Interface-Based Search and Automatic Reassembly of CAD Models for Database Expansion and Model Reuse

Lucas Vergez *, Arnaud Polette, Jean-Philippe Pernot

Arts et Métiers Institute of Technology, LISPEN, HESAM Université, F-13617 Aix-en-Provence, France

ARTICLE INFO

Dataset link: https://zenodo.org/record/7628235

Keywords:
CAD assembly
Part retrieval and reuse
Collision-free solver
Reassembly strategies
Database expansion

ABSTRACT

This paper introduces a new framework for reassembling CAD models of mechanical parts. The generated CAD assemblies are well-constrained, with collision-free parts, and they are ready-to-use for downstream applications. First, input dead CAD models candidate for the reassembly are sorted following a part-by-part interface-based identification algorithm that is capable of characterizing each part according to the assembly slots and interfaces it offers. The slots are then hashed, and the resulting keys are used for fast search of parts in the database. Thus, the parts that can be assembled are quickly identified, and their assembly can be considered according to various scenarios. To support the reassembly steps and satisfy the constraints associated with the specified interfaces, a collision-free kinematic constraint solver is also proposed. During the reassembly steps, the geometry of the slots can also be automatically modified to adjust their dimensions, in order to ensure a perfect fit and to avoid interference at the level of the interfaces. The resulting database can also be further expanded while modifying the relative positions/orientations of the assembled parts. The approach is illustrated on several test cases covering different exploitation scenarios, ranging from simple model reuse to database expansion for machine learning-based applications. Such an automatic reassembly process is particularly innovative, and it clearly paves the way for smart assembly procedures.

1. Introduction

For several years, databases of CAD models have been designed and populated to meet various application needs. For instance, it enables CAD models to be reconstructed more quickly from point clouds [1] while directly updating available parameterized CAD models [2]. But it is also very much interesting for standardization purposes, or to avoid starting a new design from scratch and reuse as much as possible existing and already optimized solutions. Until now, these developments were mainly guided by the need for greater efficiency to become more competitive, but they are now also dictated by a more global context where energy consumption and CO_2 emissions must be better controlled, thus further constraining the way growing digital resources are to be solicited. This trend has consequently created a need to develop efficient search engines capable of finding models according to multiple criteria, both for the search of parts and assemblies [3]. However, much attention has been paid to how models can be retrieved from these databases, but less to how the retrieved models can be reused for downstream applications.

Artificial intelligence has also played an important role in increasing the attractiveness of learning issues in CAD, and consequently the need for large databases of CAD models. For instance, the numerical methods used to automate the reverse engineering process are increasingly based on machine learning techniques and are now capable of not only detecting objects in point clouds, but also estimating their principal dimensions to automatically reconstruct CAD models [4]. If these learning methods are increasingly used, they are most often limited by the labeled datasets, which remain scarce for mechanical assemblies. To overcome the lack of labeled data, synthetic data generation methods have become popular recently, although the concept dates back to the 1990s. This phenomenon has been particularly accentuated with the popularization of new generative algorithms that can produce quality synthetic data on a larger scale [5]. However, automatically generating reusable CAD models remains challenging. Despite the high level of activity in the field of automatic CAD model generation, the subject remains relevant, and there are still few methods to automate the process of reassembling mechanical parts. Indeed, the assembly of mechanical parts is frequently based on engineering rules, and an intelligent assembly of parts remains complex to implement because of the diversity of possible solutions and constraints to be respected to generate reusable CAD models. For example, the LEGO shape reproduction challenge remains an open optimization problem to this day [6], and this issue is much less complex than the assembly of

* Corresponding author.
E-mail address: lucas.vergez@ensam.eu (L. Vergez).

https://doi.org/10.1016/j.cad.2023.103630
Received 14 June 2023; Received in revised form 13 September 2023; Accepted 8 October 2023
Available online 28 October 2023

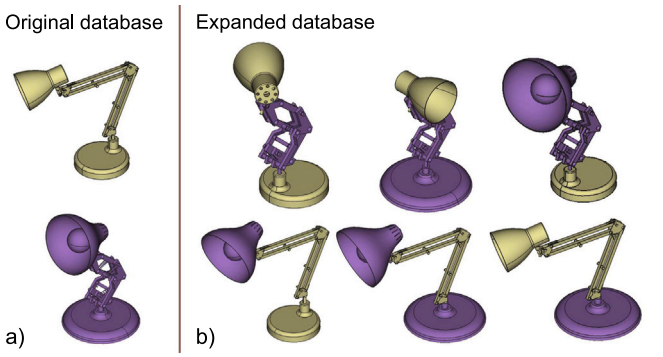


Fig. 1. Example of two original dead CAD models (a) that have been mixed to generate new well-constrained, collision-free and ready-to-use CAD assemblies (b). The colors indicate the source models. The constraints added between the parts enable their relative positions/orientations to be modified, thus further expanding the variety of shapes and configurations in the database. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

any mechanical part, for which there is a great diversity of possible interfaces between all parts. This is the problem tackled in this paper.

This paper introduces a new framework for reassembling CAD models according to different strategies and goals, ranging from expanding CAD databases for machine-learning applications, to generating CAD assemblies from assembly graphs in order to support early stages of the design process, or to automatically retrieving parts to connect disconnected assemblies. The input dead CAD models come from open CAD assembly databases in standard formats (e.g. STEP files) and they undergo several treatments. First, all possible mates between parts of the assemblies are automatically identified and used to characterize the available slots on each part, which then become candidate interfaces for the reassembly step. The slots are hashed, and the resulting keys are used for part-by-part interface-based search of parts in the databases. Thus, the parts that can be assembled are quickly identified, and their assembly can be considered according to various scenarios. The scenarios are intended to cover several use cases used to validate the robustness of the proposed framework and to show the diversity of accessible solutions. To support the reassembly step and satisfy the constraints associated to the specified interfaces, a collision-free kinematic constraint solver has been developed. Fig. 1 shows examples of CAD assemblies automatically generated from existing CAD models using the proposed approach. The generated CAD assemblies are well-constrained, with collision-free parts, and they are ready-to-use for downstream applications. The resulting database can also be further expanded while modifying the relative positions/orientations of the assembled parts. Importantly, during the reassembly steps, the geometry of the slots can be automatically modified to adjust their dimensions, in order to ensure a perfect fit and to avoid interference at the level of the interfaces. Such an automatic reassembly process is particularly innovative, and it clearly paves the way for smart assembly procedures.

In summary, the contribution is fourfold: (i) a new framework capable of reassembling CAD models according to different scenarios, and that generates well-constrained and collision-free assemblies ready-to-use for downstream applications; (ii) an ad-hoc hashing and encoding strategy that exploits multiscale descriptors to allow fast search of mechanical parts and their interfaces, and to facilitate the reassembly; (iii) new operators to modify the geometry of the slots candidate to the reassembly to adjust their dimensions and ensure a perfect fit; (iv) a new method for collision-free resolution of 3D kinematic constraints.

The paper is organized as follows. Section 2 reviews the state-of-the-art in this domain. The proposed reassembly framework is then introduced and the details of the algorithms are provided in Section 3. Section 4 presents and analyzes the results obtained following different reassembly scenarios. The last section concludes the paper and introduces future works.

Table 1
CAD databases comparison.

Database	Free	Size	Ass.	Label	Type	Cons.
3DCC	Yes	NC	Yes	No	STP	Yes
AD Gallery	No	72 841	Yes	No	STP	Yes
TraceParts	No	109 M.	No	Pos.	STP	No
3Dfindit	No	1 B.	Yes	No	STP	No
PartComm.	Yes	21 906	Yes	No	STP	No
Fischer	Yes	NC	No	No	STP	No
GrabCAD	Yes	2 M.	Yes	No	STP	No
Zhizaoyun	Yes	NC	Yes	No	STP	No
ABC	Yes	1 M.	No	Pos.	STP	No
Fusion360	Yes	8251	Yes	No	STP	Yes
ShapeNet	Yes	0.5 M.	Yes	Yes	STL	No

2. Related work

While the literature on 3D CAD part generation is vast, that on assembly generation is much less so. Actually, to tackle the issue of reassembling mechanical parts, several aspects are to be considered, among which the availability and characteristics of the existing databases, how CAD assemblies should be described for their efficient search in large databases and notably their interfaces, how mechanical parts should be assembled and the corresponding simulation methods, and how kinematic constraints should be resolved so that assembled parts do not collide.

CAD databases. CAD model databases have emerged over the last years, to save time and avoid starting from scratch, or to allow the development of machine learning-based approaches requiring the provision of large databases. Several types of CAD databases can be distinguished:

- Databases directly available from CAD software development companies (3D ContentCentral©, Autodesk Gallery [7]), and which are generally made available to the CAD software licensees. They contain plenty of models, all designed on the corresponding platforms.
- Databases with CAD models of manufacturers (TraceParts, 3Dfindit, PARTcommunity, Fischer) make it possible to list all the manufacturers' models through a single interface. The parts are often very specific and not very diverse, with redundant models varying only in size or brand. The way CAD models are simplified also varies greatly, which can affect the way interfaces between parts are described.
- Large-scale open access databases (GrabCAD, Zhizaoyun) are the largest free databases available, they list very diversified CAD models and are not attached to any manufacturer/software. Contributions can come from anyone, and there is no validation process, so there is no certainty about the quality of uploaded models, making them difficult to reuse and exploit.
- Large databases designed for research purposes (ABC-Dataset [8], Fusion360GalleryDataset [7], ShapeNet [9]) are among the largest labeled databases, and are intended to develop new algorithms and notably machine learning-based approaches.

To compare these databases, several criteria are adopted: the cost of the access to the models to distinguish freely accessible models from others, the number of available files to characterize the size of the database (M. Million, B. Billion), if the database contains assemblies, if the parts or assemblies are labeled and here a possible labeling (Pos.) is noted when a sorting has already been done in order to classify the models in precise categories, if the database contains STEP files (STP) or mesh files (STL), and finally if the assemblies are constrained.

The results of the comparison are summarized in Table 1 and show that there is a lack of methods to reassemble CAD models and generate enriched assemblies containing assembly constraints needed

for model reuse. Indeed, the only labeled databases are the ones from the manufacturers and containing little diversity (e.g. TraceParts), or databases that do not contain CAD assemblies (e.g. ShapeNet). Thus, a method for expanding CAD assembly databases would generate large CAD assembly databases from a few labeled samples.

Description of assemblies. In their survey, Lupinetti et al. [3] provide a good overview of the methods used to describe assemblies to be retrieved from CAD databases. In order to search in a database for mechanical parts with similar interfaces, it is necessary to be able to describe the interfaces between the parts. These interfaces can be retrieved by geometric methods in complex non-constrained assemblies [10]. In this context, the classification of parts is done using shape descriptors, coupled with assembly and sub-assembly recognition algorithms. The assembly descriptors of mechanical parts must be based on multiple criteria [11] to be able to find a part by its geometry, its interfaces in the sub-assembly, and the role of the sub-assembly in the assembly. There are then different topological descriptors of geometry that can be used, such as EMD [12], adjacency graph similarity [13,14], semantic similarity [15] or vector distribution [16]. These descriptors are then integrated one level higher in linkage graphs [11] or adjacency matrices [17] to compare assemblies or sub-assemblies with each other. These methods allow to find parts, sub-assemblies or assemblies in large databases using multi-level descriptors [18]. However, these methods are time-consuming because so many assembly descriptors need to be compared at different levels. It is therefore necessary to find a more compact way of describing and encoding assemblies and their interfaces to enable fast and efficient comparison and retrieval in large databases. Moreover, the problem when looking for parts to be reassembled is slightly different. First, the parts must complement each other at the level of their interfaces, and thus the search for parts in the database must be guided by complementary topological shapes. Second, it is only necessary to search for similar interfaces without all the geometry being similar, in order to generate diversity in the generated assemblies.

Assembly generation. Determining the assembly sequences of mechanical parts is an active research field [19]. These algorithms allow determining whether an assembly of parts is feasible in practice. Here, starting from a known assembly, the difficulty relies on determining an assembly sequence with a minimal number of steps. This makes it possible to reconstitute the steps to be followed during the assembly. Some recent methods tackle the sequence problem by disassembly without collisions [20], but the available solutions can hardly be directly transposed in a reassembly context where the final assembly is not known beforehand. Actually, the assembly of mechanical parts can be assimilated to the assembly of puzzle pieces such as Jigsaw puzzles [21], whose resolution is most often based on probability analysis. Their reassembly strategy allows understanding the iterative assembly of a new piece on several others. These methods have been generalized in 3D [22] in order to reconstitute objects from their fragments. They explore the choice of parts to assemble at a given time in an ongoing assembly. However, here also the parts of the assembly are known in advance, their number is fixed, and all the links are comparable to embedded links, which greatly facilitates the possibilities of inserting new parts in the assembly. Similarly, the question of LEGO construction is to find the optimal way to convert 3D polygonal mesh data into LEGO representation, given the number of LEGO bricks. The connections between the bricks are always embedded, and the bricks are assimilated to voxels, which also greatly reduces the possibilities. Here again, this is not the case in the context of this paper as the parts to be reassembled are not always known in advance.

Some publications propose methods for assembling mechanical parts. These methods can be probabilistic [23], or tile-based [24] and focus on the semantics of the parts, but not on the mechanical joints. These mechanical constraints are fixed, they concern only meshes, and do not allow kinematic degree of freedom in the assembly. Those that take mechanical constraints into account remain manual [25]. They can be based on templates but offer only a help for designers with well parameterized models as inputs [26]. Finally, some recent methods suggest reassembly strategies based on neural networks [7,27]. However, these methods do not provide complete assemblies but only the way two unknown parts should fit together, which is the same as predicting the kinematic constraint between two CAD models. Differently from Willis et al. [7], the work presented in this paper does not currently predict the kinematic constraints between two unknown parts. Indeed, the proposed approach enables parts to be reassembled according to the interfaces to which they were already linked in previous assemblies of the database. Finally, only one very recent work allow managing constraint cycles, which are very common in classical mechanical assemblies [28]. Even if it is promising for automatic assembly generation, it is however limited to point clouds.

Kinematic solvers. To allow kinematic simulations, an assembly must not contain collisions between parts. Over the last decades, many algorithms have been developed to solve problems defined by 3D kinematic constraints. Graph-based approaches and algebraic methods are the most commonly used to solve geometric constraint problems [29–31], and are dominant in 2D CAD applications. They have also been extended, more recently, to 3D cases where the management of constraints and the search for solutions are more complex. From a 2D CAD point of view, D-Cubed's algebraic approach, called Dimensional Constraint Manager, is in fact an industry standard in the field of constraint-based drafting. The more recent 3D version of this software, 3D DCM, based on a fast non-sequential solver, is used to constrain parts in assemblies and mechanisms. The FreeCAD open-source software community has created its own A2+ kinematic solver [32], which solves constraints iteratively, by adding attractive forces between constrained parts. This physical analogy provides a better understanding of the way parts are assembled. It should be noted that there are many solvers that can find a kinematic configuration which satisfies the placement of one or more parts. These are called Inverse Kinematic Solvers [33] and are most often used to determine the exact placement of a robotic arm during its movement, which corresponds to configurations where there is no constraint cycle. Moreover, to the best of our knowledge and even if there exist functionalities to detect collisions between positioned CAD parts [34], there is no kinematic constraint solver capable of handling these collisions during the kinematic resolution, in order to generate collision-free solutions. This is one of the contribution in this paper.

As a conclusion, this paper intends to overcome the limitations of the existing methods, with a fast part-by-part interface-based search and reassembly algorithm to allow the automatic generation of collision-free CAD assembly models, used for database expansion and model reuse. The generated assemblies are well-constrained, allowing to further expand the database while modifying the relative positions/orientations of the assembled parts. Here, it is important to stress that the approach proposed in this paper focuses on the reassembly of B-Rep CAD models defined by surfaces that have not been tessellated, and possibly available from STEP files. During the reassembly steps, the geometry of the interfaces can also be automatically modified to adjust their dimensions and ensure a perfect fit.

3. Reassembly framework

3.1. Overall framework

The proposed reassembly framework is composed of three relatively independent phases that can be run in parallel once initiated (Fig. 2):

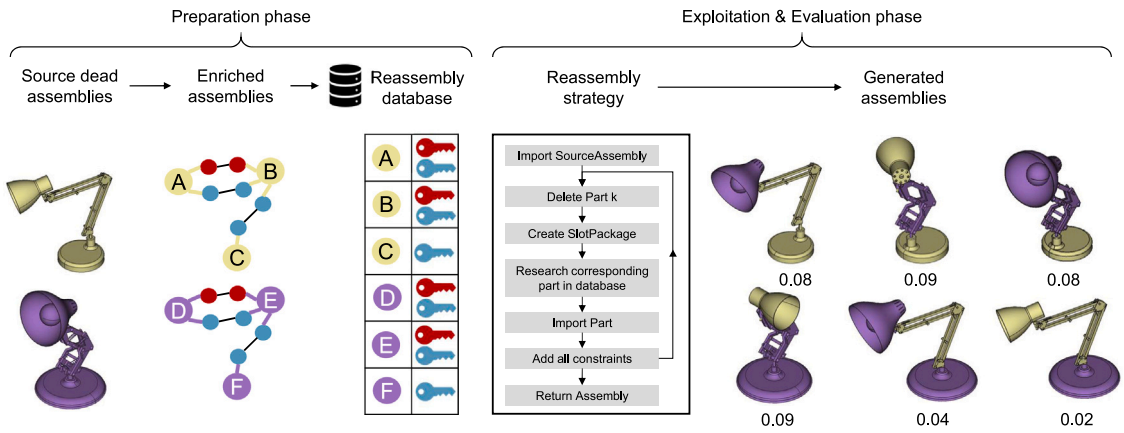


Fig. 2. Overall framework for part-by-part interface-based search and reassembly of CAD models. During the preparation phase (left), starting from dead assemblies, the algorithm first detects and hashes the possible slots and slot packages used to reassemble the input parts. The obtained enriched assemblies are stored with their search keys in the database for reuse in a reassembly strategy. In the exploitation phase (right), a reassembly is carried out while efficiently identifying candidate parts of the database, and the relevance of the reconstructions can then be assessed according to different metrics during the evaluation phase (e.g. CLIP scores in this case).

- *Preparation phase*: Source dead assemblies are first collected from existing repositories, e.g. GrabCAD [35], and enriched while detecting all possible mates between pairs of faces (Section 3.2). Each mate corresponds to a so-called slot that is then used during the exploitation phase to host another part whose slot matches. At this level, an interface between parts is defined by several slots, and it has a type depending on the involved slots. Redundant kinematic constraints are simplified to reduce complexity during the resolution. Hashes of these slots are then computed to be able to search for parts quickly during the reassembly (Section 3.3). All this information is stored in a database designed to allow fast identification of parts candidate to the reassembly.
- *Exploitation phase*: It corresponds to the use of the created database as input of several part-by-part reassembly strategies presented in Section 4. The database is then analyzed to identify candidate parts for reassembly with matching interfaces and slots. Kinematic constraints are added between the selected parts that can also undergo a modification step (Section 3.4) before starting the resolution using a newly developed collision-free kinematic constraint solver (Section 3.5).
- *Evaluation phase*: The generated CAD assemblies are finally evaluated using metrics (Section 4), including their CLIP scores to sort them by visual appearance [36].

3.2. Kinematic constraint detection and simplification (Preparation phase)

As part of the preparation phase, the first step of the reassembly algorithm consists in recovering the kinematic constraints of the source assemblies collected from existing repositories. Indeed, CAD models are often available as dead B-Rep models stored in STEP files and free of kinematic constraints. Thus, this step allows reassembling the parts in the same way as they were originally assembled, with possibly slightly different constraints than the original ones as there is no uniqueness. Redundant kinematic constraints are also simplified to reduce complexity during the resolution. This information makes it possible to get rid of a prediction of the joints between two parts [7].

3.2.1. Kinematic constraint detection

The kinematic constraint detection algorithm allows the detection of mates between two surfaces. The analysis of the Autodesk Gallery Dataset [7] shows that contacts between surfaces represent 88% of all possible kinematic constraints. Among them, planar and cylindrical contacts represent 83% of all contacts. Thus, in this paper, only these two categories are considered, even though the method is generic and remains applicable for any type of mate.

The detection of planar and cylindrical contacts is made from conditions between two faces of two B-Rep models of the assembly, following engineering rules. A pre-processing is carried out on the faces of the assembly in order to limit as much as possible the number of pairs of faces to be checked during the constraint detection. Indeed, the number of functional faces in an assembly is very low (i.e. less than 12% of all models in the Autodesk Gallery Dataset). A face refinement operation is thus carried out beforehand on all the parts of the assembly in order to group all the faces belonging to the same super-maximum face [37]. This notion of super maximal face allows grouping faces which belong to the same topological domain in the same set. This is for instance the case for cylinders, which are often defined by two half-cylinders. Once this pre-processing is done, the pairs of faces whose bounding boxes overlap and which belong to two different solids are further studied.

A face f is a geometric entity that owns several attributes. Each face f has a type t_f defining the type of the surface, i.e. either T-PLANE or T-CYLINDER in this paper. Planar surfaces are defined by their unit normal n_f and plane point p_f. Cylindrical surfaces are defined by their unit axis a_f, a center c_f lying on this axis, a radius r_f and an orientation o_f that defines the side of the material (either O-FORWARD or O-REVERSED) when considering solid models.

The following four conditions are to be satisfied to define a cylindrical contact between two faces f_1 and f_2:

1. Both faces are cylindrical: $t_{f_1} = t_{f_2}$ = T-CYLINDER
2. The orientations of the two faces are opposite: $o_{f_1} \neq o_{f_2}$
3. The axes are coaxial: $a_{f_1} \times a_{f_2} = 0$ and $a_{f_1} \times (c_{f_2} - c_{f_1}) = 0$
4. The radius are equal within a tolerance: $|r_{f_1} - r_{f_2}| < \epsilon_r$

Similarly, the following four conditions characterize a planar contact between two faces f_1 and f_2:

1. Both faces are flat: $t_{f_1} = t_{f_2}$ = T-PLANE
2. The normals are collinear: $n_{f_1} \times n_{f_2} = 0$
3. Both normals are in opposite directions: $n_{f_1} = -n_{f_2}$
4. Faces overlap within a tolerance: $|(p_{f_2} - p_{f_1}).n_{f_1}| < \epsilon_d$

When these conditions are met on a pair of faces, a constraint is added to the stack of potential constraints, and once all pairs of faces have been traversed, redundant kinematic constraints between the same two parts are removed.

3.2.2. Kinematic constraint simplification

Figs. 3a and 3b illustrate the redundancy of constraints naturally generated by the method on an academic example. Indeed, three planar

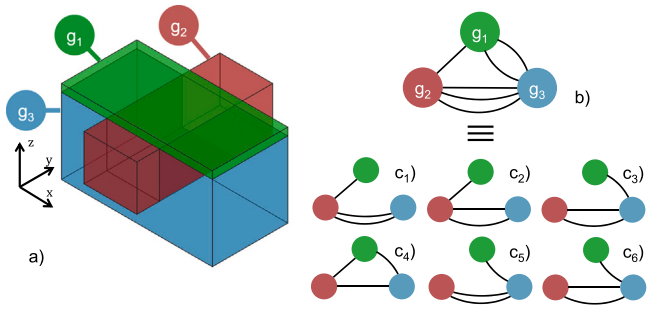


Fig. 3. Analysis and simplification of kinematic constraint redundancy.

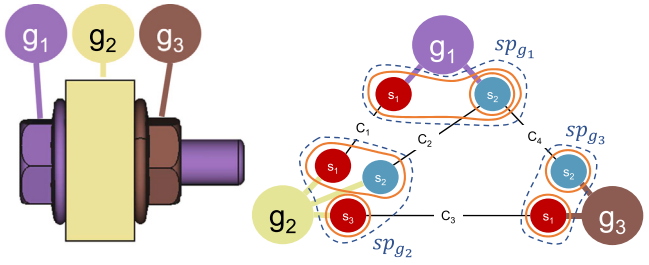


Fig. 4. Simplified example of a screw-plate-nut assembly depicting the geometries g, the slots s (red for planar slots and blue for cylindrical ones) and the slot packages circled in orange. Slot packages of geometries sp_{g_k} are circled in blue. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

contacts out of the six detected are enough to completely constrain the assembly. This redundancy can be detected while checking if the addition of a new constraint removes or not any degree of freedom (DOF) of the considered parts. If the addition does not remove any DOF, then this constraint is defined as implicit, and is not added to the list of constraints. Obviously, there is no unique solution to this problem, and multiple combinations are valid (Fig. 3c$_i$), which makes it difficult to identify the one solution adopted in the original design. To tackle this issue, constraints are compared at the level of kinematic groupings [38]. The larger the kinematic grouping, the greater the diversity of possible solutions. Constraint redundancy can also be detected between two parts. In Fig. 3, the two vertical planar constraints between the blue and the red parts are redundant, and only one planar constraint is sufficient to prevent the displacement along x-axis, thus bottom graphs c_4 to c_6 are completely equivalent. A constraint can then correspond to a set of equivalent kinematic constraints. There is no longer one entity (point, edge, or face) that is constrained with another, but a set of equivalent entities constrained with another set of entities. At the end, only one entity hosts the effective and simplified constraint, despite the implicit constraints of the other entities.

3.3. Slots and slot packages encoding (Preparation phase)

This section introduces the notions of slot and slot package, as well as the way the related information is encoded to allow fast search in the database.

3.3.1. Geometries, slots and kinematic constraints

An assembly a is composed of a set of geometries G and a set of kinematic constraints C, and it is defined as:

$$a = (G, C) \tag{1}$$

In this paper, all geometries are considered at the same level of the assembly and there is no sub-assembly. This is because sub-assemblies often have a semantic meaning, and that the proposed automatic assembly generation process does not currently allow managing the semantic meaning of sub-assemblies. However, a sub-assembly can still be considered as a single geometry to be constrained with other geometries, but it will remain frozen, thus preventing any subsequent disassembling or animation. Fig. 4 shows a simplified example of a screw-plate-nut assembly with its associated linkage multi-graph. Here, for sake of clarity, only one plate g_2 is considered. The screw has planar and cylindrical constraints with the plate, the plate has a planar constraint with the nut, and the nut has a cylindrical constraint with the screw. A geometry of the assembly a is noted g_i, and is defined by:

$$g_i = \left(S_i, P_i, \tilde{P}_i\right), \text{ with } i \in [1..N_g] \tag{2}$$

where N_g is the number of geometries in a, S_i is the set of slots available in g_i, P_i are the modifiable parameters of all the slots in S_i, and $\tilde{P}_i$ the unmodifiable ones. Following the definition of Kalogerakis et al. [23], a slot is a set of equivalent entities of the same geometry that can be used to support a kinematic constraint. Two elements are said to be equivalent with respect to a constraint when it is equivalent to apply the constraint on any of the equivalent entities. A slot thus contains two lists of parameters, the ones which can be modified and the ones that cannot:

$$s^i_\kappa = \left(E^{s,i}_\kappa, P^{s,i}_\kappa, \tilde{P}^{s,i}_\kappa\right), \text{ with } \kappa \in [1..N^i_s] \tag{3}$$

where $E^{s,i}_\kappa$ is the list of equivalent entities, $P^{s,i}_\kappa$ and $\tilde{P}^{s,i}_\kappa$ are respectively the list of modifiable and unmodifiable parameters, and N^i_s the number of slots stored in the list S_i of the geometry g_i. The parameters of a slot can be different depending on the type of the slot. Indeed, parameters in T-CYLINDER slot are: type, orientation, radius, reference axis (ra) and reference point (rp). Parameters in a T-PLANE slot are: type, reference axis (ra) and reference point (rp). Thus, the parameters of the slots can be related to either the structure of the interface, or to its dimensions. Modifiable ($\tilde{P}_i$) and unmodifiable (P_i) parameters are defined according to the context, knowing that increasing the number of modifiable parameters widens the space of reassembly solutions. This is because the modifiable parameters of a slot can be modified to enable the geometry associated with it to adapt the opposite geometry with which it is intended to be assembled. Theoretically, each parameter can belong to one of the two categories. However, even if more flexibility could be given in the future, this paper focuses on practical applications where all the structural parameters of the geometries (i.e. type and orientation) are not modifiable, while all the geometric parameters such as radii can be modified. For instance, a slot characterizing a cylindrical interface will always remain cylindrical, unlike its diameter that can change to avoid interference when reassembling with another matching cylindrical slot. The unmodifiable parameters corresponding to geometric descriptors (e.g. T-CYLINDER or O-FORWARD) are converted to numerical values to be encoded in the hash, whereas modifiable parameters are not hashed. The way the parameters can be modified during the reassembly phase is discussed in Section 3.4. Moreover, a slot can be leader or not, depending on whether its parameters serve as references for all slots assembled in the same kinematic group, and there can be only one slot leader per kinematic group.

A kinematic constraint of an assembly a is denoted $c_{k,i,j}$, and it is defined from a pair of geometric entities $e_{1,i}$ and $e_{2,j}$ belonging to two different geometries g_i and g_j, as well as from the type of constraint $t_k \in T$ = {T-PLANE, T-CYLINDER} between these two elements:

$$c_{k,i,j} = \left(\left\{e_{1,i}, e_{2,j}\right\}, t_k\right), \text{ with } k \in [1..N_c] \text{ and } (i,j) \in [1..N_g]^2 \tag{4}$$

where N_c is the number of constraints in a, and the geometric entities are either faces, or edges, or vertices of the B-Rep models.

3.3.2. Slot packages

In an interface between two parts, several slots may be involved and are to be brought together in a so-called slot package. A slot package sp_κ is an abstract representation of an interface belonging to an assembly. It contains several slots, possibly linked to different geometries of the assembly, as well as the relations between them, and the information is structured in a complete graph with attributes on each node and each edge, and with an edge between every two vertices:

$$sp_\kappa = (S_\kappa, R_\kappa), \text{ with } \kappa \in [1..N_{sp}] \tag{5}$$

where N_{sp} is the number of slot packages in the assembly, S_κ is the set of slots in the slot package, and R_κ is the set of edges between the slots to build a complete graph. Each edge represents a relative position between two slots, as calculated with Eq. (8). Similarly, sp_{g_i} represents the slot package containing all slots of a geometry g_i in its original assembly.

3.3.3. Hash function and encoding strategy

An interface encoding strategy has been developed to be able to search quickly the parts of the reassembly database that can be assembled with a given slot package. Therefore, descriptors representing similar and complementary slot packages must be computed to be used as keys for finding parts that can be reassembled. This encoding must be similar for all slot packages with similar slots in terms of unmodifiable parameters, and with relative placements between slots that must be similar. First, the encoding of a slot s can be defined with a key K_s such that:

$$K_s = \text{truncate}(\text{sha512}(\tilde{P}^s), 5) \tag{6}$$

where $\tilde{P}^s$ are the unmodifiable parameters of s, and sha512 the adopted hash function [39]. Thus, two slots defined by the same unmodifiable parameters will be characterized by the same key. Here, to avoid problems with rounding errors, values are rounded with 2 digits. Similarly, a complementary slot, i.e. a slot that can be assembled with a slot s, is characterized by its complementary key $\overline{K}_s$ such that:

$$\overline{K_s} = \text{truncate}(\text{sha512}(\text{comp}(\tilde{P}^s)), 5) \tag{7}$$

where the 'comp' function returns the complementary parameters of the input parameters. For example, the complementary orientation of O-REVERSED is the orientation O-FORWARD, and the complementary type of T-CYLINDER is also the type T-CYLINDER. From this definition, a slot key can be associated to each slot of a slot package, and consequently to each node of the slot package graph.

Then, each edge contains as attribute a hash $K_{i,j}$ of the geometric relation $R(s_i, s_j)$ between two slots s_i and s_j, which are part of the same slot package. The geometric relation is computed differently depending on the types of the slots, but it will always be represented by an 'angle', a 'distance', and 'type' of relation:

$$\begin{cases} K_{i,j} = K_{j,i} = \text{sha512}(R(s_i, s_j)) \\ R(s_i, s_j) = \big(\text{angle}(s_i, s_j), \text{distance}(s_i, s_j), \text{type}(s_i, s_j)\big) \end{cases} \tag{8}$$

Considering that $\text{card}(T) = 2$, with types T-PLANE and T-CYLINDER in this paper, there are three cases:

- If s_i and s_j are T-CYLINDER slots:

$$\begin{cases} \text{angle}(s_i, s_j) = \widehat{ra_i, ra_j}\ [\pi] \\ \text{distance}(s_i, s_j) = \text{distPoint2Line}(rp_i, rp_j, ra_j) \\ \text{type}(s_i, s_j) = \text{T-CC} \end{cases} \tag{9}$$

- If s_i and s_j are T-PLANE slots:

$$\begin{cases} \text{angle}(s_i, s_j) = \widehat{ra_i, ra_j} \\ \text{distance}(s_i, s_j) = \text{distPoint2Plane}(rp_i, rp_j, ra_j) \\ \text{type}(s_i, s_j) = \text{T-PP} \end{cases} \tag{10}$$

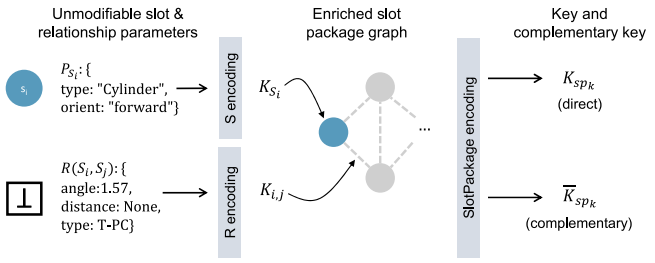


Fig. 5. Slot package encoding process: slots and geometric relationships are encoded in respectively the nodes and edges of the slot package graph, and the key and complementary key associated to this graph are then computed using Weisfeiler–Lehman graph kernels [40].

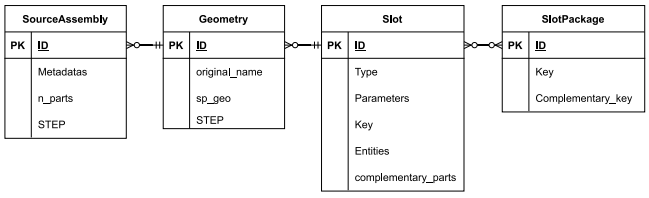


Fig. 6. Entity–relationship diagram of the SQL reassembly database.

- If s_i and s_j are respectively T-PLANE and T-CYLINDER slots:

$$\begin{cases} \text{angle}(s_i, s_j) = \widehat{ra_i, ra_j}\ [\pi] \\ \text{distance}(s_i, s_j) = None \\ \text{type}(s_i, s_j) = \text{T-PC} \end{cases} \tag{11}$$

where $\text{distPoint2Line}(p_*, p_l, a_l)$ and $\text{distPoint2Plane}(p_*, p_p, a_p)$ compute respectively the projection distance of the point p_* on the line defined by the point p_l and its axis a_l, and of the point p_* on the plane defined by the normal a_p and the point p_p.

Based on these definitions, the key K_{sp_i} and complementary key $\overline{K}_{sp_i}$ of a slot package sp_i can be computed using the Weisfeiler–Lehman graph kernels [40]. This is illustrated on Fig. 5. It allows manipulating the underlying complete graph in a vector space, which is currently the most used method for graph hashing. It also guarantees that two different hashes imply that the two graphs are not isomorphic, and thus that the slot packages are not joinable, i.e. that the geometries cannot be reassembled. This guarantees the contrapositive:

$$sp_i \text{ and } sp_j \text{ joinable} \implies \begin{cases} K_{sp_i} = \overline{K}_{sp_j} \\ \overline{K}_{sp_i} = K_{sp_j} \end{cases} \tag{12}$$

In the case of slot packages, the isomorphism is always verified between two complete graphs as long as they have the same number of nodes. The difference is made on the attributes of each node and each edge of the graph. The attribute of a node corresponds to the key of the corresponding slot, and the attribute of an edge between two nodes corresponds to the key computed while hashing the geometric relation between the two related slots. Thus, the equality of these keys then implies a similarity at the slot package level. The descriptor of a complementary slot package can then be found by replacing in the graph all the encoding of the slots with their complementary encoding.

3.3.4. Reassembly database structure

The database is a relational SQL database designed to be able to find a part, by checking hash equalities of slots or slot packages. The entity–relationship diagram is described in Fig. 6.

A slot package is linked to one or more slots, and a slot can belong to one or more slot packages, so a transition table is added

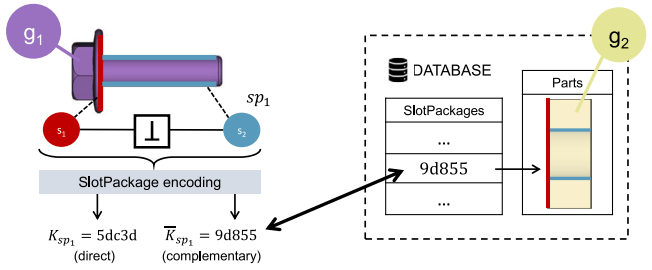


Fig. 7. Search of a screw-complementary part in a database using the complementary key obtained from the slot package encoding of the screw. Keys have been truncated for sake of clarity.

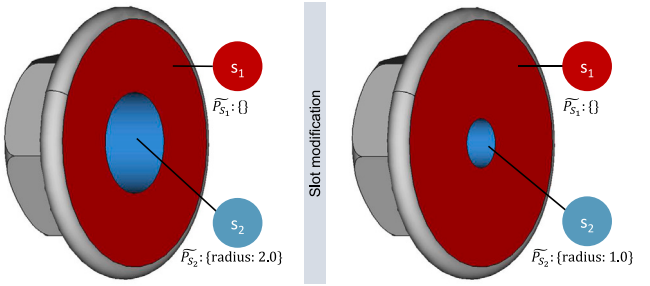


Fig. 8. Modification of a dead CAD model (left) using boolean operations (right), and following the two assumptions as explained in Section 3.4.

implicitly between the slot table and the slot package table. Each object is included once in the object to its left, so adding the primary key of the right object in the left object is enough to represent the relational link between the tables. The slot (or slot packages) is hashed and the resulting hash indicates where the corresponding value is stored through an associative array. Thus, searching for geometries in the database with a condition on slots or slot packages comes to only one hashing constant operation. This is why the execution time of these queries is not directly linked to the size of the database, which allows to be far more efficient than the most recent approaches [3].

3.3.5. *Use of hashes for fitting part search*

The use of hashes to search for candidate geometries for the reassembly is illustrated in the configuration of Fig. 4, and for the two slot packages of g_1 and g_2 containing two slots each. This example is further detailed in Fig. 7 that shows the computational path to find g_2 of the database to be reassembled with g_1. Using the hashes K_{s_1} and K_{s_2} of the slots, and the hash $K_{1,2}$ of the relation between the two slots, the hash K_{sp_1} and complementary hash $\overline{K}_{sp_1}$ of the slot package are computed and used to identify the right slot package and part in the database.

As the comparison of hashes is performed on the slot packages, once the geometries identified, it is still needed to identify the slots to be constrained together. To do this, an isomorphism is applied between the two slot packages to link the joinable slots together. Once the mapping is obtained, constraints can be added between the geometric elements of the slots.

Finally, it can be noticed that there are quite few different slot package keys when considering the adopted Autodesk Gallery Dataset that indeed consists of only 850 different keys over 35,000 slots packages. Thus, the hash conflicts are not studied here considering the small variety of slot packages obtained in the mechanical assemblies. However, this is not crucial, because if two different slot packages have identical hashes, the assembly function described in Section 4.3 will not find an isomorphism between the two slot packages, and the algorithm will proceed without adding any constraints between the two slot packages.

3.4. *Constraint management and modification (Exploitation phase)*

Before the resolution by the collision-free solver, modifiable parameters of the slots (e.g. radius of a hole or cylinder) may need to be adjusted to match and enable a perfect fit without interference between the underlying geometries. Indeed, this is because the function works at the level of the unmodifiable parameters (e.g. orientation of the cylinder or plane). This step is optional and if modifications are not performed, then reassembled models whose modifiable parameters do not match are disregarded. To keep the method valid, some assumptions are made:

1. Modifying a parameter of a slot does not propagate changes to unmodifiable parameters of the geometry. It must also not change the semantic, e.g. the semantic category of the geometry must persist after the modification.
2. The modification function must return a correspondence mapping between the old and new faces of the modified geometry, in order to be able to keep working on the related slots (persistent naming problem).

In this paper, the modification of radius has been implemented using boolean operations. It allows the solution space to be significantly enlarged during reassembly. Indeed, without such a modification, the search for parts in the database would be limited to parts with exactly the same radius. A slot modification performed with our method is shown in Fig. 8.

3.5. *Collision-free resolution of 3D kinematic constraints (Exploitation phase)*

Once the kinematic constraints have been added and simplified (Section 3.2), and once the geometric modifications of the interfaces made (Section 3.4), the resolution can start while ensuring that the geometries do not collide. To the best of our knowledge, there is currently no CAD solver capable of solving kinematic constraints while avoiding collisions between parts. To this aim, a new collision-free kinematic CAD assembly solver has been developed based on the A2+ solver of Braun et al. [32].

3.5.1. *Principle of the A2+ solver*

The A2+ solver is a physics-based iterative algorithm that is the most widely used in the open-source CAD community. To satisfy the kinematic constraints, geometries are submitted to forces and moments, thus inducing translations and rotations of the N_g geometries $g_k \in G$ of the assembly a taking part in the resolution. At an iteration t, the force and moment applied to a geometry g_k are calculated considering the contribution of each constraint $c_i \in C(g_k)$, with $C(g_k)$ the set of constraints in which g_k is involved:

$$\vec{F}_{g_k}(t) = \sum_{c_i \in C(g_k)} \vec{F}_{c_i \to g_k}(t) \tag{13}$$

$$\vec{\Omega}_{g_k}(t) = \sum_{c_i \in C(g_k)} \left\| \mathrm{BBC}(g_k) - \mathrm{FAP}(\vec{F}_{c_i \to g_k}(t)) \right\| . \vec{F}_{c_i \to g_k}(t) \tag{14}$$

where BBC() and FAP() functions return respectively the center of the bounding box of a geometry and the application point of a force. Moreover, the force induced by a constraint on a geometry depends on its type $t_i \in$ {T-PLANE, T-CYLINDER}:

$$\vec{F}_{c_i \to g_k}(t) = \begin{cases} rp(c_i, g_{\overline{k}}) - p_1 & \text{if } t_i = \text{T-CYLINDER} \\ p_2 - rp(c_i, g_k) & \text{if } t_i = \text{T-PLANE} \end{cases} \tag{15}$$

where $rp(c_i, g_k)$ and $rp(c_i, g_{\overline{k}})$ are the reference points of the constraint c_i on respectively g_k and $g_{\overline{k}}$, with $g_{\overline{k}}$ the complementary geometry of g_k

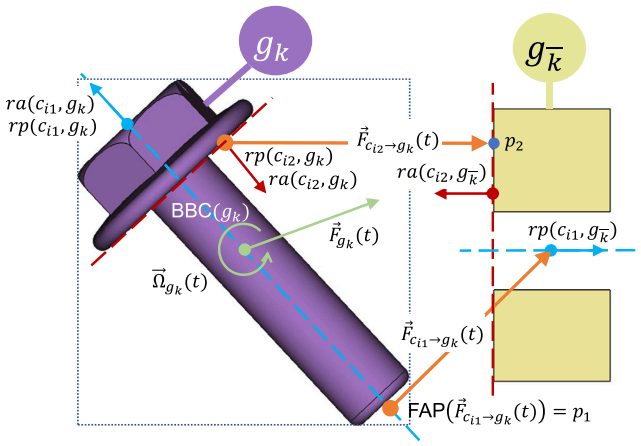


Fig. 9. Forces applied on a screw to be mounted in a plate, and induced by axial and planar constraints c_{i1} and c_{i2} at iteration t. Induced forces are in orange, and total force and moment on the screw are in green (different scale for the sum). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

involved in c_i. Point p_1 is the projection of the reference point $rp(c_i, g_{\overline{k}})$ onto the reference axis of g_k defined by $rp(c_i, g_k)$ and $ra(c_i, g_k)$, and p_2 is the projection of the reference point $rp(c_i, g_k)$ onto the reference plane of $g_{\overline{k}}$ defined by $rp(c_i, g_{\overline{k}})$ and $ra(c_i, g_{\overline{k}})$. This is illustrated in the example of Fig. 9 that describes the forces applied to a geometry g_k (a screw) to be mounted in a complementary geometry $g_{\overline{k}}$ (a plate with a hole). The plate is considered as fixed in this example.

The new placement $P_{g_k}(t+1)$ of each geometry g_k is achieved through a translation and rotation whose amplitude are directly computed from the forces and moments:

$$P_{g_k}(t+1) = P_{g_k}(t) + \left\{ \begin{array}{c} k_F \times \vec{F}_{g_k}(t) \\ k_\Omega \times \vec{\Omega}_{g_k}(t) \end{array} \right\} \tag{16}$$

where the coefficients k_F and k_Ω weight the forces and moments applied to the geometries to allow stabilizing the convergence during the resolution. Their default values are set respectively to 0.5 and 0.1 in the FreeCAD A2+ workbench. Step after step, the amplitude of the forces decreases until they vanish within a tolerance. More precisely, the convergence is observed while following the evolution of the error function:

$$\text{Error}(t) = \max_{\substack{g_k \in G \\ c_i \in C(g_k)}} \left\| \vec{F}_{c_i \to g_k}(t) \right\| \tag{17}$$

Little perturbations are also applied so that the system does not converge to an unstable equilibrium. Indeed, a force can induce a zero moment when the direction of the force passes through the rotation point of the geometry. Moreover, when the system is stable and the error is non-zero, it means that the system is over-constrained, and the resolution is not possible. The redundant constraint must then be removed in order to solve the system. Here, if such an over-constrained reassembly configuration arises, the configuration is simply disregarded and the reassembly algorithm proceeds to the next step.

3.5.2. Collision forces calculation

To extend the capabilities of the solver and allow solutions to be found where parts do not collide, in this paper, new collision forces are added during the resolution. Basically, the idea is to move the constrained assembly to find a configuration that does not have colliding geometries. It is therefore essential to first solve the constraints without collision forces before adding them. Then, at each iteration t a collision force is calculated for each geometry g_k:

$$\vec{F}^{coll}_{g_k}(t) = \sum_{g_i \neq k \in G} \vec{F}^{coll}_{g_i \to g_k}(t) \tag{18}$$

The collision detection itself can be done by different methods. For instance, extreme distance computation is commonly used in OpenCascade to detect collisions, using notably the B-Rep Extrema method. Ray tracing also uses collision detection to compute reflected light on B-Rep models. However, it is highly optimized for the special case of light, and not really adapted to detect a collision between two CAD models. Today, a commonly used solution consists in computing the intersection volume g_{ik} between the two geometries g_i and g_k, and this is the method that has been explored in this paper to compute the collision forces between the geometries. Following the idea that the more the geometries intersect, the more the volume $\text{Vol}(g_{ik})$ of the intersection is, and the more the collision forces should be, the following equations are defined:

$$\vec{F}^{coll}_{g_i \to g_k}(t) = k_C \times \text{Vol}(g_{ik}) \times \frac{\text{BBC}(g_k) - \text{BBC}(g_{ik})}{\left\| \text{BBC}(g_k) - \text{BBC}(g_{ik}) \right\|} \tag{19}$$

where k_C is a coefficient used to weight the amplitude of the collision forces, and which has been empirically tuned to the value 2.0. The direction of the collision force is defined by the unit vector going from the center of the bounding box of g_{ik} to the center of the bounding box of g_k. The application point of the force is the center of the bounding box of the intersection volume:

$$\text{FAP}(\vec{F}^{coll}_{g_k}(t)) = \text{BBC}(g_{ik}) \tag{20}$$

These forces are added to the forces computed from the constraints and explained in the previous subsection, and they are then treated in the same way to calculate the overall forces and moments applied to the geometries. This is visible from the pseudocode of the collision-free solver as is described in algorithm 1.

Algorithm 1 Collision-free solver algorithm

Require: $a, \max_{iter}, \epsilon$
Ensure: a, Solved
  $t \leftarrow 0$
  Solved $\leftarrow None$
  Run $\leftarrow true$
  $\vec{F}^{dist}(0) = \vec{0}$
  while Run $= true$ **do**
    for $k \leftarrow 1, N_g$ **do**
      $\vec{F}_{g_k}(t) = \sum_{c_i \in C(g_k)} \vec{F}_{c_i \to g_k}(t) + \sum_{g_{i \neq k} \in G} \vec{F}^{coll}_{g_i \to g_k}(t) + \vec{F}^{dist}(t)$;
      $P_{g_k}(t+1) = P_{g_k}(t) + \left\{ \begin{array}{c} k_F \times \vec{F}_{g_k}(t) \\ k_\Omega \times \vec{\Omega}_{g_k}(t) \end{array} \right\}$
      Update geometry placement
    end for
    Compute Error
    if Error$(t) \leq \epsilon$ **then**
      Solved $\leftarrow true$ ▷ The system is solved
      Run $\leftarrow false$
    end if
    if $t \geq \max_{iter}$ **then**
      Solved $\leftarrow false$ ▷ The system cannot be solved
      Run $\leftarrow false$
    end if
    if $|\text{Error}(t-1) - \text{Error}(t)| \leq \epsilon$ and $t \neq 0$ **then**
      $\vec{F}^{dist}(t+1) = \vec{\epsilon}_p$ ▷ Random little perturbation
    else
      $\vec{F}^{dist}(t+1) = \vec{0}$
    end if
    $t \leftarrow t+1$
  end while

The behavior of the collision-free kinematic solver is illustrated in the example of Fig. 10 where a screw and a nut are to be assembled to a plate (simplified example with a single plate for sake of clarity). Basically, once assembled, there is no collision between the three

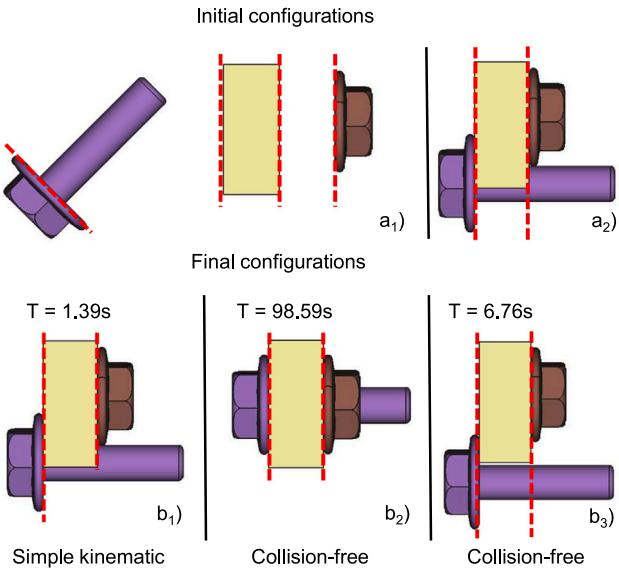


Fig. 10. Examples of use of the collision-free solver on a bolted assembly using only T-PLANE constraints: the top row corresponds to the initial configurations (a_1 for b_1 and b_2, and a_2 for b_3), and the bottom row to the assemblies after resolution ((b_1) simple kinematic resolution, (b_2) collision-free resolution, (b_3) collision-free resolution after a simple kinematic resolution).

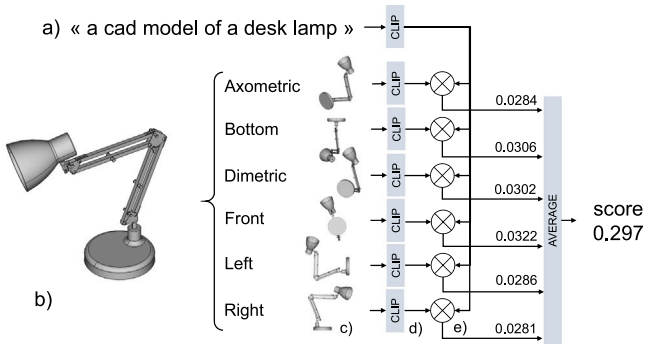


Fig. 11. Example of CLIP-based similarity assessment between the CAD model of a lamp (b), and a reference sentence (a). The comparison is performed according to 6 different views (c). The sentence and the 6 images are independently encoded using CLIP (d), then the resulting latent vectors are compared using the cosine method (e). The average of the 6 distances is finally used as a score to characterize the deviation between the sentence and the reassembled CAD model.

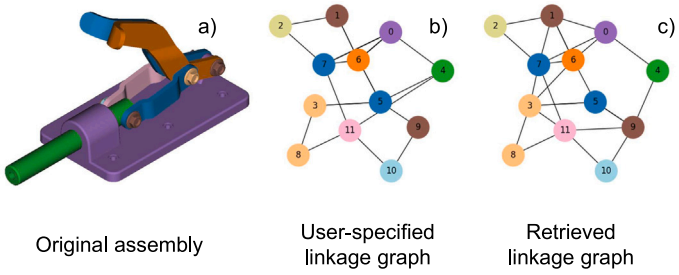


Fig. 12. Example of a test assembly from Autodesk Gallery (a), its user-specified linkage-graph as defined in the original dataset (b), and the one retrieved following the approach proposed in this paper (c).

parts. Thus, to see the difference, only T-PLANE constraints are specified between the screw and the plate on one side, and between the nut and the plate on the other side. There is no T-CYLINDER constraint. The left column (a_1, b_1) shows the result obtained using a simple and fast kinematic resolution without collision detection. The solution is obtained in 1.39s, but the geometries collide. The second column (a_1, b_2) demonstrates the capabilities of the solver when the collision forces are added. Even if no coaxiality constraints are defined, coaxiality is reached and a non-colliding solution is obtained in 98.59 s. The third column (a_2, b_3) shows the result obtained when combining a simple resolution followed by a collision-free resolution. This last resolution is a good tradeoff as it allows to reduce the computation time (1.39 s + 6.76 s), with a non-colliding solution, even if in this case, additional coaxiality constraints should be added to get a well-constrained bolted assembly.

3.6. Visual evaluation by CLIP (Evaluation phase)

The last step of the proposed reassembly framework consists in assessing the overall quality of the generated models. Different metrics could then be imagined to characterize for instance the usefulness, the assemblability, the complexity, or the functioning of a CAD assembly. However, this is highly dependent on the context in which the model is to be used downstream, and this issue has not been considered in this paper. Here, the focus is on assessing the visual appearance of the geometries, comparing their degree of similarity with either descriptors or reference models, using a CLIP-based metric [36]. This is certainly less application-dependent than evaluating the level of usefulness or the functioning of an assembly.

Basically, CLIP is a neural network that encodes images and text in the same vector space. This makes it possible to measure the distance between an image and a text that serves as a descriptor. For example, encoding the word 'cat' and encoding a photo of a cat will produce two latent vectors with a small distance between them. Here, these distances can be useful for checking the visual consistency of generated assemblies. The assessment process is described in Fig. 11. A reference sentence is first transformed in a latent vector using CLIP, and the same applies for 6 views of the 3D reassembled CAD model to be evaluated. Next, the vector of each image is compared with that of the reference sentence, and the set of deviations is then average to obtain an overall assessment of the distance between the CAD model and the reference sentence. The lower the score, the better the reference sentence characterizes well the considered CAD model. In this way, automatically reassembled CAD models can easily be sorted, and more distant models can be disregarded. This is illustrated in the results section.

4. Results and discussion

Different experiments have been performed to validate the automatic reassembly approach for database expansion and model reuse. All of them have been carried out on an Intel® Core™ i9-11950H processor at 2.60 GHz, with 32Go RAM and with disable GPU acceleration.

4.1. Comparing user-specified and automatically identified mating face pairs

The first experiment aims at comparing the results obtained automatically by the mating face identification algorithm proposed in this paper, to the ones considered as specified by a designer. Importantly, the user-specified linkage-graphs are directly available from the considered Autodesk Gallery dataset, and no particular study has been carried out with real designers.

The automatic identification was carried out with a tolerance $\epsilon_d = \epsilon_r = 1$ mm to consider possible clearances between parts. Among the results, the ones obtained on the example of Fig. 12 are representative of most configurations encountered during this experiment. Indeed, here, the proposed approach detects 24 kinematic joints (c), compared to the 26 defined in the original assembly (c) from the dataset. Of the 24

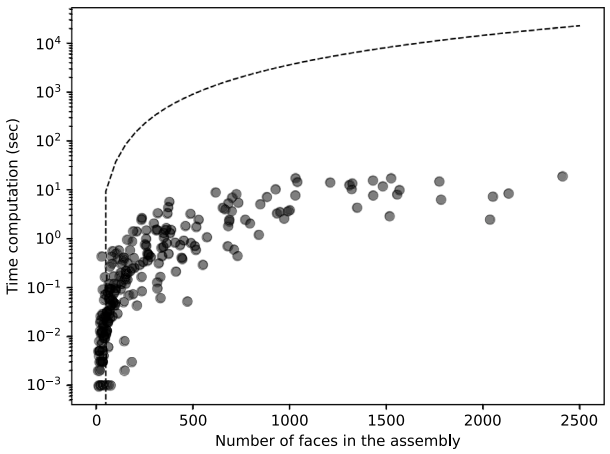


Fig. 13. Face mate computation times as a function of the number of faces present in the first 700 assemblies of the Autodesk Gallery Dataset.

constraints identified, 17 are strictly identical to the ones present in the original assembly. The original assembly has 2 redundant cylindrical mates and 2 redundant toroidal mates (pin joints), while 3 cylindrical mates are missing. Actually, the toroidal mates in the original assembly have been completely modeled by the proposed identification algorithm thanks to the planar and cylindrical mates. Moreover, the constraints resulting from the proposed identification algorithm always refer to actual contacts between two faces of the model, conveying a more physical meaning than a CAD modeler allows. Indeed, using a CAD modeler, there are for instance no particular restrictions on specifying constraints between two axes even if they are not attached to material and surfaces in contact, thus creating purely virtual constraints with respect to physical and technological considerations.

The performance of the automatic mating face identification algorithm can also be analyzed through its capacity to efficiently detect contacts in a large dataset. Fig. 13 depicts face mate computation times as a function of the number of faces present in the assemblies of the Autodesk Gallery dataset. The dot curve represent the theoretical computation time $O(n^2)$, without bounding box pre-filtering. Thus, although the complexity of the algorithm is quadratic, this set of experiments shows that the computation time is much lower in practice.

4.2. Part retrieval

The second experiment assesses the ability of the proposed approach to search for parts in a reasonable time. Table 2 analyzes 4 different queries tested on two databases containing 4 assemblies (total of 155 parts) and 12 assemblies (total of 1118 parts) respectively. The 'Screw' and 'Unique part' queries search for parts that can be assembled in the same way as the provided parts. The 'Unique part' was chosen so that no part is similar in the database, and just itself is retrieved. The 'Screw matching' query searches for a part that has only the complementary slot package of a screw. The 'Planar slot part' query searches for all parts that have a planar slot. Each execution time is an average of response times on 10 similar SQL queries. As a result, one can notice that the response time increases with the size of the database. The impact of database size is less when the query condition is on slots and not on slot packages. This is correlated to the length of the key, which contains 5 characters in the case of slots, and 32 characters in the case of slot packages. Anyway, the execution times are very low compared to the size of the database, and this allows a fast search for components to be reassembled. Indeed, thanks to the developed encoding strategy, the algorithm has no need to loop over all parts to find the matching ones, unlike research strategies developed by Lupinetti et al. [11]. Thus, the time complexity does not depend on the size of the database. Examples of retrieved parts are shown in Fig. 14.

Table 2
Average execution times when querying the reassembly database.

Query slotpackage	Time (ms)	Parts retrieved	Size database
Screw part	37	7	155
Screw matching	43	3	
Unique part	44	1	
Planar slot part	31	147	
Screw part	180	144	1 118
Screw matching	180	26	
Unique part	180	1	
Planar slot part	39	737	

4.3. Automatic CAD assembly generation

The automatic generation of CAD assemblies is a very broad problem, and the way in which mechanical parts can be reassembled together differs according to the final objective. To this aim, three reassembly strategies (ReplacePart, TargetGraph and FillBetweenTwoParts) have been imagined together with their algorithms, but other ones could be set up. These algorithms call functions that are not fully detailed in the paper, due to space limitation, but they are based on elementary functions explained in Section 3.3. For example, the function $Assemble(sp_1, sp_2)$ performs an isomorphism between the two complete graphs of slot packages sp_1 and sp_2. Moreover, for all associations of slots, the modification is performed on the non-leader slot(s). Finally, once the parts reassembled following a given strategy, a simple kinematic resolution followed by the collision-free kinematic resolution are done in order to find a collision-free configuration of the assembly.

4.3.1. ReplacePart strategy

The ReplacePart strategy aims at replacing a part on any source assembly in any way possible while considering all available slot packages. This scenario aims to extend existing assembly databases, to create large ones ready for machine learning-based applications, for example. For the time being, as the proposed approach does not directly consider semantic information, the original database must be composed of assemblies that are roughly similar, so that the replaced parts do not have major semantic differences. Algorithm 2 details the different steps. The inputs are the source assemblies and the part to be replaced. A slot package is first created from the released slots in the assembly, and a search for parts with a complementary slot package is done. The found geometries are then imported and assembled one by one to explore the whole solution space. When needed, modifications are performed using the operators described in Section 3.4.

Algorithm 2 ReplacePart strategy

Require: a, k ▷ Assembly a and part index to be replaced k
Ensure: a
  $NCG \leftarrow$ Non-constrained geometries
  Delete(g_k)
  Delete(NCG)
  $sp_1 \leftarrow$ Empty SlotPackage
  for $i, j \leftarrow$ Av_slots(a) **do** ▷ s, g indexes of available slots in a
    Add_slot(s^i_j, sp_1) ▷ Add a slot in a sp
  end for
  $G \leftarrow g_i \in N^{db}_g \mid K_{sp_{g_i}} = \overline{K}_{sp_1}$
  $g_k \leftarrow g \in G$ ▷ Choice can be random or guided
  Import(g_k)
  Assemble(sp_1, sp_{g_k})

This strategy has been validated on the example of desk lamps whose bases and shaders have been replaced, and slot packages have

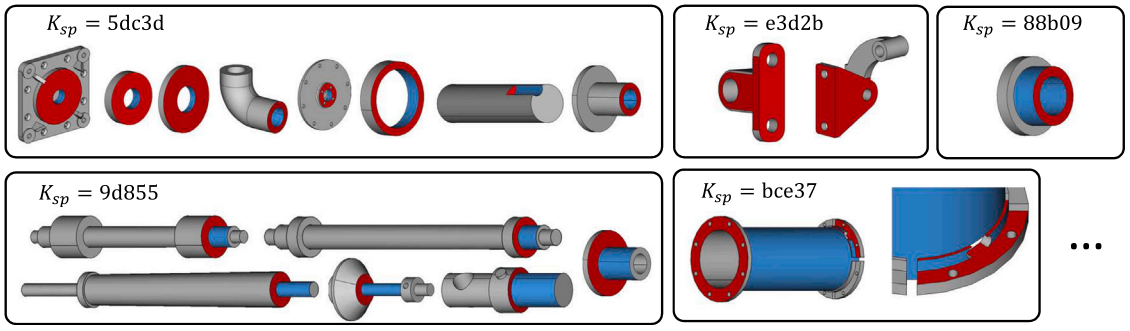


Fig. 14. Retrieved parts when using different slot packages and keys as queries, with all the retrieved parts coming from the same steam engine assembly database.

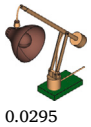
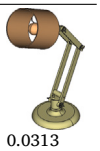
Fig. 15. Sample of 45 assemblies generated with ReplacePart strategy from 5 desk lamp input assemblies, while replacing the shaders and the bases.

been modified to fit (Fig. 15). The initial database contains 15 parts and 5 lamp assemblies, and 125 assemblies were generated. The import time of the source assembly is 1.2 s, while the time of each iteration is 2.5 s on average. Clearly, this is nothing like what a user could do. The iteration time is mostly determined by the collision-free kinematic resolution time, which can be time-consuming in some cases. The Fig. 15 shows a sample of the results obtained, while the complete dataset (125 assembled lamps) is available at the following URL: https://doi.org/10.5281/zenodo.7628235. No semantic information is taken into account during the reassembly process. The lamp example works well because the original database contains only desk lamps, i.e. similar models that can interestingly be combined. This strategy is particularly efficient with its automatic shape modification capabilities, as the five original lamps do not exactly have similar interfaces, and this is particularly useful for extending databases to be used for instance in machine learning-based applications.

Results are then compared with different metrics using CLIP algorithm [36] to assess the visual appearance of four of the generated assemblies (Table 3 with lamps #1 to #4). Basically, CLIP algorithm computes the distance between two features encoded from images or text (Section 3.6). Thus, two comparisons are here performed:

- First, the visual appearance of an assembly is assessed while averaging the CLIP values obtained when comparing multiple screenshots of the 3D model from 6 viewpoints to a sentence characterizing it, e.g. “A CAD model of a lamp” in the present case. The lower the score, the closer the latent vector of the image is to the latent vector of the text, and the more visually similar the models are. The results (Table 3, row 1) show a very high proximity between the 4 lamps, but lamp #1 remains the most distant from the sentence, which could allow it to be disqualified during reassembly. Indeed, the shader has been replaced by a part whose slots match, but whose semantic is completely different.
- Second, the similarity between two assemblies can be characterized. A reference assembly must then be defined. Here, assembly #3 is the reference. The evaluation can then be visual by

Table 3
Comparison of CAD models of lamps automatically generated with ReplacePart strategy: using CLIP and a sentence as reference (row 1), using CLIP and images of lamp #3 as a reference (row 2).

ID	#1	#2	#3	#4
Images				
CLIP wrt “Lamp”	0.0325	0.0295	0.0316	0.0313
CLIP wrt #3	0.1190	0.1187	0.0	0.0781

CLIP (i.e. average of the distances between the latent vectors of the 6 views). The results (Table 3, row 2) does not allow to differentiate the assemblies #1 and #2 significantly compared to the reference assembly #3. This is normal because the analysis does not include a semantic analysis, but only a geometric one.

4.3.2. *TargetGraph strategy*

The TargetGraph strategy aims to construct multiple assemblies conforming to an input linkage graph. This strategy can be particularly useful at the start of the design phase to quickly draft CAD assembly models associated with a given linkage graph. It can also be used to support the innovation process, generating multiple alternative solutions that meet the same requirements in terms of relationships between the parts involved.

Algorithm 3 TargetGraph strategy

Require: a ▷ Source assembly
Ensure: a
  for $i \leftarrow 1, N_g$ **do**
    $G \leftarrow g_k \forall k \in [1, N_g^{db}] \mid K_{sp_{g_i}} = K_{sp_{g_k}}$
    $g_{db} \leftarrow g \in G$ ▷ Choice can be random or guided
    Import(g_{db}) in a
  end for
  for $c_{k,i,j} \in a$ **do**
    map $\leftarrow$ isomorphism(sp_{g_i}, sp_{g_j})
    for $s_1, s_2 \in map$ **do**
      Assemble(s_1, s_2)
    end for
  end for

Algorithm 3 first searches for corresponding geometries in the database with the same general slot packages as the source geometries. In a second step, all geometries are constrained to each other following the linkage graph of the source assembly. An isomorphism must first be performed between the two slot packages of the source part and the part found in the database. This isomorphism allows to find a mapping of the slots which must be assembled. If no assembly error is returned, the final assembly has the same linkage graph as the source assembly.

Fig. 16 shows an example of the use of this strategy on a mechanical assembly to be achieved. The target linkage graph is specified for each interface to be completed, where in the gray nodes of the graphs represent the two tubes to be assembled. The graph on the left represents a screw-plate-nut assembly, while the one on the right contains an additional washer. Alternative solutions are then automatically retrieved from the database (composed of screws, nuts and washers) for each target graph, and a sample of three bolted solutions is proposed for each graph. A final solution is chosen to assemble the two tubes. The leader slots are defined here on the gray geometry slots, so the added components are automatically modified to fit the original interfaces. Here, linkage graphs are user-specified and correspond to classical ways of assembling parts with bolted solutions. Following this strategy, many alternative configurations can be tested and evaluated before choosing the final one, and this is much appreciated to optimize the considered design. The sources and generated CAD assemblies are available at the following URL: https://doi.org/10.5281/zenodo.7628235.

4.3.3. *FillBetweenTwoParts strategy*

This strategy aims to find and assemble a part to connect two given parts. This can be of interest to reuse existing parts, or for standardization purposes. The algorithm 4 presents the different steps. It takes as argument the two geometries to be connected, as well as their slot packages. Geometries that can connect these two parts are then searched in the database, and assembled to connect the two input parts.

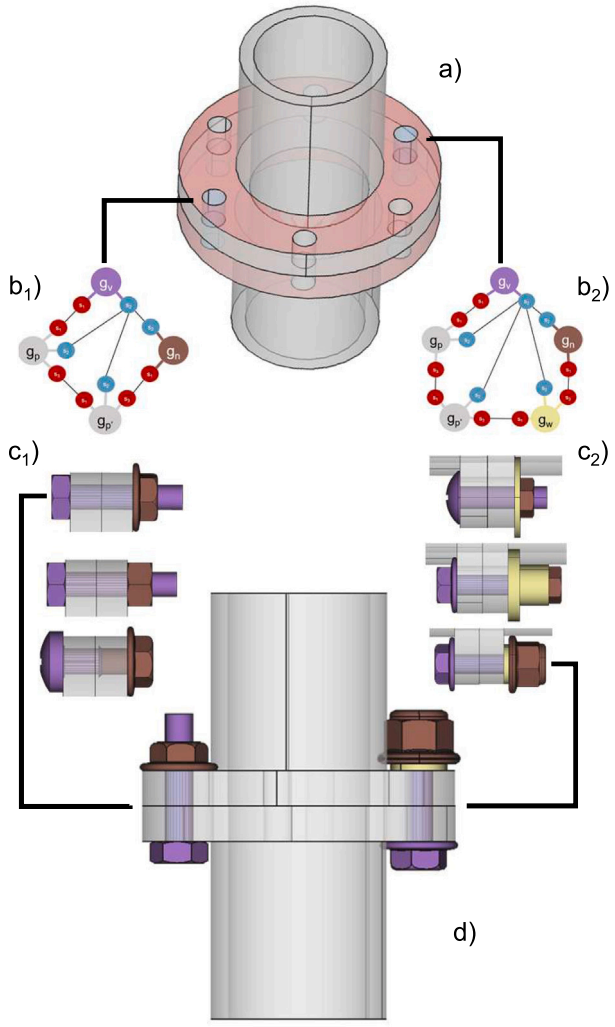


Fig. 16. Example of solutions obtained from the TargetGraph strategy. An assembly of two tubes to be bolted is used as input (a), and target linkage graphs are provided (b_1 for screw-nut and b_2 for screw-washer-nut). Possible solutions are generated for each graph (c_1, c_2). One of these solutions has been finally chosen, and the added parts are automatically modified to fit inside (d).

Algorithm 4 FillBetweenTwoParts strategy

Require: a, sp_1, sp_2 ▷ Empty assembly and two sp to fill
Ensure: a
  Import(g_1)
  Import(g_2)
  $G \leftarrow k \in [1, N_g^{db}] \mid \exists sp_i, sp_j \mid S_i \in g_k, S_j \in g_k, K_{sp_i} = \overline{K}_{sp_1}, K_{sp_j} = \overline{K}_{sp_2}$
  for $m \leftarrow G$ **do**
    Import(g_m)
    Assemble($sp_i^{g_m}, sp_1$)
    Assemble($sp_j^{g_m}, sp_2$)
  end for

Fig. 18 shows samples of assemblies generated following the FillBetweenTwoParts strategy and using purple and brown parts as input parts. The original CAD assembly model is the assembly (a). The database contains a lamp and a steam engine in order to diversify the model reuse. The iterations are fast (<1 s) and the import of the original assembly is almost instantaneous. Here, many parts allow the

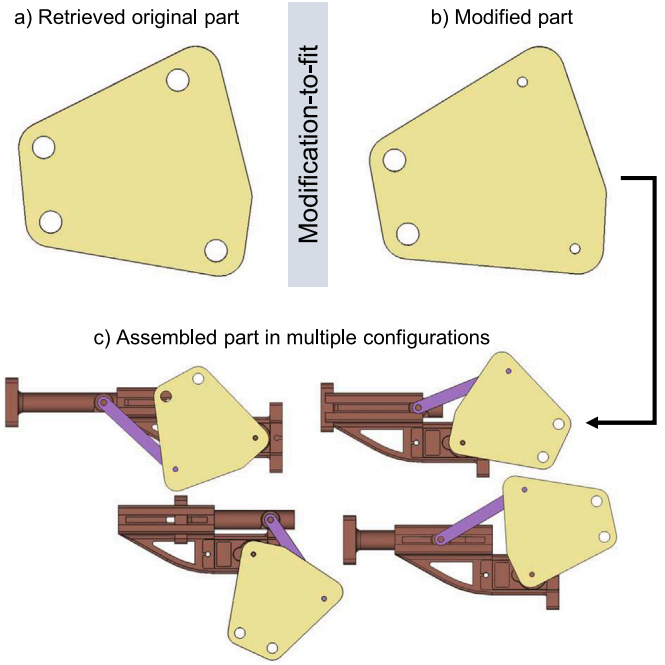


Fig. 17. Focus on configuration b_8 of Fig. 18: the retrieved part (a) is modified to fit in the new slots (b), and then assembled in the original assembly. Multiple configurations are shown (c) to demonstrate how to exploit the newly created kinematic constraints between the parts.

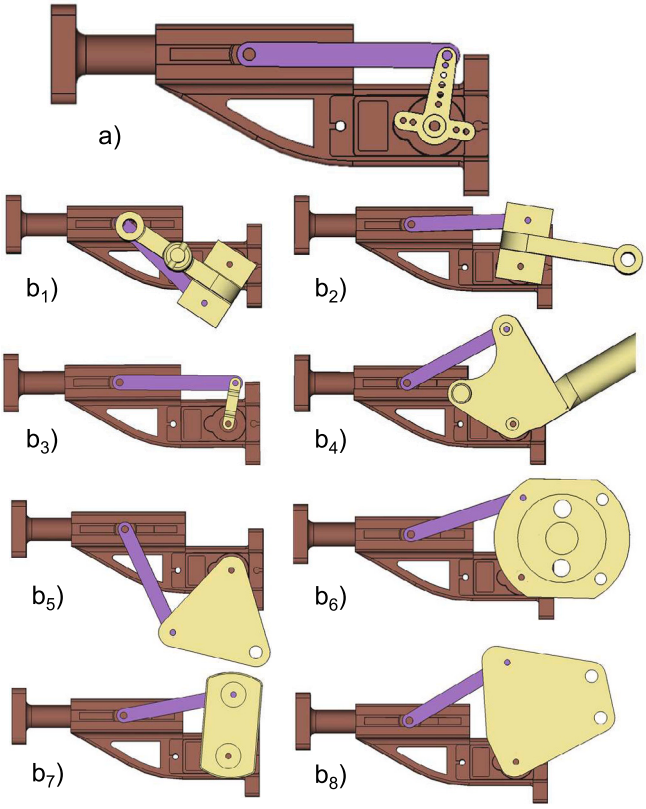


Fig. 18. Generated assemblies with FillBetweenTwoParts strategy using purple and brown parts as inputs. The original assembly model (a) is automatically assembled to the retrieved parts whose dimensions have been modified to fit (b_1 to b_8).

assembly of the rod and the servo-motor, but many have other functionalities, e.g plates or bases. The dimensions of the new assembled part (in yellow) are automatically modified to fit with cylinders from the servo-motor and rod, and the assembly itself is also updated. This is illustrated on Figs. 17a and 17b. This is particularly interesting to test alternative design solutions, with possible reuse of shapes. When the part is replaced, the functioning of the assembly is not checked, and a final verification of the mechanism must be carried out. This is done manually as shown on Figs. 17c. This method greatly facilitates the search for mechanical parts for reuse purposed in multipe configurations. All the sources and generated models are available at the following URL: https://doi.org/10.5281/zenodo.7628235.

5. Conclusion and future work

This paper has introduced a new framework for reassembling CAD models using a part-by-part interface-based identification algorithm that is capable of characterizing each part according to the assembly slots and interfaces it offers. To allow fast search of parts in the database, the slots are hashed, and the generated keys are used to easily identify parts candidate for the reassembly. Several reassembly strategies have also been proposed to validate the interest of the approach in various scenarios, including expanding a database to create multiple labeled assemblies from a few, or creating assemblies from a graph, or connecting existing parts in an assembly for model reuse or standardization purposed. During the reassembly steps, the geometry of the slots can be automatically modified to adjust their dimensions, in order to ensure a perfect fit and to avoid interference at the level of the interfaces. A collision-free kinematic solver has also been developed to automatically generate visually consistent assemblies. The generated CAD assemblies are thus well-constrained, with collision-free parts, and they are ready-to-use for downstream applications. The resulting database can also be further expanded while modifying the relative positions/orientations of the assembled parts. Such an automatic reassembly process is particularly innovative, and it clearly paves the way for smart assembly procedures.

Regarding the future works, four main improvements are already thought. First, some steps of the proposed reassembling strategy require to deal with good quality models. This is for instance the case for the collision-free solver that requires the parts to be watertight to be able to compute the intersection volumes between colliding parts. This is an assumption in this work, but also a strength of the proposed approach, as the huge batch of newly created models can inherit of this good quality. In the future, it will be interesting to imagine alternative solutions to be able to tackle a larger set of configurations where CAD parts and assemblies are not perfect. Second, more types of constraints should be managed to enrich the capacity of the method to detect additional kinematic constraints. Also, the filtering of the generated assemblies according to their functional characteristics (e.g. mechanical ones) is to be considered. The method can also be extended to treat sub-assemblies, that are indeed essential when generating semantically consistent assemblies. Finally, the modification functions should be expanded to enlarge the capacity of the method in generating a larger range of admissible solutions from dead CAD models, and to allow more parts to be reused.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Vergez Lucas reports financial support was provided by French National Research Agency. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is available at: https://zenodo.org/record/7628235.

Acknowledgments

This work was supported by the French National Research Agency. The authors would also like to thank the reviewers for their valuable comments, which were very helpful in proposing this paper.

References

[1] Berger M, Tagliasacchi A, Seversky LM, Alliez P, Guennebaud G, Levine JA, et al. A survey of surface reconstruction from point clouds. Comp Graphics Forum 2017;36(1):301–29.
[2] Shah GA, Polette A, Pernot J-P, Giannini F, Monti M. Simulated annealing-based fitting of CAD models to point clouds of mechanical parts' assemblies. Eng Comput 2021;37(4):2891–909.
[3] Lupinetti K, Pernot J-P, Monti M, Giannini F. Content-based CAD assembly model retrieval: Survey and future challenges. CAD 2019;113:62–81.
[4] Hu S, Polette A, Pernot J-P. SMA-Net: Deep learning-based identification and fitting of CAD models from point clouds. Eng Comp 2022;38:1–22.
[5] Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. J Big Data 2019;6(1):1–48.
[6] Kim JW, Kang KK, Lee JH. Survey on automated LEGO assembly construction. In: 22th Conf. in central Eur. comp. graphics. 2014, p. 89–96.
[7] Willis KD, Jayaraman PK, Chu H, Tian Y, Li Y, Grandi D, et al. Joinable: Learning bottom-up assembly of parametric cad joints. In: Proc. of the IEEE/CVF conf. on comp. vision and pattern recognition. 2022, p. 15849–60.
[8] Koch S, Matveev A, Jiang Z, Williams F, Artemov A, Burnaev E, et al. ABC: A big cad model dataset for geometric deep learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019, p. 9601–11.
[9] Chang AX, Funkhouser T, Guibas L, Hanrahan P, Huang Q, Li Z, et al. ShapeNet: An information-rich 3D model repository. Tech. Rep., Stanford University, Princeton University, Toyota Technological Institute at Chicago; 2015, arXiv: 1512.03012 [cs.GR].
[10] Jourdes F, Bonneau G-P, Hahmann S, Léon J-C, Faure F. Computation of components' interfaces in highly complex assemblies. Comput Aided Des 2014;46:170–8.
[11] Lupinetti K, Giannini F, Monti M, Pernot J-P. Multi-criteria retrieval of CAD assembly models. J Comput Des Eng 2018;5(1):41–53.
[12] Wang P, Li Y, Zhang J, Yu J. An assembly retrieval approach based on shape distributions and Earth Mover's Distance. Int J Adv Manuf Technol 2016;86(9):2635–51.
[13] You C-F, Tsai Y-L. 3D solid model retrieval for engineering reuse based on local feature correspondence. Int J Adv Manuf Technol 2010;46(5):649–61.
[14] Ma L, Huang Z, Wang Y. Automatic discovery of common design structures in CAD models. Comput Graph 2010;34(5):545–55.
[15] Chen X, Gao S, Guo S, Bai J. A flexible assembly retrieval approach for model reuse. Comput Aided Des 2012;44(6):554–74.
[16] Cardone A, Gupta S. Similarity assessment based on face alignment using attributed vectors. CAD Appl 2006;3(5):645–54.
[17] Tsai C-Y, Tien F-C, Pan T-Y. Development of an XML-based structural product retrieval system for virtual enterprises. Int J Prod Res 2004;42(8):1505–24.
[18] Bonino B, Giannini F, Monti M, Raffaeli R. Shape and context-based recognition of standard mechanical parts in CAD models. Comput Aided Des 2023;155:103438.
[19] Bahubalendruni MR, Biswal BB. A review on assembly sequence generation and its automation. Proc Inst Mech Eng J Mech Eng Sci 2016;230(5):824–38.
[20] Tian Y, Xu J, Li Y, Luo J, Sueda S, Li H, et al. Assemble them all: Physics-based planning for generalizable assembly by disassembly. ACM Trans Graph 2022;41(6):1–11.
[21] Freeman H, Garder L. Apictorial Jigsaw puzzles: The computer solution of a problem in pattern recognition. IEEE TEC 1964;13(2):118–27.
[22] Huang Q-X, Flöry S, Gelfand N, Hofer M, Pottmann H. Reassembling fractured objects by geometric matching. ACM 2006;569–78.
[23] Kalogerakis E, Chaudhuri S, Koller D, Koltun V. A probabilistic model for component-based shape synthesis. ACM TOG 2012;31(4):1–11.
[24] Liu H, Vimont U, Wand M, Cani M-P, Hahmann S, Rohmer D, et al. Replaceable substructures for efficient part-based modeling. Comput Graph Forum 2015;34:503–13.
[25] Zhang K-K, Hu K-M, Yin L-C, Yan D-M, Wang B. CAD parts-based assembly modeling by probabilistic reasoning. In: 14th International conference on computer-aided design and computer graphics. IEEE; 2015, p. 89–96.
[26] Schulz A, Shamir A, Levin DI, Sitthi-Amorn P, Matusik W. Design and fabrication by example. ACM (TOG) 2014;33(4):1–11.
[27] Jones B, Hildreth D, Chen D, Baran I, Kim VG, Schulz A. Automate: A dataset and learning approach for automatic mating of cad assemblies. ACM Trans Graph 2021;40(6):1–18.
[28] Li Y, Mo K, Duan Y, Wang H, Zhang J, Shao L, et al. Category-level multi-part multi-joint 3D shape assembly. Tech. Rep., 2023, arXiv preprint arXiv: 2303.06163.
[29] Westhues J, Coworkers. SolveSpace repository. https://github.com/solvespace/solvespace.
[30] Lei Z, Coworkers. Assembly3 repository. https://github.com/realthunder/FreeCAD_assembly3.
[31] Fudos I, Hoffmann CM. A graph-constructive approach to solving systems of geometric constraints. ACM TOG 1997;16(2):179–216.
[32] Braun K, Coworkers. A2+ WorkBench Repository. https://github.com/kbwbe/A2plus.
[33] Aristidou A, Lasenby J. Inverse kinematics: A review of existing techniques and introduction of a new fast iterative solver. Tech. rep., Univ. of Cambridge, Department of Engineering; 2009.
[34] Behandish M, Ilieş HT. Peg-in-hole revisited: A generic force model for haptic assembly. J Comput Inf Sci Eng 2015;15(4):041004.
[35] GrabCAD library. https://grabcad.com/library.
[36] Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, et al. Learning transferable visual models from natural language supervision. In: International conference on machine learning. PMLR; 2021, p. 8748–63.
[37] Crozet S, Léon J-C, Merlhiot X. Fast computation of local minimal distances between CAD models for dynamics simulation. Comput-Aided Des Appl 2018;15(4):585–600.
[38] Turner JU, Subramaniam S, Gupta S. Constraint representation and reduction in assembly modeling and analysis. IEEE Trans Robot Autom 1992;8(6):741–50.
[39] Gueron S, Johnson S, Walker J. SHA-512/256. In: Conf. on information technology: New generations. Vol. 69. IEEE; 2011, p. 354–8.
[40] Shervashidze N, Schweitzer P, Van Leeuwen EJ, Mehlhorn K, Borgwardt KM. Weisfeiler-Lehman graph kernels. J Mach Learn Res 2011;12(9):2539–61.