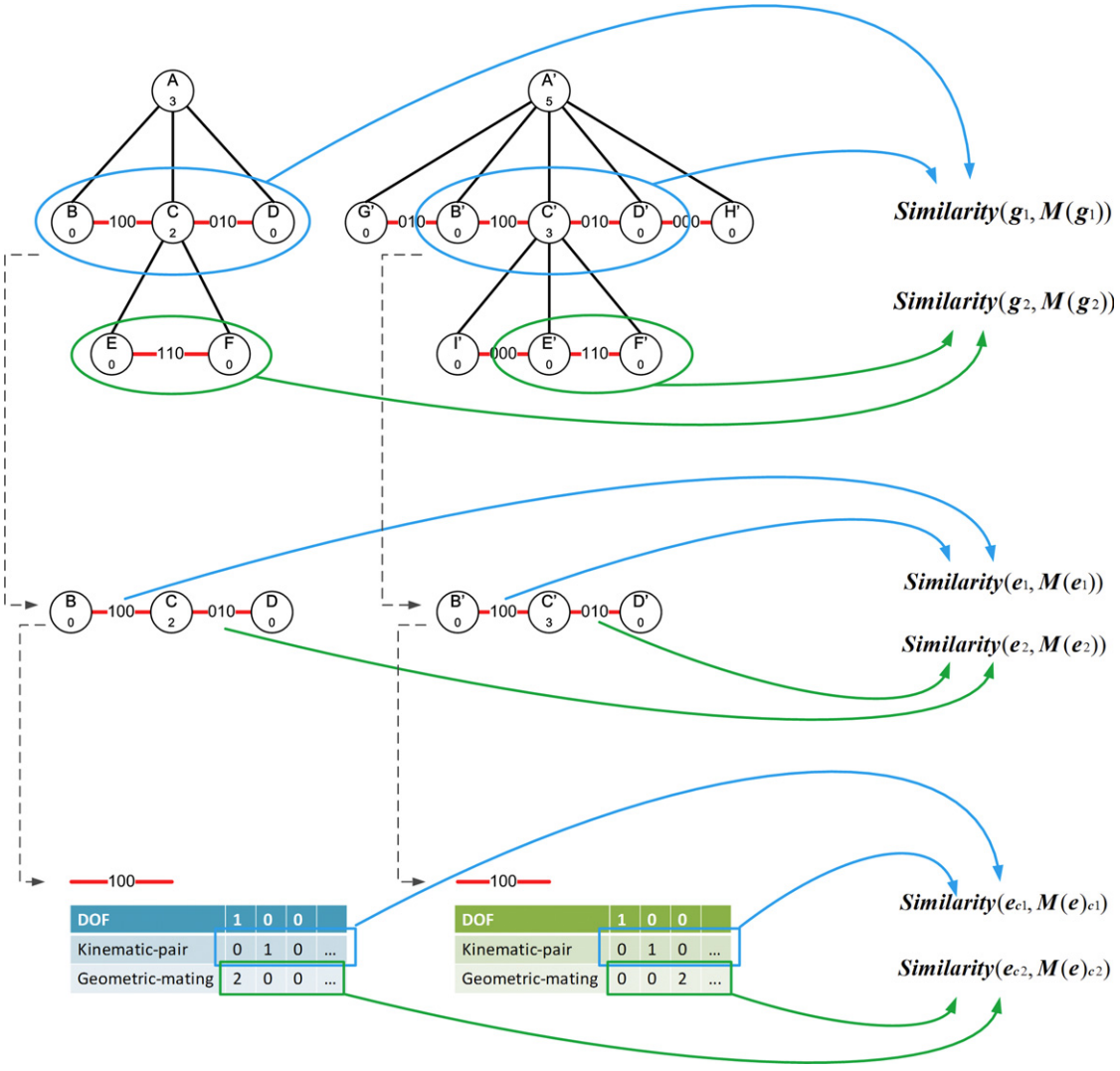
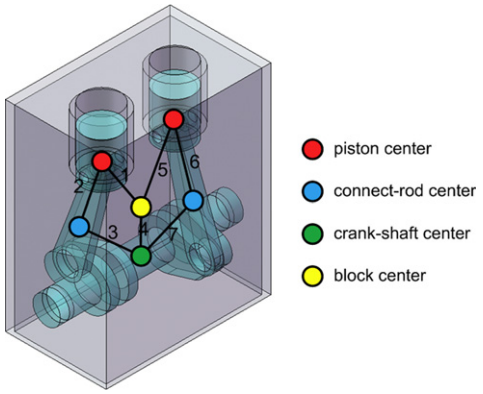


Fig. 13. Assembling similarity calculation.



(a) Assembly-bone.

	Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7
Edge 1	Ø	a1	Ø	a2	a3	Ø	Ø
Edge 2	a1	Ø	a4	Ø	Ø	Ø	Ø
Edge 3	Ø	a4	Ø	a5	Ø	Ø	a6
Edge 4	a2	Ø	a5	Ø	a7	Ø	a8
Edge 5	a3	Ø	Ø	a7	Ø	a9	Ø
Edge 6	Ø	Ø	Ø	Ø	a9	Ø	a10
Edge 7	Ø	Ø	a6	a8	Ø	a10	Ø

(b) Angle table.

Fig. 14. Layout information for geometrical similarity calculation.

Layout information

The layout information mainly describes how the components of an assembly are arranged in the 3D space. This information can play a good role supplementary to the assembly's assembling structure in topology. An abstract representation "assembly-bone" which describes the layout of an assembly is shown in Fig. 14(a).

The assembly-bone is a spatial geometrical-structure composed of several line segments connecting the geometric-centers of components. Each line segment for connecting the geometric-centers of two components is present only when there is a corresponding assembly interface between the two components. A natural method for comparing two assembly-bones is extracting their shape-distribution vectors as in [49]. However, the matched

assembling graph in the database model may often be a sub-graph of the whole assembling graph. Since it is impractical to compute the assembly-bones of all the sub-graphs when establishing the database because of the unacceptable time and space requirements, the shape-distributions have to be computed during real-time comparing, which will waste too much retrieval time. As a result, we use another method "angle table" (Fig. 14(b)) here for a comparison of the assembly-bones. The angle table is an $N \times N$ matrix. N is the count of edges (line segments) in the assembly-bone. The row and column represent the edges of the assembly-bone listed in an assigned order. A value in position $[i, j]$ is the spatial angle between the ith edge and the jth edge. If two edges are not connected directly (to a unique geometric-center), the corresponding value is left unassigned. When the corresponding angle tables of two assembly-bones are constructed, they could be compared as vectors (Rows in matrix can be concatenated.), and the result of the comparison is the layout similarity between the two matched assembling graphs. A pre-requisite of this comparing is arranging the edges in the two tables in the same order based on the result M of the topological match mentioned in 5.1. (Each edge in one assembly-bone has its matched edge in the other one.)

Shape information.

The shape information describes the shapes of sub-components in the assembling graph. As mentioned in Section 4.3., the shape distribution vectors of the sub-components are calculated in advance. Here these vectors are compared directly, and the weighted sum of the similarities between the vectors of corresponding sub-components is the overall shape similarity between two matched assembling graphs.

After the assembling similarity and geometrical similarity are calculated respectively, their weighted sum multiplied by an attenuation factor is the similarity between two assembling graphs in a matched assembling graph-pair. The attenuation factor depicts how much the matched assembling sub-graph takes up as a sub-part in the original one in the descriptor of the database model (e.g., how much (B', C', D') takes up in (G', B', C', D', H') as shown in Fig. 13). At last, all the similarities between assembling graph-pairs in a successful match are summed together as described before to get the final result.

One thing deserving to be mentioned is that, since the steps in similarity calculation are independent to some extent, flexible calculation in practical assembly retrieval can be provided. For example, when a user submits a query without geometrical information, the calculation of geometrical similarity can be omitted. Moreover, when a user submits a query with only DOFs and kinematic-pairs information, the comparison of geometric-matings can also be omitted in the calculation of assembling similarity. This kind of flexibility gives the users many extra-controls and extra-conveniences on submitting the query during practical assembly retrieval, and different levels of matching for assembly retrieval can be carried out according to the input query. Meanwhile, partial retrieval can be well supported since (a) all subassemblies are parsed and stored as candidates in advance, and (b) the matching algorithm has the ability to find partial segments similar to the query in target graphs.

6. Assembly indexing and filtering

Although the VF2 sub-graph isomorphism algorithm used in the hierarchical graph matching is a quite efficient algorithm among the exact graph matching algorithms, it may still make the response time unacceptable when the amount of assemblies in the library becomes extremely large during practical assembly retrieval. Therefore, an efficient indexing structure for quick filtering of unmatchable models should be established for the assembly library to accelerate the assembly retrieval, especially the partial retrieval.

6.1. Indexing

To index the multilevel assembly descriptor, statistical values such as the number of components can be useful in many cases, and it is not hard to integrate these values into our assembly retrieval approach. Here, another indexing mechanism which has better discriminability and good efficiency is presented.

In the indexing mechanism, the first thing that needs to be done is vectorizing the multilevel assembly descriptor. We choose to vectorize the first-level assembling graph of the assembly descriptor because of its key-role for discrimination of assemblies. The heuristics behind is that if the descriptors of two assemblies cannot be matched appropriately on topology of the first-level assembling graph, then they are not regarded as similar models. There are many studies on the vectorization of graphs, such as the spectral graph theory [55] which studies the adjacency matrix of graphs by means of linear algebra. In the work, we adopt the random walks method discussed in [56] to vectorize the assembling graph for its good discrimination ability.

A Random Walk (RW) on a given graph G is described by a probabilistic model which allows us to compute the probability $xp(t)$ of being located in each vertex p at time t. A steady state distribution of the RW can be calculated as:

$$\boldsymbol{x}^* = \frac{1-d}{N}(I - d\boldsymbol{W})^{-1}\prod = \frac{1-d}{N}\sum_{k=0}^{\infty} d^k \boldsymbol{W}^k \bullet \prod,$$

where d is the damping factor, $\prod$ is a vector whose entries are all equal to one and $\boldsymbol{W}$ is a matrix whose element at position $[i, j]$ is the probability of following an edge from vertex i to vertex j. If there is no edge connecting vertex i and vertex j, the value is 0. If there is edge connecting vertex I and vertex j, then the probability is given as:

$$\boldsymbol{x}(j|i) = \frac{f(\boldsymbol{v}_j, \boldsymbol{v}_i)}{\sum_{k \in ch[i]} f(\boldsymbol{v}_k, \boldsymbol{v}_i)}.$$

The $ch[i]$ is the set of vertices directly connected to vertex i. The function f is the tendency function which calculates the tendency value of following an edge from vertex i to vertex j by using the node properties or edge properties. The tendency value used in the work is the integer value calculated from the DOFs information in an assembly interface. For example, one translational and one rotational DOF without composite DOFs are expressed as integer 110. The details about the random walk method can be found in [56].

The steady state distribution vector as is calculated above is an n-dimensional (n is the count of nodes in the graph) vector which can be used as an abstraction of the assembly descriptor. Therefore, the indexing structure should be able to support the searching in high-dimensional spaces ([57] is a good survey for the high-dimensional indexing techniques). Here we adopt an indexing structure NB-Tree [13] for its simplicity and high efficiency. In general, it calculates the Euclidean norm of an input high-dimensional vector and inserts this norm into a B^+-Tree as the key. Based on this, each assembly or subassembly in the library is parsed and the corresponding graph vector is extracted and inserted into the NB-Tree. The whole process described above is shown in Fig. 15.

The indexing process described above can be carried out on all descriptors of the assemblies or subassemblies in the library. Although the subassemblies extracted are all useful subparts of the original models and can support the partial retrieval to some extent, there are still some cases in which the input query is only a segment of the target assembling graph, e.g. the sample shown in Fig. 13, and in these cases, subassemblies are unable to contribute more. During the matching process described above, this is not a problem because the sub-graph isomorphism algorithm will find

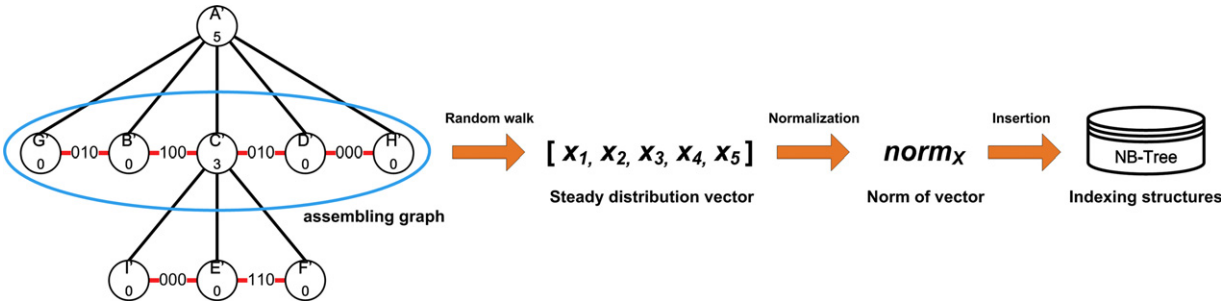
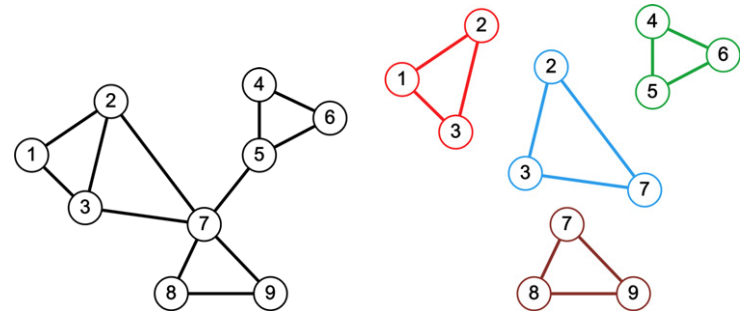


Fig. 15. Indexing steps of an assembly descriptor.



(a) Original graph. (b) Minimum cycle basis.

Fig. 16. A sample graph and its minimum cycle basis.

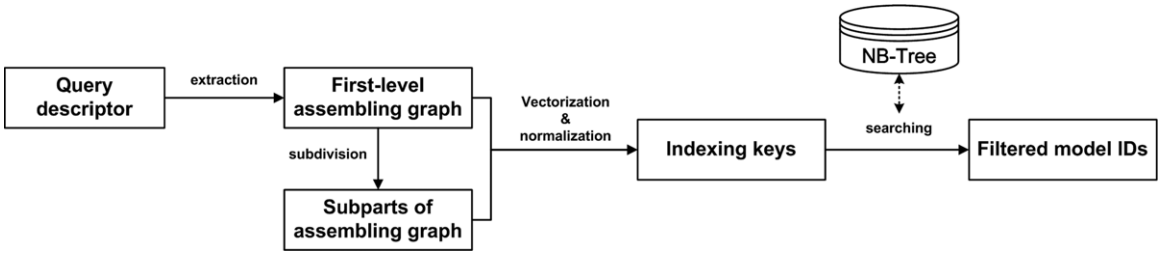


Fig. 17. Flow chart of the filtering process.

all graphs which contain the query graph. However, during the indexing process, it requires a suitable mechanism for indexing such sub-graphs if partial retrieval needs to be well supported. Obviously, indexing of all sub-graphs of an assembling graph is unpractical because of the huge amount of space and time requirements. Hence, we present a sub-graph indexing mechanism in the work to support the partial retrieval better.

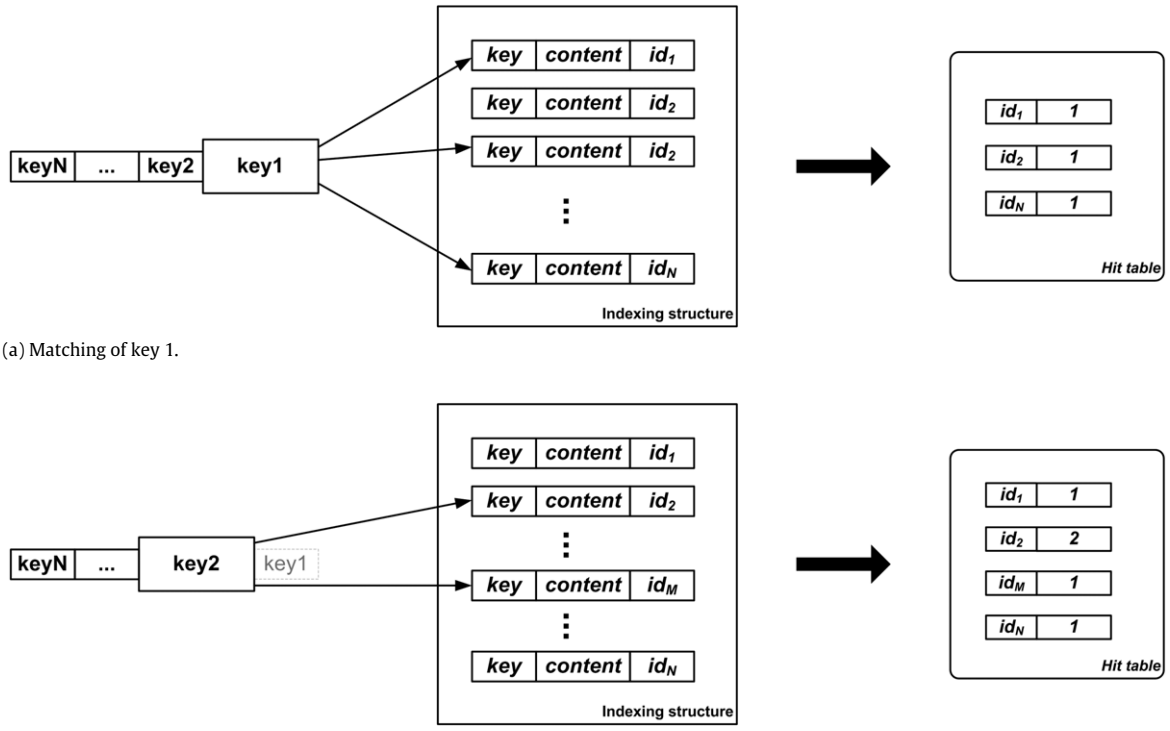
In the sub-graph indexing mechanism, the first-level assembling graph is first divided into some meaningful subparts. Then these subparts are converted to their RW vectors as mentioned above to be inserted into the NB-Tree structure. Currently, the dividing strategy is based on a simple heuristic experience: *cycles in the assembling graph may often represent atomic-subparts of the original model*. However, the enumeration of all cycles is also unpractical because the number of cycles can grow exponentially with the number of vertices; hence we choose to calculate the minimum-cycle-basis (MCB) of the assembling graph. For the details about the MCB finding algorithm used here, one can be referred to [58]. Based on the MCB, the number of cycles found is controlled by the cyclomatic number $(e - v + p)$, where e is the number of edges, v the number of vertices and p the number of connected components of the graph. Fig. 16 shows a sample graph and its MCB.

After the MCB is found, the repetitive cycles ("repetitive" in that the topology and properties of the cycles concerned are all the same) in the MCB are then discarded because the same cycle does not need to be indexed more than twice; otherwise it will increase the responding time of assembly retrieval. An example of repetitive cycle can be found in the first-level assembling graph of Fig. 2. Finally the cycles left without repetition are inserted into the NB-Tree for further filtering of unmatchable assemblies.

6.2. Filtering

During assembly retrieval, a filtering process is executed to exclude the unwanted assemblies based on the indexing structure established beforehand as mentioned in 6.1. The flow chart of the filtering process is shown in Fig. 17. In the process, the first-level assembling graph in the query descriptor is subdivided into several sub-parts, and then the original graph and these sub-parts are all vectorized and normalized in the same way as described in the indexing step. After that, the calculated norms are used as the indexing keys to search in the NB-Tree structure. Finally, the IDs of the database models matched to any of the keys are returned as the filtering result.

There are two different strategies for the subdivision of query graphs: (a) when the query graph is not very large, all cycles of the query graph are found out to generate the indexing keys; (The details of the algorithm for finding all cycles of a graph can be found in [59].) (b) when the query graph is larger than a threshold, only MCB is found on the query graph to generate the indexing keys. However, the filtering accuracy may decrease in this case since some cycles present in the query graph matchable to library graphs



(a) Matching of key 1.

(b) Matching of key 2.

Fig. 18. Searching in the indexing structure.

may be missed in the search occasionally. Fortunately, the query is usually not very complex because of customers' habits during searching.

The process of searching in the indexing structure is shown in Fig. 18. Each indexing key generated from the query is used to match the keys in the indexing structure, and a hit table is maintained at the same time. The hit table records the hit count of assembly IDs in the library. Every time a key-match is found, the hit count of the corresponding id increases by one. After all the indexing keys from the query are searched, the records in the hit table can be sorted by the hit count, as more hits implies more opportunities for the query to be matchable to an assembly. Then the corresponding IDs of the top-N records (The parameter N can be controlled by users.) can be selected for direct feedback or further matching process. All the assemblies without any hit are then discarded.

Although the filtering process may not seem intuitive, it indeed works well in practical assembly retrieval. The reasons behind are these: (a) if the first-level assembling graph of the query is the same with the graphs of library assemblies, they may very likely be matched successfully; (b) if the graph of the query contains one or more cycles of MCB of the library graphs, the query may very likely be a sub-graph of the library graphs.

7. Implementation

7.1. System

The proposed assembly retrieval approach has been implemented in a multi-module prototype system. The UI module (Fig. 19) is developed by using Microsoft Visual C# 2008, while the core module for the matching of descriptors (based on the VFLIB [60]), similarity assessment, indexing and filtering is developed by using Microsoft Visual C++ 2008, which is built as a win32 library invoked by the UI module during retrieval. Besides the two main modules, a C++/CLR module is developed as the translator to deal with the interoperability between them. Moreover, Mysql5.0 is used as the database system and GraphViz2.26.3 is used for visualization of graphs.

Currently, our model library for retrieval contains 2249 parameterized assembly models and 10 062 part models (SolidWorks2009 type) downloaded from the engineering models repository [61] on the web. Fig. 20 shows a portion of the assembly models in the library. Meanwhile, an assembly descriptor generator is written by Microsoft Visual C# 2008 as a plug-in tool to interact with SolidWorks2009 system for parsing models and generating the assembly descriptors.

7.2. Search samples

As is mentioned above, high-level topology could be directly used to search assemblies in our approach. Fig. 21 shows two sample results obtained with the kinematics-graph query. The first query is a structure with six revolute-pairs concatenated together, and the five top results shown are 6-DOF mechanical-hand models. The second query is a human-like structure with spherical-pairs as the interfaces between different parts, and the six top results shown are different human-like creatures.

We also make an application of our assembly retrieval approach to top-down assembly design [62]. During the top-down assembly design, skeletons, which are preliminary structures with some rough child components connected together through assembly interfaces, are often designed first to conduct the following design works. Here, an individual module is used for parsing the skeleton-based query, and assemblies similar to the query are retrieved by the system, which could be further reused to facilitate the following design works. Fig. 22 shows some sample queries and the corresponding top results (similarity $>80\%$) retrieved.

It can be seen from the search results that the retrieved assemblies possess assembly interfaces with different geometric details (Fig. 23(a)). The upper one is a dove-tail connection while the bottom one is a compound pin-hole connection. However,

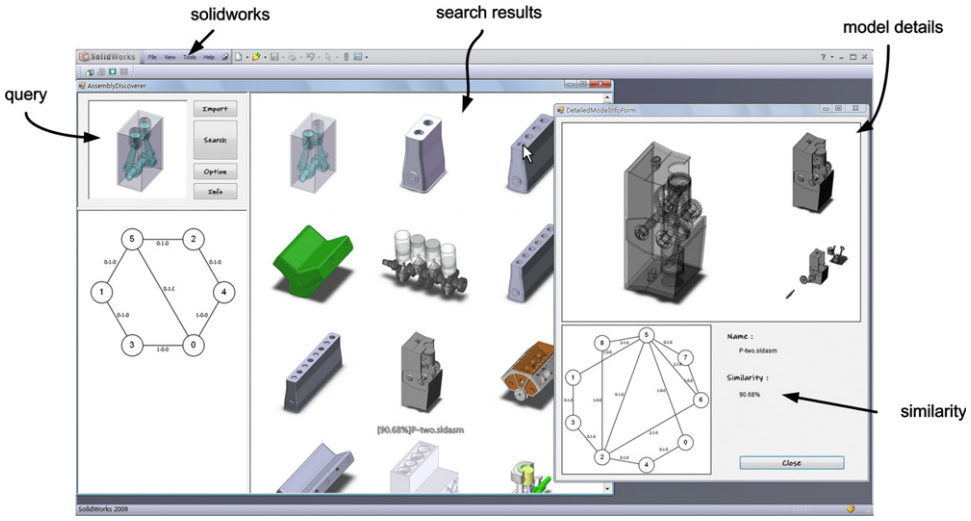


Fig. 19. User interface of the flexible assembly retrieval system.

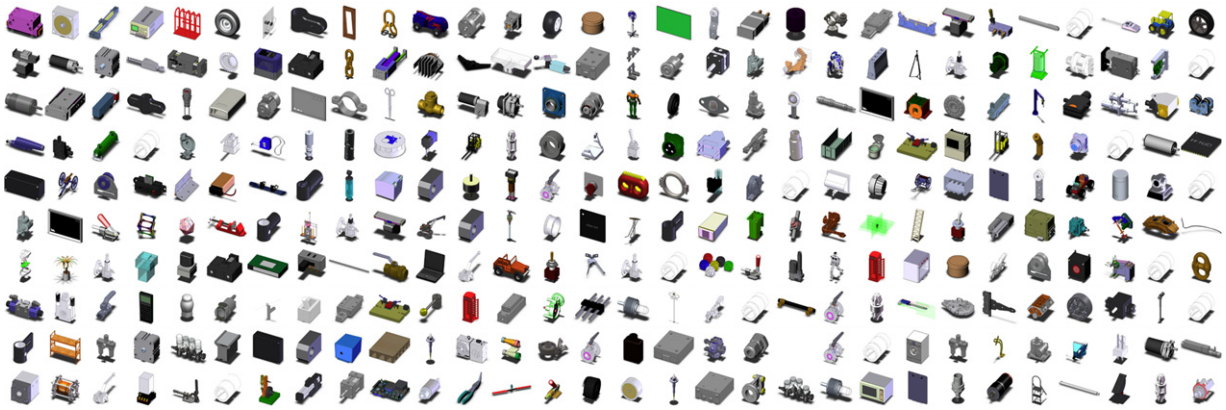


Fig. 20. Some assemblies in the model library for retrieval.

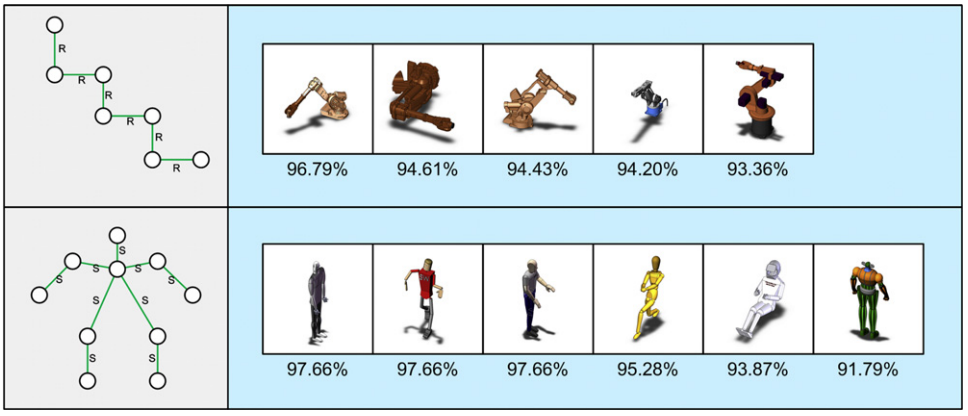


Fig. 21. Search samples of the assembly retrieval system (kinematics-graph query).

the two assembly interfaces both possess one translational DOF and one prismatic pair (sliding). In other words, these two assembly interfaces have the same kinematic characteristics (both coincide with the query) but different geometric-matings. This case can demonstrate how semantic assembly interfaces act on the assembly retrieval, while low-level differences in assemblies do not exclude them from retrieval results but affect the similarities instead.

Fig. 23(b) shows the search details that demonstrate the effect of using the hierarchical assembly structure. The connecting-rod in the skeleton-based query is a single component while the corresponding component in the retrieved engine assembly is a sub-assembly. If hierarchical structure is not used, the assembly interface for the rod sub-assembly between the "upper-rod" and the "end-cap" may be defined externally. As a result, the engine assembly would be abandoned during the assembly retrieval.

The support of partial retrieval can be given from two aspects: one is that the retrieved assembly is a sub-assembly of a bigger one, e.g. the results of the sample search for engine skeleton in Fig. 22 are subassemblies of complex truck models; the other is

Fig. 22. Search samples of the assembly retrieval system (skeleton query).

that the assembling graph structures of search results contain the query structure as a segment which is shown in Fig. 23(c). (Query engine is a two-cylinder design, while the retrieved one is an eight-cylinder design.)

The geometric information is useful for reordering the search results. This can help users find the models they want much more quickly in the result. For example, the 3rd result and the 4th result in the engine sample have identical topology structures, but the layout of the 4th result is *V*-style and hence it is less similar to the query. Moreover, the 2nd result and the 3rd result in the car sample also have identical topology structures; however the shape of the mainframe part in the 2nd result is more similar to the corresponding one in the car skeleton query so that it is put before the 3rd one.

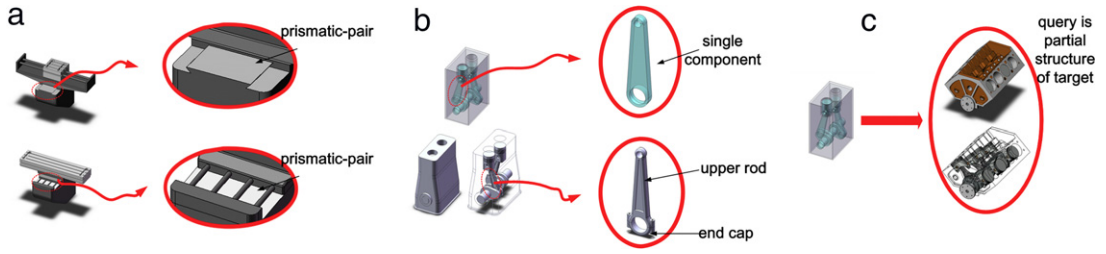


Fig. 23. Details of the search results.

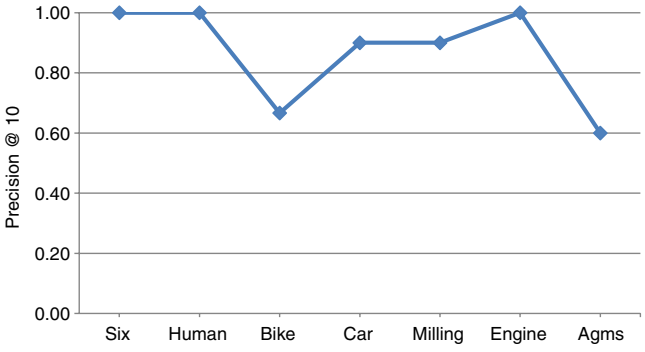


Fig. 24. Precision of top 10 assembly retrieval results.

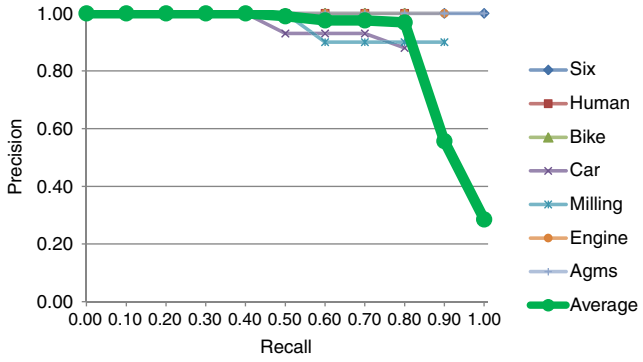


Fig. 25. Precision/recall graph of assembly retrieval results.

As we can see, Figs. 21 and 22 show two different ways for assembly retrieval. Compared to the skeleton based query, the kinematics-graph query is simpler and more convenient to be sketched out, but the similarity result is coarser and less-discriminative. Hence it can be a good way for quick search. On the other hand, the skeleton based query provides more comprehensive information, and the result is subtler. Therefore, it can help users locate the models they want more precisely and quickly in a large database. Besides these two ways, there could be many other flexible means for searching assemblies which are applicable to specific situations and needs, and the corresponding implementation will not be hard since the multilevel assembly descriptor exhibits the necessary abilities.

7.3. Performance evaluation

In order to evaluate the proposed assembly retrieval methods, two graduate students are invited from Mechanical Engineering Department. They are asked to label all the assemblies in the library either as relevant or irrelevant to the queries. Discussion is permitted between the two subjects to achieve agreement, and the final labeled data are used as the ground truth for evaluations below. The same queries shown in Figs. 21 and 22 are used here. In these two figures, some retrieved results are annotated with little red crosses on the bottom-right, which indicate the irrelevance to the corresponding queries.

Fig. 24 gives the precision of top 10 search results for each query. In the case that the retrieved results are less than 10, the number of retrieved results is used in the calculation instead. By looking into the details of Fig. 24, it is clear that every search gets very good accuracy (>0.85) except for the queries "Bike" and "Agms". The precision of these two queries are 0.67 and 0.60 respectively, which are barely satisfactory. However, after an analysis of the corresponding queries, we find that the result is largely due to the fact that the amount of all relevant models for these two queries is less than 10. This means that the top 10 precisions for these queries are actually the final precisions. In that way, we think these precisions are fairly acceptable.

In Fig. 25, the precision–recall graph is given. Each query's interpolated precision–recall curve is shown with thin lines in different colors, and the sampled precision and recall values are also plotted as different point labels. Besides that, the thickest green curve in the figure is the averaged 11-point interpolated precision–recall curve of all 7 queries. It could be seen that the main part (recall <0.8) of the average curve starting from the top-left point is nearly horizontal, which means that the assembly search maintains the precision at a relatively high value when the number of recalled relevant models increases. This can actually be foreseeable since the multilevel assembly descriptor and the matching algorithm used in the work utilize information possessing high distinguishability. On the other hand, the average curve near the tail begins to fall (when the recall value is bigger than 0.8), until arriving at the final value of precision around 0.3. Here the calculation style of the average precision–recall curve is the main reason. As could be noticed, some precision–recall curves cannot get to the points where the recall value is 1, which indicates not all of the relevant models returned in these searches. Accordingly, value 0 (precision) is used instead in the average calculation at the above points, hence the decrease of the average precision. By checking the details of the relevant models which are not retrieved, we find that they are more or less in an abnormal status with regard to the organization of assembly structures (since most of the models are downloaded from public repositories in the internet, we cannot control this irrationality). For example, an upper rod and a piston are combined into a sub-assembly in an engine model, which is not the way present in other engine models. However, this may imply a problem that some models could have comprehensions in other than the most common ways. Therefore, how to resolve these potential differences in essentially identical models may be an interesting topic in assembly retrieval.

The ranking orders in the search results are determined by the calculated similarities. Actually, the absolute values of the similarities are not important, but the relative orders between different models are crucial. Therefore, the two subjects are also asked to adjust the orders of all the search results shown in Figs. 21 and 22 for order evaluation. Then the original orders set by our system are compared with the orders corrected by the subjects. Fig. 26 shows the result of the comparison in the form of a relative order matrix similar to the similarity matrix defined in [49]. In the

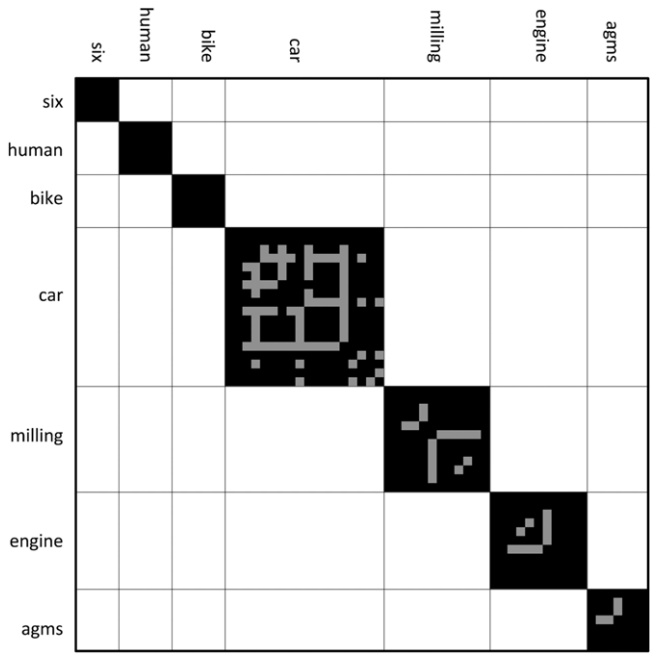


Fig. 26. Relative order matrix for assembly retrieval results.

similarity matrix, the color of each pixel indicates the similarity between two models corresponding to the row and column indices, whereas, in the relative order matrix shown here, the color of each pixel indicates whether the relative order between two models given by our system is correct or not (black is true, gray is false, white is undefined since the two models are not retrieved by the same query). It could be seen that there are generally much fewer gray pixels than black ones in the matrix diagonal, which indicates that the ranking orders given by the system are in a reasonable scope. Moreover, it could be also noticed that the number of gray pixels belonging to the "car" query is obviously greater than that to the other queries. Here the reason behind is mainly related to the number of search results. When abundant models are present, even a professional designer may not feel it easy to determine a comprehensive order among them. Therefore, the relatively higher rate of wrong orders in the "car" block is not a totally unexpected result.

Generally speaking, the absence of a good benchmark dataset is still a main challenge for all related research on assembly retrieval, especially for the evaluations of methods. We will continue to explore the possibility of establishing a more reasonable and comprehensive dataset for evaluation based on the current library we use.

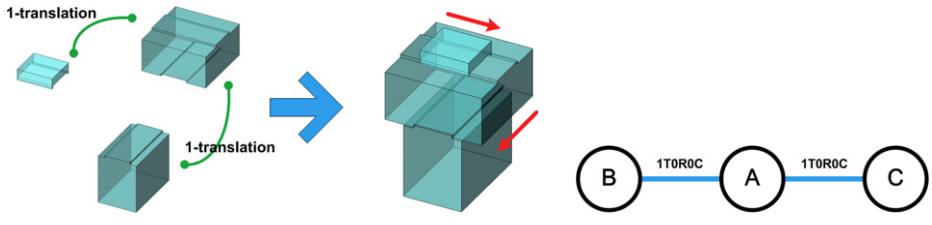
7.4. Efficiency

The statistics about the running time of the assembly retrieval system is shown in Table 2. The test is executed on a PC with Intel Core2 Quad CPU Q9400 (2.66 GHz) and 4 GB memories. Each value in the table is the average time of 100 running under the same configuration. It could be seen that the performance is quite pleasant, while the most time-consuming search (car skeleton query) is 164.23 ms without indexing mechanism. However, the search time could still become high if the size of the library becomes larger, e.g. 10^6 assemblies. In this case, the indexing mechanism will play an important role for accelerating the assembly retrieval, since the search time with indexing in the table shows obvious advancement compared to the search time without indexing.

Table 2
Running time statistics of assembly retrieval.

Query	Search time (in milliseconds)	
	Without indexing	With indexing
Six-DOF graph	39.87	4.46
Human-like graph	47.51	4.95
Bike skeleton	41.05	5.40
Car skeleton	164.23	12.72
Milling machine skeleton	56.39	3.55
Engine skeleton	26.68	8.80
Agms skeleton	105.31	3.01

As regards the details of Table 2, there is an interesting phenomenon which merits our attention. Specifically, while the search for the "engine skeleton" query takes 26.68 ms without indexing and 8.8 ms with indexing, the search for "milling machine skeleton" query, which takes 56.39 ms without indexing (i.e. 2 times longer as compared to engine skeleton), takes only 3.55 ms with indexing (i.e. less than 1/2 times as compared to engine skeleton). In our assembly retrieval method, the most time-consuming step is the hierarchical graph matching. Hence, the number of graph matching operations and the specific structures involved in each of them actually determine the whole search time. In case of the "milling machine skeleton" query, the assembling-graph structure has 3 nodes and 2 edges, each edge representing a 1-translational DOF between the two nodes connected to it (Fig. 27). This structure is quite ordinary so that many models in our assembly library can contain it. Moreover, a model containing this structure can even have it at many different places, which means that there will be a lot of match candidates for graph matching (costing more time). On the other hand, the "engine skeleton" has a relatively more complex structure (Fig. 2) as compared to the "milling machine skeleton", which can be contained by much fewer models (several times difference in the statistical data) in our assembly library. These facts can thus explain the reason why the search for "milling machine skeleton" costs more time than the search for "engine skeleton". However, when the indexing mechanisms presented in the work are utilized, things become quite different. As is known, the models involved in the graph matching are those which cannot be filtered out by the indexing mechanisms. Since the sub-graph indexing mechanism in the work only deals with the cycle structures currently, the graph structures containing the non-cycle query structure as sub-graphs will all be discarded in the filtering process. This in turn greatly reduces the



(a) Milling machine skeleton. (b) Assembling-graph structure.

Fig. 27. Milling machine skeleton query and its assembling-graph structure.

Table 3
Types of translational freedom (L for line, P for plane, v for vector and p for point).

Type	Description	Geometrical element
T0	No translation allowed	$\varnothing$
T1	Translation in one direction	$L(v_0, p_0)$
T2	Translation in a plane	$P(n_0, p)$
T3	Free translation	$L(v, p)$

Table 4
Types of rotational freedom.

Type	Description	Geometrical element
R0	No rotation allowed	$\varnothing$
R1	Rotation about a given axis	$L(v_0, p_0)$
R1-a	Rotation about any axis in a given direction	$L(v_0, p)$
Rf	Rotation about any axis through a given point	$L(v, p_0)$
Rf-a	Rotation about any axis	$L(v, p)$

number of models which need further graph matching in search of the "milling machine skeleton", and hence diminishes its search time to less than half of that for the "engine skeleton". In fact, the current sub-graph indexing mechanism as discussed above may generate some false negatives which could potentially be matched to the query. Therefore, it is very important to find more useful structural patterns in assembly models besides the cycle structure used in our sub-graph indexing mechanism, which will be our future work.

8. Conclusion and future works

In this paper, a flexible and effective approach is presented for searching assemblies in the product library. The multilevel assembly descriptor gathers different levels of information important for distinguishing assemblies. The hierarchical assembly structure and the semantic assembly interface can well preserve the implicit high-level design semantics hidden in assembly models. Moreover, the layout information and the shape information are also kept for better discriminability. Based on the multilevel assembly descriptor, a corresponding matching algorithm is also designed to compare different assemblies and calculate the similarity between them. In addition, an efficient indexing mechanism is presented to accelerate the assembly retrieval, which can also support partial retrieval to some extent. The whole assembly retrieval approach is a flexible and unified framework: users can provide different queries as they wish, such as the rough query and the partial query, and according to different queries, appropriate information will be used for the calculations in the search. Finally, a prototype system is presented to verify our assembly retrieval approach, which shows good performance and efficiency. Meanwhile, an application of the assembly retrieval approach to top-down assembly design demonstrates a promising future for it.

In the future, several things could be done to improve the assembly retrieval approach presented in this paper:

(a) The search samples given in the implementation section show that there are still some occasional models unwanted by the queries. To further improve the precision of assembly retrieval, the similarity calculation should be carefully modified, and some additional semantic information such as the function and loads could be involved.
(b) The current indexing mechanism does not handle all the subparts of an assembly, hence extra efforts should still be put into finding the useful patterns of important and meaningful subparts in assembly models to help implementing better indexing mechanisms.
(c) Reasonable identification of the interface-parts should be addressed to minimize their negative impact on matching.

Acknowledgment

The authors are very grateful to the financial support from the National Science Foundation of China (No. 61173125).

Appendix

See Tables 3–8.

Table 5
Mapping between typical geometric-matings and types of freedom.

Mating	Geometry	Translational element	Rotational element	Types of freedom
Coincident	Point–point	$\varnothing$	$L(v, p_0)$	(T0, Rf)
	Line–line	$L(v_0, p_0)$	$L(v_0, p_0)$	(T1, R1)
	Plane–plane	$P(n_0, p)$	$L(n_0, p)$	(T2, R1-a)
	Point–line	$L(v_0, p_0)$	$L(v, p_0)$	(T1, Rf)
	Point–plane	$P(n_0, p_0)$	$L(v, p_0)$	(T2, Rf)
	Line–plane	$P(n_0, p_0)$	$L(v_0, p_0)$, $L(n_0, p)$	(T2, (R1, R1-a))
Concentric	Cylinder–cylinder	$L(v_0, p_0)$	$L(v_0, p_0)$	(T1, R1)
	Cone–cone	$\varnothing$	$L(v_0, p_0)$	(T0, R1)
Distance	Point–point	$\varnothing$	$L(v, p_0)$, $L(v, p_0)$	(T0, (Rf, Rf))
	Point–line	$L(v_0, p_0)$	$L(v_0, p_0)$, $L(v, p_0)$	(T1, (R1, Rf))
		$P(n_0, p)$	$L(v_0, p_0)$,$L(v_0, p_0)$	(T2, (R1, R1))
	Line–line	$L(v_0, p_0)$	$L(v_0, p_0)$	(T1, R1)
	Point–plane	$P(n_0, p)$	$L(n_0, p)$, $L(v, p_0)$	(T2, (R1-a, Rf))
	Plane–plane	$P(n_0, p)$	$L(n_0, p)$	(T2, R1-a)
	Line–plane	$P(n_0, p)$	$L(v_0, p_0)$, $L(n_0, p)$	(T2, (R1, R1-a))
Tangent	Plane–cylinder	$P(n_0, p)$	$L(v_0, p_0)$, $L(n_0, p)$	(T2, (R1, R1-a))
	Cylinder–cylinder	$L(v_0, p_0)$	$L(v_0, p_0)$, $L(v_0, p_0)$	(T1, (R1, R1))
	Plane–sphere	$P(n_0, p)$	$L(n_0, p)$, $L(v, p_0)$	(T2, (R1-a, Rf))
Angle	Line–line	$L(v, p_0)$	$L(v_0, p)$, $L(v_0, p)$	(T3, (R1-a, R1-a))
	Line–plane	$L(v, p_0)$	$L(v_0, p)$, $L(n_0, p)$	(T3, (R1-a, R1-a))
	Plane–plane	$L(v, p_0)$	$L(n_0, p)$, $L(n_0, p)$	(T3, (R1-a, R1-a))
Parallel	Line–line	$L(v, p_0)$	$L(v_0, p)$	(T3, R1-a)
	Line–plane	$L(v, p_0)$	$L(v_0, p)$, $L(n_0, p)$	(T3, (R1-a, R1-a))
	Plane–plane	$L(v, p_0)$	$L(n_0, p)$	(T3, R1-a)
Perpendicular	Line–line	$L(v, p_0)$	$L(v_0, p)$, $L(v_0, p)$	(T3, (R1-a, R1-a))
	Line–plane	$L(v, p_0)$	$L(n_0, p)$	(T3, R1-a)
	Plane–plane	$L(v, p_0)$	$L(n_0, p)$, $L(n_0, p)$	(T3, (R1-a, R1-a))

Table 6
Mapping between typical kinematic-pairs and types of freedom.

Kinematic-pairs	Translational element	Rotational element	Types of freedom
Planar pair	$P(n_0, p)$	$L(n_0, p)$	(T2, R1-a)
Prismatic pair	$L(v_0, p_0)$	∅	(T1, R0)
Cylindrical pair	$L(v_0, p_0)$	$L(v_0, p_0)$	(T1, R1)
Revolute pair	∅	$L(v_0, p_0)$	(T0, R1)
Spherical pair	∅	$L(v, p_0)$	(T0, Rf)
Rigid pair	∅	∅	(T0, R0)

Table 7
Reducing rules of translational freedom.

Type 1	Type 2	Reducing result of translational freedom
T0	Tx	T0
T3	Tx	Tx
T1	T1	If the directions are the same, or exactly opposite, result is T1. Else, result is T0.
T1	T2	If the T1 direction lies on the T2 plane, result is T1. Else, result is T0.
T2	T2	If the plane normals are the same, result is T2 Else, result is T1, where the T1 direction is given by the intersection of the two planes.

Table 8
Reducing rules of rotational freedom.

Type 1	Type 2	Reducing result of rotational freedom
R0	Rx	R0
Rf-a	Rx	Rx
R1	R1	If the two axes have the same direction, and if the origin point of one axis lies on the other axis, then the result is R1. Else, the result is R0.
R1	R1-a	If the two axes have the same direction, then the result is R1. Else, the result is R0.
R1	Rf	If the Rf origin is on the R1 axis, then the result is R1. Else, the result is R0.
R1-a	R1-a	If the two axes have the same direction, then the result is R1-a. Else, the result is R0.
R1-a	Rf	R1
Rf	Rf	If the two origin points coincide, the result is Rf. Else the result is R1, with the axis joining the two origin points.

References

[1] Llewelyn AI. Review of CAD/CAM. Computer-Aided Design 1989;21:297–302.
[2] Gunn T. The mechanization of design and manufacturing. Scientific American 1982;247:114–30.
[3] Deshmukh AS, Banerjee AG, Gupta SK, Sriram RD. Content-based assembly search: a step towards assembly reuse. Computer-Aided Design 2008;40:244–61.
[4] Smeulders AWM, Worring M, Santini S, Gupta A, Jain R. Content-based image retrieval at the end of the early years. IEEE Transactions on Pattern Analysis and Machine Intelligence 2000;22:1349–80.
[5] Tangelder J, Veltkamp R. A survey of content based 3D shape retrieval methods. Multimedia Tools and Applications 2008;39:441–71.
[6] Iyer N, Jayanti S, Lou K, Kalyanaraman Y, Ramani K. Three-dimensional shape searching: state-of-the-art review and future trends. Computer-Aided Design 2005;37:509–30.
[7] You CF, Tsai YL. 3D solid model retrieval for engineering reuse based on local feature correspondence. The International Journal of Advanced Manufacturing Technology 2010;46:649–61.
[8] Chang S, Perry B, Rosenfeld A. Content-based multimedia information access. Kluwer Press; 1999.
[9] Rui Y, Huang T, Chang S. Image retrieval: current techniques, promising directions, and open issues. Journal of Visual Communication and Image Representation 1999;10:39–62.
[10] Park J, Um B. A new approach to similarity retrieval of 2-D graphic objects based on dominant shapes. Pattern Recognition Letters 1999;20:591–616.
[11] Leung W, Chen T. User-independent retrieval of free-form hand-drawn sketches. In: IEEE international conference on acoustics speech and signal processing. 2002.
[12] Leung W, Chen T. Hierarchical matching for retrieval of hand-drawn sketches. In: Proceedings of the IEEE international conference on multimedia and exposition. 2002.
[13] Fonseca M, Jorge J. Towards content-based retrieval of technical drawings through high-dimensional indexing. Computers & Graphics 2003;27:61–9.
[14] Pu J, Ramani K. On visual similarity based 2D drawing retrieval. Computer-Aided Design 2006;38:249–59.
[15] Hou S, Ramani K. Classifier combination for sketch-based 3D part retrieval. Computers & Graphics 2007;31:598–609.
[16] Hou S, Ramani K. Structure-oriented contour representation and matching for engineering shapes. Computer-Aided Design 2008;40:94–108.
[17] Fonseca M, Ferreira A, Jorge J. Sketch-based retrieval of complex drawings using hierarchical topology and geometry. Computer-Aided Design 2009.
[18] Cardone A, Gupta S, Karnik M. A survey of shape similarity assessment algorithms for product design and manufacturing applications. Journal of Computing and Information Science in Engineering 2003;3:109.
[19] Bimbo A, Pala P. Content-based retrieval of 3D models. ACM Transactions on Multimedia Computing, Communications, and Applications 2006;2:20–43.
[20] Cardone A, Gupta SK, Deshmukh A, Karnik M. Machining feature-based similarity assessment algorithms for prismatic machined parts. Computer-Aided Design 2006;38:954–72.
[21] Funkhouser T, Shilane P. Partial matching of 3D shapes with priority-driven search. In: Proceedings of the fourth eurographics symposium on geometry processing. Eurographics Association; 2006. p. 142.
[22] Gao W, Gao SM, Liu YS, Bai J, Hu BK. Multiresolutional similarity assessment and retrieval of solid models based on DBMS. Computer-Aided Design 2006;38:985–1001.
[23] Kuo C, Cheng S. 3D model retrieval using principal plane analysis and dynamic programming. Pattern Recognition 2007;40:742–55.
[24] Li M, Zhang YF, Fuh JYH, Qiu ZM. Toward effective mechanical design reuse: CAD model retrieval based on general and partial shapes. Journal of Mechanical Design 2009;131:8.
[25] Bai J, Gao S, Tang W, Liu Y, Guo S. Design reuse oriented partial retrieval of CAD models. Computer-Aided Design 2010.
[26] Hu B, Liu Y, Gao S, Sun R, Xian C. Parallel relevance feedback for 3D model retrieval based on fast weighted-center particle swarm optimization. Pattern Recognition 2010.
[27] Aamodt A, Plaza E. Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications 1994;7:39–59.
[28] Wood W. Case-based conceptual design information server for concurrent engineering. Computer-Aided Design 1996;28:361–9.
[29] Kim G. Case-based design for assembly. Computer-Aided Design 1997;29:497–506.
[30] Hu L, Xu C, Wang Y, Liu G. Mechanical product case representation and case retrieval based on object-oriented technique. Journal of Nanjing University of Science and Technology (Natural Science) 2009.
[31] Wu SF, Wang ZY, Pang LL. Rapid design platform for mechanical products based on CBR. Advanced Materials Research 2010;102:262–6.
[32] Chao YS, Liu HJ. Case retrieval in body-in-white parts based on similarities of welding and assembly process. Computer Integrated Manufacturing Systems 2011;17:30–6.
[33] Maher ML, Gomez de Silva Garza A. Case-based reasoning in design. IEEE Expert 1997;12:34–41.
[34] Goel A, Craw S. Design, innovation and case-based reasoning. The Knowledge Engineering Review 2006;20:271–6.
[35] Regli W, Cicirello V. Managing digital libraries for computer-aided design. Computer Aided Design 2000;32:119–32.
[36] Deshmukh A, Gupta S, Karnik M, Sriram R. A system for performing content-based searches on a database of mechanical assemblies. In: ASME international mechanical engineering congress & exposition; 2005.
[37] Gupta SK, Cardone A, Deshmukh A. Content-based search techniques for searching CAD databases. Computer-Aided Design and Applications 2006;3:811–9.
[38] Ullmann J. An algorithm for subgraph isomorphism. Journal of the ACM 1976;23:31–42.

[39] Gaag A, Kohn A, Lindemann U. Function-based solution retrieval and semantic search in mechanical engineering; 2009.
[40] Chakrabarty S, Chougule R, Lesperance RM. Ontology-guided knowledge retrieval in an automobile assembly environment. The International Journal of Advanced Manufacturing Technology 2009;44:1237–49.
[41] Eriksen EP, Moffitt ME, Warren TM. Retrieval of bottom hole assembly during casing while drilling operations. In: Google patents; 2010.
[42] Kim BC, Mun D, Han S. Retrieval of CAD model data based on Web Services for collaborative product development in a distributed environment. The International Journal of Advanced Manufacturing Technology 2010;50: 1085–99.
[43] Lee K, Gossard DC. A hierarchical data structure for representing assemblies: part 1. Computer-Aided Design 1985;17:15–9.
[44] Mäntylä M. A modeling system for top-down design of assembled products. IBM Journal of Research and Development 1990;34:636–59.
[45] Shah J, Mäntylä M. Parametric and feature-based CAD/CAM: concepts, techniques, and applications. Wiley-Interscience; 1995.
[46] Ambrósio JAC, Eberhard P. Advanced design of mechanical systems: from analysis to optimization. Springer Verlag; 2009.
[47] Molian MS. Storage and retrieval of description of mechanisms and mechanical devices according to kinematic type. Journal of Mechanisms 1969;4:311–23.
[48] Chiou SJ, Sridhar K. Automated conceptual design of mechanisms. Mechanism and Machine Theory 1999;34:467–95.
[49] Osada R, Funkhouser T, Chazelle B, Dobkin D. Shape distributions. ACM Transactions on Graphics 2002;21:807–32.
[50] Ip C, Lapadat D, Sieger L, Regli W. Using shape distributions to compare solid models. In: Proceedings of the seventh ACM symposium on solid modeling and applications. ACM; 2002. p. 273–80.
[51] Xiaoning G. Technical report: a kinematics analysis and simulation system for the complex virtual prototyping under CAVE; 2005.
[52] Turner JU, Subramaniam S, Gupta S. Constraint representation and reduction in assembly modeling and analysis. IEEE Transactions on Robotics and Automation 1992;8:741–50.
[53] Cordella L, Foggia P, Sansone C, Vento M. A (sub) graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 2004;1367–72.
[54] Conte D, Foggia P, Sansone C, Vento M. Thirty years of graph matching in pattern recognition. International Journal of Pattern Recognition and Artificial Intelligence 2004;18:265–98.
[55] Chung FRK. Spectral graph theory. Amer. Mathematical Society; 1997.
[56] Gori M, Maggini M, Sarti L. Exact and approximate graph matching using random walks. IEEE Transactions on Pattern Analysis and Machine Intelligence 2005;27:1100–11.
[57] Böhm C, Berchtold S, Keim D. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Computing Surveys 2001;33:322–73.
[58] Horton JD. A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM Journal on Computing 1987;16:358.
[59] Johnson DB. Finding all the elementary circuits of a directed graph. SIAM Journal on Computing 1975;4:77–84.
[60] http://amalfi.dis.unina.it/graph/.
[61] http://www.3dcontentcentral.com.
[62] Chen X, Gao S, Yang Y, Zhang S. Multi-level assembly model for top-down design of mechanical products. Computer-Aided Design 2011; doi:10.1016/j.cad.2010.12.008.