On the Role of Graph Theory Apparatus in a CAD Modeling Kernel

Sergey Slyadnev[1], Alexander Malyshev[1], Andrey Voevodin[1], and Vadim Turlapov[2]

[1] OPEN CASCADE, Nizhny Novgorod, Russia.
[2] Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia.
{sergey.slyadnev, al.s.malyshev, a.m.voyevodin, vadim.turlapov}@gmail.com

Abstract. This paper summarizes the experience of authors in solving a broad range of CAD modeling problems where the formalism of graph theory demonstrates its expressive power. Some results reported in this paper have never been published elsewhere. The set of topological and geometric heuristics backing the subgraph isomorphism algorithm is presented to achieve decent performance in our extensible feature recognition framework. By the example of sheet metal features, we show that using wise topological and geometric heuristics speeds up the search process up to interactive performance rates. For detecting CAD part's type, we present the connected components' analysis in the attributed adjacency graph. Our approach allows for identifying two-sided CAD parts, such as sheet metals, tubes, and flat plates. We use the notion of face transition graph for the unfoldability analysis. The basic operations on hierarchical assembly graphs are formalized in terms of graph theory for handling CAD assemblies. We describe instance singling operation that allows for addressing unique part's occurrences in the component tree of an assembly. The presented algorithms and ideas demonstrated their efficiency and accuracy in the bunch of industrial applications developed by our team.

Keywords: Computer-aided Design, Geometric Modeling, Feature Recognition, Hierarchical Assembly Graph, Attributed Adjacency Graph, Subgraph Isomorphism

1 Introduction

Because of its inherent simplicity, the apparatus of graph theory found extensive use in industrial geometric modeling. One of the well-turned ideas emerged early on was distinguishing between the syntax (topology) and the semantics (geometry) of digital shape representation. In the boundary representation scheme [15], the topology can be expressed in the form of an acyclic directed simple graph (having no self-loops and parallel edges [4]). The topology graph determines how the primary topological elements (faces, edges, and vertices) are nested. The possibility to have several parent nodes for a

single child enables sharing and instancing of the boundary elements. Therefore, graph structures persist in the very foundation of the geometric modeling systems.

Besides the topology graph, many other graph structures arise in the modeling systems for solving specific problems. We name a few most commonly used data structures here. The *face adjacency graphs* are extensively used in feature recognition methods. The *hierarchical assembly graphs* are employed to represent and manipulate complex product representations where single components are arranged following their "part-of" relations. The *function dependency graphs* allow building up the parametric CAD models and automate the dependent algorithms' execution. Additional specialized graph structures serve at solving specific design problems, such as recognition of a model type or digital shape reconstruction. For polygonal models, graph formalism facilitates segmentation aimed at the recovery of missing design intent.

In this paper, we review some of the commonly used graph structures following our CAD software development experience. Some reported results have never been published elsewhere. Among such, the formalism that we used for simplification of CAD models in our CAD Processor software [13] (see subsection 2.1). The graph-based approach aimed at the recognition of two-sided models and their types (section 3) also presents the novelty of our paper.

A graph is a pair (V, E), where V is a set of vertices (nodes) and E is a set of edges (arcs). We use the terms "nodes," "vertices," "arcs," and "edges" interchangeably as the meaning of each term should be clear from the context.

1.1 Attributed adjacency graph

One of the early contributions to the feature recognition field was the introduction of attributed adjacency graph or AAG [8]. The AAG is an undirected graph whose nodes represent B-rep faces, and the arcs encode their adjacency relations (Fig. 1). AAG facilitates solving a wide range of recognition problems bringing there the formalism of graph theory. All nodes and arcs of the AAG can be associated with the application-specific attributes. The attributes aim to capture specific properties of the underlying shape, such as dihedral angles, surface types, and different topological cues (e.g., presence of inner contours in faces).

The AAG plays a central role in recognizing volume features, such as holes, pockets, or bosses [11]. The same apparatus applies to the recognition of the secondary features, such as blends and chamfers [17]. The methods presented in sections 2 and 3 exploit AAG as the primary data structure.

1.2 Hierarchical assembly graph

The hierarchical assembly graph (HAG) is a directed acyclic graph aimed at representing complex CAD products. The nodes of the graph represent CAD parts and subassemblies (also called "prototypes"), while arcs denote occurrences of prototypes within each other following their "part-of" relations. Affine transformations are attached to arcs to give the corresponding occurrences proper placement in the modeling space. We present more details on HAG in section 4.

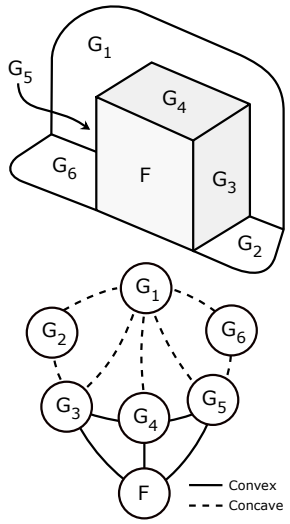

Fig. 1. Attributed adjacency graph (on the bottom) for a portion of a CAD model (on the top). The dashed lines represent concave dihedral angles. The solid lines represent convex dihedral angles.

1.3 Other uses of graphs

Graph theory found such widespread use in the computer-aided design field that it is hardly possible to enumerate all its occurrences. Below, we briefly discuss some additional application areas that are of particular interest to our research team. The reader is directed to the cited papers for more details on the touched subjects.

Topological naming One issue which is well-understood yet hard to resolve is topological naming. The naming mechanism supplies boundary elements of a CAD model with persistent identifiers invariant over modifications of a model, including its rebuilding from scratch. The application has to reattach any attributes associated with faces, edges, and vertices whenever a model's geometry is changed. Such a mechanism is traditionally a backbone of history-based parametric modeling systems.

Rémi Lequette attempts to establish a topological naming mechanism in a geometric modeling kernel (CAS.CADE) independent from any CAD system [10]. Two approaches are discussed by Lequette: labeling and pattern matching. The labeling technique attaches application-specific identifiers directly to the boundary elements of a model (if the geometric kernel supports this) or to the nodes of a history graph. Given that geometric modeling operations support the history of modification, these approaches are equivalent. The pattern matching technique based on searching for graph isomorphisms is seen as a complementary technique for labeling and is aimed at removing ambiguities.

Jiri Kripac introduces one of the well-known approaches [9] to solve the topological naming problem. His method is based on the utilization of a face history graph.

The graph tracks the evolution of faces, including their creation, split, merge, and deletion. The face history graph is a directed acyclic graph with the nodes representing the faces being traced. The incoming edges of this graph represent the information of the ancestors of the face. The outgoing edges represent what happened to the face.

It should be noted that both approaches mentioned above use a variety of topological and geometric cues for seeking the boundary elements. A general mechanism for pattern matching is, therefore, of particular interest. In section 2.2, we elaborate on a set of efficient heuristics for matching CAD model's faces.

Region adjacency graph Region adjacency graph (RAG) is the generalization of attributed adjacency graph to the sets of faces. Unlike AAG, where each node represents a single face of a boundary representation, each node of the RAG encodes a region (a collection of faces). The graph edges express adjacency relations between the regions. One of RAG's application is the mesh segmentation [1]. As in the case of AAG, the nodes and arcs of RAG carry on the application-specific attributes, such as angle type, curvature maps, and others.

Mesh-based regions tend to capture the missing design intent of a polygonal model. The knowledge of regions unlocks further 3D processing workflows, such as quadrilateral meshing [3, 2], digital shape reconstruction [22], mesh refinement or simplification [5].

Dependency graphs A dependency graph is a data structure that allows for the automatic execution of algorithms whose inputs and outputs are interrelated. Such graphs found extensive use in the engineering software packages ranging from computer-aided design to numerical simulation (ANSYS Workbench system is an example).

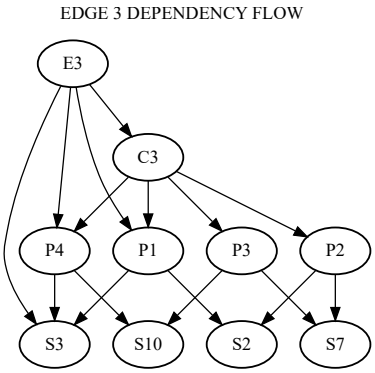

Fig. 2. A component of a dependency graph in digital shape reconstruction system.

Fig. 2 illustrates a part of the dependency graph used in our reverse engineering software package [16] for parameterizing the shape reconstruction process. The nodes

of this graph represent the algorithms, and the arcs denote the dependency relations. For example, the algorithm P1 cannot be executed unless the algorithms C3 and E3 are done. A critical aspect of any system that employs dependency graphs is searching and resolving cyclic dependencies.

2 Feature recognition

In this section, we discuss the applications of graph theory to solving the feature recognition problem. In contrast to the "design-by-features" paradigm of modeling, feature recognition allows for extracting the *design intent* from a CAD model that does not possess a construction history. The applications of the feature recognition technique are manifold, ranging from shape modeling to fabrication costs estimation.

This section explains how graph theory's formalism can be exploited in solving two engineering problems: CAD data simplification and sheet metal unfolding. *Feature recognition* aims at extracting specific groups of B-rep faces for further reasoning. In *simplification*, the engineer may want to suppress all small holes and blends to switch from a detailed design geometry to a simulation-ready model (e.g., to perform structural analysis with FEM). In sheet metal recognition, extracting such features as bends, walls, louvers, or bridges (to name a few) facilitates manufacturing planning.

2.1 Isolated features

One approach for the recognition of isolated features was presented in our previous paper [11]. In particular, it was shown how to compose different geometric and topological heuristics for detecting specific types of volume features, such as holes, bosses, pockets, and arbitrarily shaped cavities. In addition to that work, we present a new method for detecting isolated groups of faces that are separable from the given set of capping faces.

Let G be the attributed adjacency graph of a CAD part in question, and

$$F = \left\{ f_b^1, f_b^2, ..., f_b^N \right\}$$

be the serial indices of the selected *base faces* [11] (we assume that some topology exploration algorithm enumerates all faces of a model). A base face is a face separating a feature from the rest of a CAD model (we elaborate on the separability property below).

The following properties are checked to detect and suppress the isolated features starting from a set of base faces:

Separability The subgraph $G - F$ obtained by the deletion of the base faces' nodes should contain at least $c + 1$ connected components (there can be more if the base faces enclose several inner contours). Here, c is the number of connected components in the initial graph G. This condition can only hold if the target feature is geometrically separable, i.e., it does not contain any perforations such as through holes. Therefore, for the successful detection of isolated features, the CAD model may need additional simplification to ensure the separability property.

Suppressibility This property captures the ability to suppress the detected feature faces for the sake of CAD part simplification [19]. If $|F| = 1$, suppression is done by a graph reduction algorithm that eliminates the detected boundary elements from the topology graph of a B-rep model [11]. Such an approach allows for working at the model's syntax level without running any geometric computations. Therefore, the suppression method is fast and reliable, though it is limited to simple cases only. If $|F| > 1$, a more general face removal (FR) operator [23] is used for suppression. The FR algorithm removes a target face or a set of faces from the structure of a model with subsequently merging the adjacent faces to restore adjacency. The analysis of suppressibility employs checking two properties expanded below.

Inner boundary inclusion Let $C \subset (G - F)$ be a connected component capped by the base faces F. Let $e(\cdot)$ denote all B-rep edges extracted from a face set in question. Let $i(e(\cdot))$ denote the inner edges, i.e., the edges constituting inner contours of a face set (Fig. 3). Then, the condition $i(e(F)) \subseteq e(C)$ expresses the requirement of boundary inclusion for the inner edges of F into the separated feature C. If this condition does not hold, the inner edges of the base faces could not be suppressed. The inclusion condition should be tested for each inner contour of the base set F independently as, for example, this condition does not hold for a base face with multiple holes.

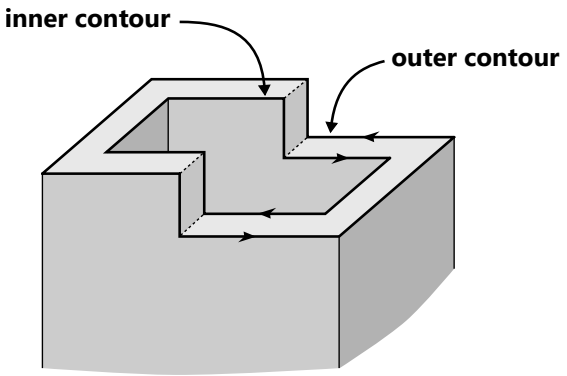

Fig. 3. To the detection of isolated features.

Outer boundary exclusion Let $o(e(\cdot))$ denote the outer edges of a given face set (Fig. 3). Let $C_o \subset G$ denote the faces in G that are adjacent to F via its outer edges $o(e(F))$. The condition $C \cap C_o = \emptyset$ restricts the isolated feature C from having any common edges with the outer contour of F. If this condition does not hold, the outer edges of F do not remain intact on suppression, hence the feature is not suppressible.

The outer boundary exclusion condition may look excessive as most separable features growing from the inner edges automatically satisfy it. However, this condition

ensures that the suppression algorithm will never affect the capping faces' outer boundaries as such behavior is considered destructive.

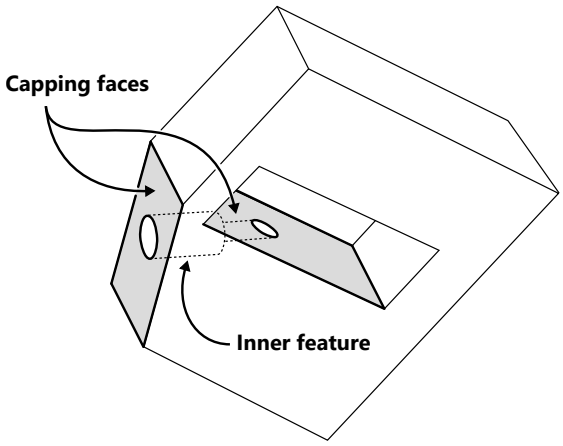

Fig. 4. Sample part with a pair of capping faces selected to isolate an inner cavity.

The formalism outlined above allows for generalization to a disconnected set of faces when f_b^i are not adjacent (Fig. 4). As a result, through channels between the specified inlet and outlet faces can be extracted (to be used in CFD analysis, for example).

2.2 Efficient heuristics for subgraph isomorphism

Two graphs G and G' are said to be isomorphic if there is a one-to-one correspondence between their nodes and arcs such that the incidence relationship is preserved [4]. Similarly, a graph P is isomorphic to a subgraph of G if there can be found a subgraph $g \subset G$ isomorphic to P.

The subgraph isomorphism problem is known to be NP-complete [6]. The discovery of NP-completeness is often posed as an argument to avoid using the algorithm as it is anticipated to be computationally exhaustive and hence impractical. We consider two primary techniques for gaining a satisfactory performance in the graph matching procedure: sparsing M_0 matrix and dimension reduction. Both methods are described below in more detail.

We apply the classic Ullman's algorithm for subgraph isomorphism [21]. Adrian Neumann gives some helpful implementation guides on his website [12] that we also followed.

The algorithm starts by constructing an attributed adjacency graph G for the whole CAD part (Fig. 5). The pattern feature to match is represented by the attributed adjacency graph P (Fig. 6). If what follows, we assume that all graphs are specified with their adjacency matrices. The adjacency matrix G for the test part illustrated by Fig.

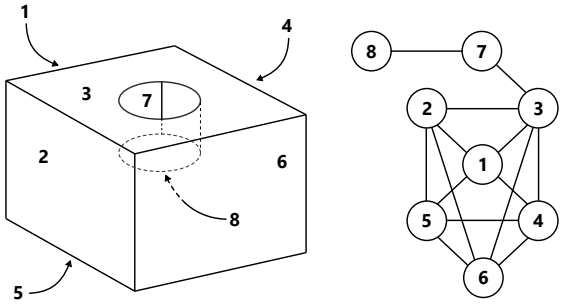

Fig. 5. Sample part with its attributed adjacency graph G for subgraph matching.

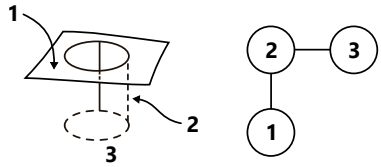

Fig. 6. Pattern feature with its attributed adjacency graph P.

5 has the following form (dashed lines decorate the corresponding 1-based indices of B-rep faces):

```
1 2 3 4 5 6 7 8
----------------+
0 1 1 1 1 0 0 0 | 1
1 0 1 0 1 1 0 0 | 2
1 1 0 1 0 1 1 0 | 3
1 0 1 0 1 1 0 0 | 4
1 1 0 1 0 1 0 0 | 5
0 1 1 1 1 0 0 0 | 6
0 0 1 0 0 0 0 1 | 7
0 0 0 0 0 0 1 0 | 8
```

The adjacency matrix P is represented as follows:

```
1 2 3
------+
0 1 0 | 1
1 0 1 | 2
0 1 0 | 3
```

The algorithm sequentially constructs candidate isomorphism matrices M of dimensions $[K \times N]$, where K is the number of nodes in P, and N is the number of nodes in G. Checking

$$M(MG)^T = P \tag{1}$$

one can verify that the following matrix M encodes the sought-for bijection (that can be verified looking at Figures 5 and 6):

```
1 2 3 4 5 6 7 8
---------------+
0 0 1 0 0 0 0 0 | 1
0 0 0 0 0 0 1 0 | 2
0 0 0 0 0 0 0 1 | 3
```

The initial matrix M_0 is constructed in a way to specify all possible bijections between the nodes of P and G. E.g., the following matrix prohibits mapping $(2, 8)$ as $d(2) \neq d(8)$, where $d(\cdot)$ denotes a degree of a graph node.

```
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
```

The algorithm constructs candidate matrices M deriving them from M_0 so that each row contains exactly one 1, and a column contains at most one 1. Each candidate matrix is checked for isomorphism using the equation (1).

For efficient matching, the initial matrix M_0 should be as sparse as possible. Furthermore, the dimension N can often be reduced based on the geometric rationale. Both reduction techniques contribute to the search space pruning employing dedicated geometric and topological heuristics that are discussed below.

Sparsing M_0 matrix The element $M(v_P, v_G)$ is nullified if any of the following conditions are satisfied for any pair of vertices in P and G graphs:

- $d(v_P) > d(v_G)$, i.e., the pattern's vertex v_P has more incident arcs than the corresponding vertex v_G of the problem graph G. This condition is hardly a heuristic as it should always hold. We enumerate it here with others to have the entire heuristics set in one list.
- Since AAG encodes concave/convex properties of the dihedral angles in its arc attributes, they are also taken into account. Let $\alpha(\cdot)$ be angle type attribute in the adjacency graph. We formulate $A_P = \{\alpha(e_i) : e_i \in E_P\}$, where E_P is the set of edges for the graph P. The set A_G is formulated similarly. The heuristic for $M(v_P, v_G)$ nullification requires that $A_P \subset A_G$, i.e. all the pattern's angles can be found in the subgraph of G being tested.
- As every AAG node corresponds to a B-rep face, some additional properties can be queried from the geometric and topological data structures. E.g., the number of vertices, edges, and contours should be identical for the matched faces (though, in some cases, this restriction can be relaxed). The types of host surfaces should match as well.
- More heuristics can be added to narrow down the search space. E.g., it is possible to take into account such geometric properties as dimensions of the B-rep elements being matched.

Dimension reduction Joshi and Chang proposed a heuristic aimed at efficient recognition of negative volume features, such as holes or pockets [8]. The heuristic consists of eliminating (from AAG) the B-rep faces having convex edges only. Another graph reduction heuristic that we adopted in our work is the exclusion of faces having internal loops ("base faces"). Applicability of one or another heuristic depends on the type of feature being matched. E.g., base faces can often be excluded when searching for sheet metal features, such as bridges or louvers [7], but not for countersunk or counterbored holes (the latter often employ faces with inner contours).

A practical implementation of the graph isomorphism method should employ one or the other heuristic set depending on the feature type. From the software architecture standpoint, all detectable features and their effective heuristics can be organized in a catalog (e.g., a file or a database). With such a data-driven architecture, the users are permitted to extend the catalog with their application-specific features.

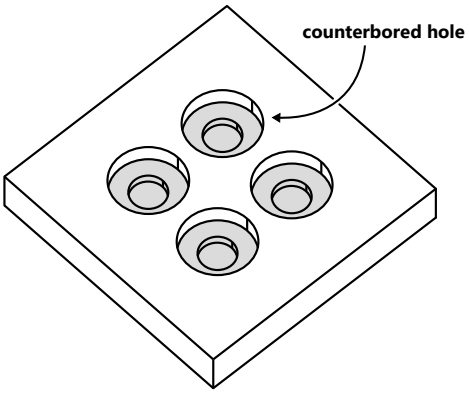

Fig. 7. Sample flat part with a number of blind counterbored holes.

Table 1 shows the results of experiment conducted on a laptop computer with Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz, 16GB RAM, OS Windows 10. The input CAD model represents a flat plate with a rectangular pattern of counterbored holes (Fig. 7). Each hole is a shell composed of four faces ($K = 4$). The total number of faces N is a modeling parameter.

As one can see from the experimental data, excluding convex-only faces allows us to cope with the algorithm's high inherent complexity. The detection is done in a fraction of second even for large CAD parts containing thousands of faces.

Another advantage of the heuristics mentioned above is that they break down the initial graph G into a set of connected components. Since all connected components are independent of each other, the graph isomorphism algorithm can be launched in parallel mode (using several CPU or GPU cores). In our experiments, we process each connected component separately and leave parallel processing for future investigation.

Table 1. Execution time of subgraph isomorphism with and without graph reduction heuristics. Graph isomorphism was used to detect all counterbored holes on a flat shape ($K = 4$).

N	Num. features	No heuristics [sec]	Exclude convex [sec]
406	100	0.06	0.04
906	225	0.6	0.05
1606	400	3.2	0.09
2506	625	12.2	0.15
3606	900	36.9	0.22
4906	1225	98	0.35

3 Object classification

3.1 Two-sided models

Two-sided models are widely spread in the industry. For these kinds of models, humans can quickly identify a lower and an upper side, usually residing at a constant thickness from each other. Some examples of such objects include flat plates (Fig. 7), folded sheet metal parts (Fig. 8), profiles, and tubes (Fig. 9). These objects are computationally convenient as they possess a certain set of properties that allow for reliable checking by ad-hoc heuristics. The graph-based heuristics play a central role in capturing the essence of a two-sided object: the possibility of separating both sides. Here we describe the recognition process for the folded sheet metal parts, including straight rectangular tubes. For other two-sided objects, the recognition principles may differ in the set of employed heuristics, while the basic principle of AAG separation holds.

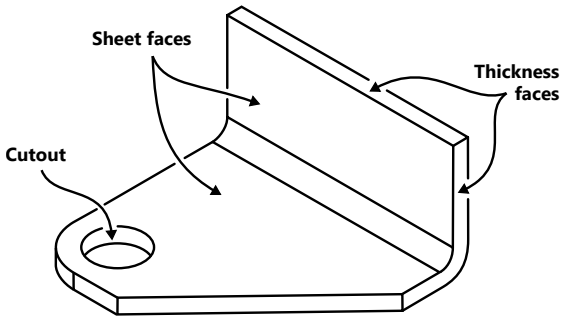

Fig. 8. Sheet metal part.

The recognition process starts from choosing a seed face (typically, a planar one) and finding the opposite (mate) face by casting a ray pointing inside the part's material.

From the two detected faces, a propagation procedure starts. All neighbor faces are visited in the order, which assumes that the cylindrical faces (bends) accompany the planar faces. All neighbors should be smoothly connected. The iteration is done twice: starting from the seed face and its opposite one. The iteration procedure terminates when no neighbor faces are left in the model. All faces remaining unvisited by the end of propagation are declared as "thickness candidates." The indices of these faces ground the primary separation heuristics.

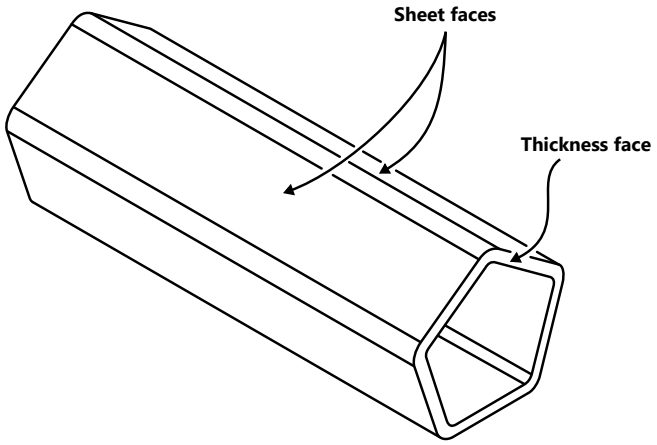

Fig. 9. Tube part.

The first heuristic consists of the deletion of the unvisited nodes from the AAG (Fig. 10a) and checking the number of connected components remaining in the graph (the elimination of a graph vertex always implies the deletion of its incident edges). For the two-sided parts, precisely two connected components have to remain after the removal of the thickness candidates (Fig. 10b).

The other heuristics starts from deleting all but unvisited faces from the initial state of AAG. The remaining nodes may yield a set of connected components representing both the sheet model's cutouts (Fig. 8) and its corner thickness faces. The cutouts are distinguished as the face sets residing on the internal contours of the base sheet faces (detected by the propagation process as described above). The detected groups of inner faces are eliminated from the reduced graph so that it finally contains one or several connected components representing only the outer cuttings of the sheet. The remaining nodes should either have no incident edges or yield circuits in the graph, i.e., they should happen to be arranged in loops, where each vertex has a degree of two (Fig. 10c).

3.2 Check for unbendable sheets

If a CAD part is recognized as a sheet metal object, that does not necessarily mean that it allows for unfolding (that is a basic requirement for properly designed sheet metals).

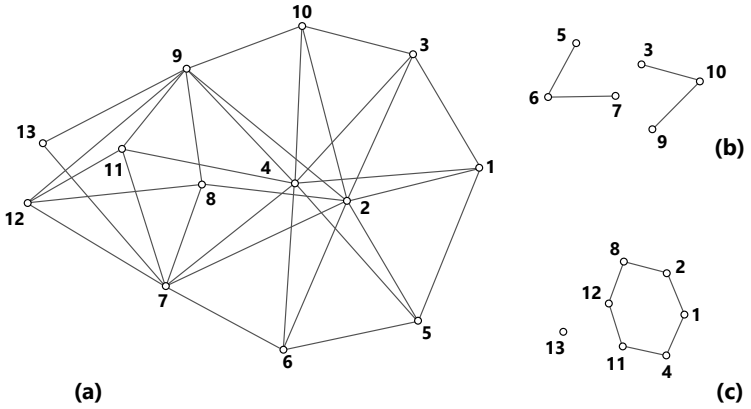

Fig. 10. AAG decomposition into connected components for the recognition of sheet metal parts. The graph in (a) corresponds to the CAD part illustrated in the Fig. 8. The derived connected components for the sheet and thickness faces are shown in (b) and (c).

From the AAG, we derive a graph named *face transition graph* (FTG). FTG is constructed by eliminating all vertices v_b representing the bend faces and collapsing their incident edges. The remaining vertices represent the sheet faces (sheet metal walls).

A two-stage procedure verifies if a CAD model is topologically closed. At the first stage, the FTG is searched for a circuit (Fig. 11), i.e. a closed train of vertices having the degree of two: $v_1\, e_1\, v_2\, e_2 \ldots v_1$, $deg(v_i) = 2$. If no circuits are detected, the verification procedure stops. Otherwise, a circuit is subsequently verified against the extra set of geometric heuristics that imply certain geometric checks on visiting the graph nodes. Stated briefly, the heuristics aim to calculate the angle spanned by the radius vector, pointing to each face being traversed. If the accumulated angle equals 2π, the CAD model is considered closed, hence not unfoldable. Otherwise, the next circuit is checked until no circuits remain in the FTG.

4 Hierarchical assembly graph

A hierarchical assembly graph (HAG) represents the as-designed structure of a digital product. The nodes of a HAG are the assembly components nested into each other. The directed arcs denote "part-of" relationships between the components. The components are nested into each other by instancing, i.e., referencing a unique geometry or subassembly (we use the "prototype" term to refer to both) with a specific matrix of rigid body transformation. The possibility for a prototype to be referenced several times yields multiple parent nodes in the graph. For example, consider a car chassis containing two axles and two wheels. Then, given that HAG is a multigraph (nodes are the prototypes, arcs are the instances), the assembly can be represented as shown in Fig. 12.

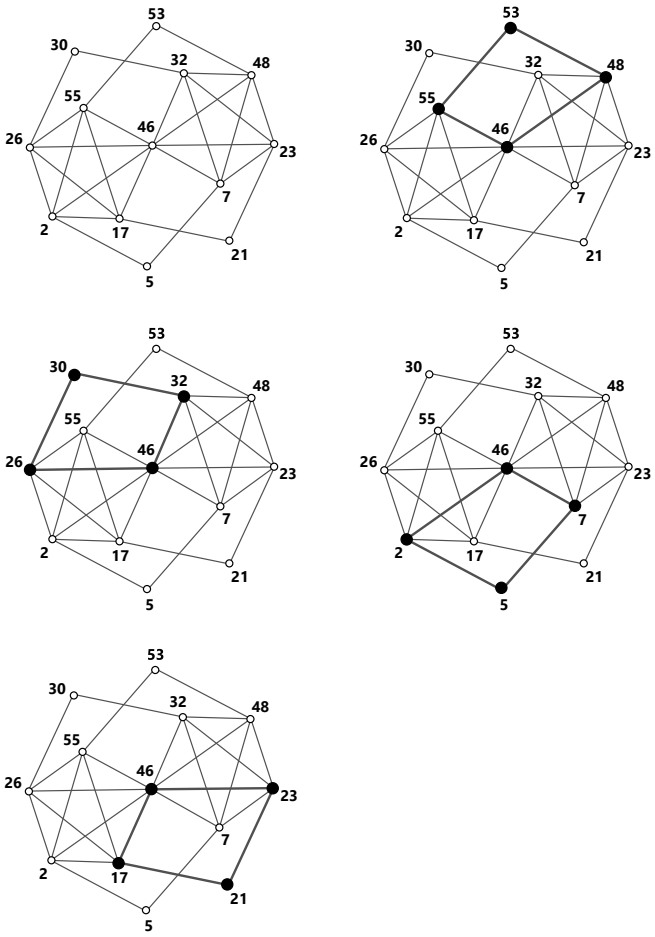

Fig. 11. FTG constructed for one side of a sample rectangular tube. Three simple circuits are outlined in bold. The indices of nodes correspond to the indices of faces in the CAD model.

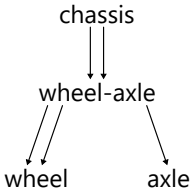

Fig. 12. A hierarchical assembly graph for a chassis model (nodes are the prototypes and arcs are the instances).

Any modification of a prototype affects all its occurrences in a product. However, it is often necessary to work with a single occurrence of a component that is addressed uniquely by a path of edges in the graph. Therefore, a mechanism to extract unique occurrences of the assembly components is of high interest. Such a tool called *instance singling* was thoroughly discussed by A. Rappoport [14].

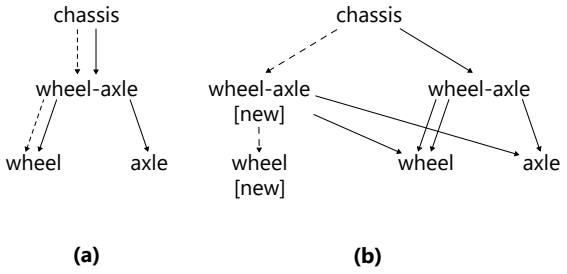

Fig. 13. (a) The hierarchical assembly graph for the chassis model. The dashed lines denote the path to be singled. (b) The hierarchical assembly graph after transformation. The dashed lines denote the image of the input path in the modified graph.

Let us consider a chassis model composed of two wheel-axle subassemblies (Fig. 12). A path from the root node down to the ultimate leaf addresses a single wheel of interest (Fig. 13a). After instance singling, the graph illustrated in Fig. 13b is obtained. Informally, instance singling operates as a "path disambiguation" operation. Once isolated, any domain-specific attributes, such as colors or design review notes, can be associated with the part in question, not affecting its other occurrences. In the example above, instance singling operation aims at deep copying the geometric representation of a part while preserving its placement in the modeling space.

Let $M_{i,j}$ denote the nodes of the HAG, where i is the depth from the root, and j is a serial index within a parent. Let $P = (e_0, e_1, ...e_{k-1})$ be a path to the node $M_{k,j}$ (not necessarily the leaf one). The instance singling operation transforms the hierarchical graph in a way to keep the number of paths unchanged while reusing as many existing connections between nodes as possible. The operation automatically copies the nodes preceding $M_{k,j}$ along the path P. Any edge $e \notin P$ and incident with the nodes of P is reused. Some results of the singling transformation are depicted in Fig. 14.

5 Conclusions and future work

Understanding graph formalism is one of the essential competencies a CAD practitioner has to possess. Graphs not only allow us to design efficient data structures but serve to formalize the driving ideas behind the CAD algorithms. For example, as we saw above, graphs can play a central role in recognizing sheet metal parts. As practice proves, adjacency graphs, including their variations, such as RAG or FTG, define a sound framework for feature recognition and object classification. At the higher level of

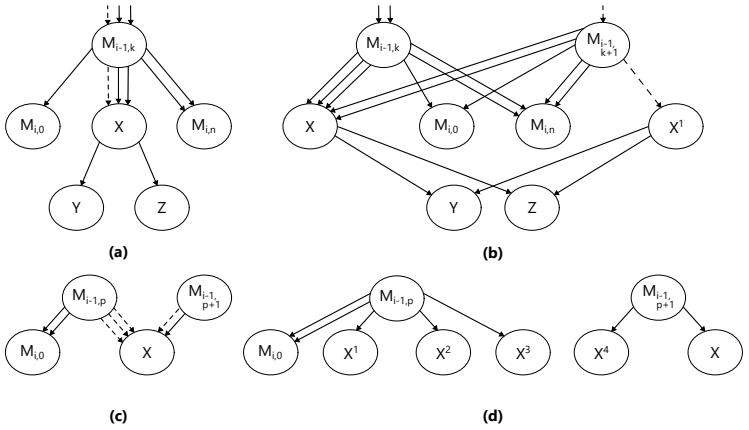

Fig. 14. (a) The initial HAG. The dashed lines denote the path to be singled. (b) The transformed HAG. The dashed lines denote the image of the singled path. The figures (c) and (d) illustrate how one prototype can be instanced by several assemblies.

abstraction, hierarchical assembly graphs serve to represent complex digital products. Any modification to the product structure is then expressed in the language of graph theory. A sound formalism is a critical factor in developing efficient and sustainable algorithms.

We publish parts of our research under open-source terms to stimulate interest in geometric modeling and feature recognition. Our Analysis Situs framework [18] allows reading CAD models in STEP format, construct AAG, and compute basic shape characteristics, such as dihedral angles. While initially purposed as a prototyping workbench, this software starts seeing some interest from academia [24] and industry. The recognition of sheet metal parts and feature-based approaches for CAD simplification are commercialized in our CAD Processor software package [13] backed by the corresponding research paper [20].

Some topics remain for future investigation. Among such, we consider further development of graph isomorphism approach aimed at combining feature definitions with the corresponding ad-hoc heuristics. Additionally, we are looking into developing a data framework for the hierarchical assembly graphs and exposing a common API for dealing with product structures while remaining compliant with international standards like ISO 10303 (STEP). Designing such a framework is currently our work in progress. The utilization of the region adjacency graphs for mesh segmentation is another direction of our future work.

Graphs allow for insightful visualization being of much help in 3D data analysis and geometric reasoning. According to our experience, graph visualization facilities are underexploited in the CAD community. Therefore, additional efforts worth spending on developing a software package aimed at cognitive visualization of graph structures in their application to CAD.

References

1. Agathos, A., Pratikakis, I., Perantonis, S., Sapidis, N., Azariadis, P.: 3D Mesh Segmentation Methodologies for CAD applications. Computer-Aided Design and Applications **4**(6), 827–841 (jan 2007)
2. Boier-Martin, I., Rushmeier, H., Jin, J.: Parameterization of triangle meshes over quadrilateral domains. ACM International Conference Proceeding Series **71**, 193–203 (2004)
3. Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., Zorin, D.: Quad-Mesh Generation and Processing: A Survey. Computer Graphics Forum **32**(6), 51–76 (sep 2013)
4. Deo, N.: Graph Theory with Applications to Engineering and Computer Science (Prentice Hall Series in Automatic Computation). Prentice-Hall, Inc., USA (1974)
5. Gao, S., Zhao, W., Lin, H., Yang, F., Chen, X.: Feature suppression based CAD mesh model simplification. Computer-Aided Design **42**(12), 1178–1188 (dec 2010)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA (1979)
7. Gupta, R.K., Gurumoorthy, B.: Classification, representation, and automatic extraction of deformation features in sheet metal parts. Computer-Aided Design **45**(11), 1469–1484 (nov 2013)
8. Joshi, S., Chang, T.C.: Graph-based heuristics for recognition of machined features from a 3D solid model. Computer-Aided Design **20**(2), 58–66 (mar 1988)
9. Kripac, J.: A mechanism for persistently naming topological entities in history-based parametric solid models. Computer-Aided Design **29**(2), 113–122 (feb 1997)
10. Lequette, R.: Considerations on topological naming. In: Pratt, M.J., Sriram, R.D., Wozny, M.J. (eds.) Product Modeling for Computer Integrated Design and Manufacture: TC5/WG5.2 International Workshop on Geometric Modeling in Computer Aided Design 19–23 May 1996, Airlie, Virginia, USA. pp. 394–403. Springer US, Boston, MA (1997)
11. Malyshev, A., Slyadnev, S., Turlapov, V.: Graph-based feature recognition and suppression on the solid models. In: GraphiCon 2017. pp. 319–322 (2017)
12. Neumann, A.: Ullman's subgraph isomorphism algorithm. URL: https://adriann.github.io/ullman subgraph isomorphism.html (accessed: 20.06.2020)
13. OPENCASCADE: CAD Processor: CAD-neutral simplification and preparation. URL: https://www.opencascade.com/content/cad-processor (accessed: 28.06.2020)
14. Rappoport, A.: A scheme for single instance representation in hierarchical assembly graphs. In: Falcidieno, B., Kunii, T.L. (eds.) Modeling in Computer Graphics. pp. 213–223. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
15. Requicha, A.G.: Representations for Rigid Solids: Theory, Methods, and Systems. ACM Computing Surveys **12**(4), 437–464 (dec 1980)
16. Slyadnev, S.E., Turlapov, V.E.: To the Development of Open Source Software for the Reconstruction of CAD Models. Programming and Computer Software **45**(4), 202–212 (jul 2019)
17. Slyadnev, S.E., Turlapov, V.E.: Simplification of CAD Models by Automatic Recognition and Suppression of Blend Chains. Programming and Computer Software **46**(3), 233–243 (may 2020)
18. Slyadnev, S., Malyshev, A., Turlapov, V.: CAD model inspection utility and prototyping framework based on OpenCascade. In: GraphiCon 2017. pp. 323–327. Perm, Russia (2017)
19. Slyadnev, S., Malyshev, A., Turlapov, V.: Automated history-free simplification of mechanical CAD models and assemblies. In: GraphiCon 2018. pp. 488–494 (2018)
20. Slyadnev, S., Malyshev, A., Turlapov, V.: Automated history-free simplification of mechanical CAD models and assemblies (in Russian). In: GraphiCon 2018. pp. 488–494 (2018)
21. Ullmann, J.R.: An Algorithm for Subgraph Isomorphism. Journal of the ACM (JACM) **23**(1), 31–42 (jan 1976). https://doi.org/10.1145/321921.321925

22. Varady, T.: Automatic Procedures to Create CAD Models from Measured Data. Computer-Aided Design and Applications **5**(5), 577–588 (jan 2008)
23. Venkataraman, S., Sohoni, M.: Reconstruction of feature volumes and feature suppression. In: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications. p. 60–71. SMA '02, Association for Computing Machinery, New York, NY, USA (2002)
24. Zhu, F.: Application of analytical and AI-based feature detecting methods for an Energy optimized industrial process. Bachelor's thesis, Technische Universitat Darmstadt (2019)