A Prototype of an Automated Feature Recognition Algorithm for Aerospace Sheet Metal Parts

Seyedmorteza Ghaffarishahri[1] and Louis Rivest[2]

[1]École de technologie supérieure, Université du Québec, seyedmorteza.ghaffarishahri.1@ens.etsmtl.ca
[2]École de technologie supérieure, Université du Québec, louis.rivest@etsmtl.ca

Corresponding author: Seyedmorteza Ghaffarishahri, seyedmorteza.ghaffarishahri.1@ens.etsmtl.ca

Abstract. Automated feature recognition (AFR) makes it possible to abstract semantic information from neutral CAD models. In an earlier work, we proposed an AFR method for aerospace sheet metal (ASM) parts. In this new work, that method's implementation as an AFR prototype is outlined and the differences between the prototype and the original proposal are pointed out. Then, streamlined data structures are described and explained. They are used to organize the B rep elements extracted from the ASM parts' STEP models, classify and enhance them, and structure the features recognized from the STEP models. Next, a few examples of the algorithms that are implemented in the prototype to manipulate the B rep elements and recognize features are represented and explained. The details of the algorithms are presented in the appendices. To validate the AFR method and verify its correct implementation, a collection of 26 real-world ASM parts was used to create CAD models that were subsequently converted to STEP models. The STEP models were processed to recognize their features, and the results show perfect accuracy. A few of the output feature files are presented in detail. Our results confirm great potential for further AFR method development for rather specialized domains of application.

Keywords: aerospace sheet metal, automated feature recognition, feature models, STEP models.
DOI: https://doi.org/10.14733/cadaps.2022.624-661

1 INTRODUCTION

Many commercial CAD solutions provide feature-based CAD tools to model generic sheet metal parts. CATIA [1, 2] and NX 12 for Design [3] are two solutions that provide features that are specialized for and commonly used in aerospace sheet metal (ASM) part design, including curved flanges, joggles, and stringer cutouts, to name a few. The ASM models produced by commercial CAD solutions are not exchanged in their native format, but rather via their boundary representation (B-rep) using Standard for the Exchange of Product (STEP) model data [4].

However, the specialized features cannot be exchanged via STEP. An ASM feature-recognition method for STEP models—the B-rep model—could therefore significantly elevate the level of design information exchanged via STEP. This research project aims to propose an automated method for recognizing features from ASM B-rep models. Our overarching goal is to elevate the of information available for downstream applications such as 3D model difference identification [5]. This way, it becomes possible to indicate to the user, for example, how the length of a flange differs between two models.

Automated feature recognition (AFR) is an established way to abstract semantics from neutral CAD models. It dates back to the 1980s and has been investigated extensively since then [6]. AFR can be approached by rule-based methods (like syntactic pattern recognition, state transition diagrams and automata, logic rules and expert systems, the graph-based approach, the convex hull volumetric decomposition approach, the cell-based volumetric decomposition approach, the hint-based approach and the hybrid approach) or artificial neural network methods (like the graph-based approach, face coding, the contour-syntactic approach and volume decomposition) [6]. It seems that in the last decade AFR-related works have focused more on its application [7-12], optimization [13-16] and implementation [16, 17] than on proposing new methods. A rather inconspicuous application of AFR is sheet metal parts.

Although there is relatively little literature about AFR specific to sheet metal parts, there are quite a few noticeable and inspiring works [12, 18-27]. While a few sheet metal AFR methods take a comprehensive approach, such as the one proposed by Nnaji et al. [18], most can be categorized as being for shear features, generic deformation features or freeform sheet metal parts. Because freeform AFR methods [12, 26, 27] are not relevant to the scope of this paper, the previous works on them are not reviewed in detail here. The sheet metal AFR solutions that are currently on the market, i.e., FeatureWorks [28] and Sheet Metal Feature Recognition Library [29], are implementations of AFR methods for generic sheet metal deformation features. Our proposed AFR method [30] is unique as it was specialized for aerospace sheet metal part features.
Shear features are sheet metal part features that are created by shearing operations like blanking, notching, piercing and cutting off. AFR methods for shear features can be based on geometric and topological reasoning, like in the work of Jagirdar et al. [19]; profile offsetting for layout punching tool path specifications, like in the work of Devarajan et al. [20]; or using a center-plane model to calculate the shear layout, like in the work of Kannan and Shunmugam [21].

Sheet metal generic deformation features, on the other hand, result from either only deformation operations or shear and deformation operations. Hence, there is always a deformation footprint in their structures. Although the AFR methods proposed for sheet metal part deformation features are essentially conventional rule-based AFR methods, they have evolved independently. For example, Liu et al. [22] presented a fundamental study that addressed some of the main gaps in prior works, including feature intersection and array features. Feature intersection occurs when features intersect such that one is split or topologically impacted. An array feature is made up of repeated features. Kannan and Shunmugam [21, 23] also made two significant contributions to sheet metal AFR: 1) creating the method on STEP AP 203 that led to the promotion of its applicability, and 2) proposing and using a calculation algorithm for a center-plane model. The outcome of their proposal to use a center-plane model was twofold: it proved that a) a reduction in topographical information improved computational performance, and b) said reduction did not cripple the method. Prior to Kannan and Shunmugam, Jagirdar et al. [24] assumed that a sheet metal model is represented by its center-plane equivalent and accordingly proposed an AFR method. They considered "cross-bend" features (features that pass through a bend) as the only cases of feature intersection in sheet metal parts and proposed a technique to identify intersecting features. Subsequent studies omitted the problem of intersecting features entirely, making Jagirdar et al.'s work quite unique. And most recently, Gupta and Gurumoorthy [25] proposed a general solution for recognizing generic deformation features. In their paper, cylindrical, conical, spherical and toroidal faces are considered transitive entities to characterize deformation. This method is more geometrically general than the other methods set out in published works.

We proposed in [30] an AFR method for ASM parts that was inspired by the recent focus on AFR application in the literature. Features were identified by studying design guidelines and 168 different structural ASM fuselage and cockpit parts. A structural system is comprised of a thin-skinned shell that is stiffened by longitudinal stringers that are supported by transverse frames to form a semi-monocoque structure [31]. This structure is very efficient and has a high strength-to-weight ratio. Brake-formed and hydro-formed parts—the parts studied to propose the AFR method—are used to form frames, bulkheads, passenger and cargo floor structures [31], and cockpit components.

Implementation of the proposed AFR method is not detailed in the literature. Instead, only the programming language or solution, CAD platform, or kernel used in the implementation is mentioned. The C++ programming language seems to be most popular because of its strong object-oriented programming (OOP) package and the fact that it is the industry standard for developing 3D modeling solutions, which makes it possible to use it with many CAD platforms [9, 22] and graphics kernels [13, 14, 26, 27]. Table 1 lists the programming languages or solutions, CAD platforms, and kernels that have been used to implement AFR methods.
This template file contains all relevant information to process papers into the right format for the journal CAD and Applications. Please do not change the preset styles. Either type your paper directly into this file, substituting appropriate paragraphs and headings, or bring in your text in a plain format and then change it to the appropriate style by simply selecting the intended style from the menu in the MS-Word toolbar.

AFR implementation means	*References*
C language	[19, 24]
C++ language and ACIS™ kernel	[13, 14]
C++ language and Rhino CAD platform (and openNURBS functions)	[9]
C++ language and based on UGNX2.0 platform	[22]
C++ language and OpenGL kernel	[26]
C++ language and Open CASCADE kernel	[27]
Java language	[15-17]
MATLAB	[12, 25]

Table 1: AFR implementation means used in the literature.

In this paper, we describe the implementation and prototyping of our AFR method that we previously proposed for ASM parts in [30]. Some improvements were made to the original method, and the feature taxonomy was changed to match the one detailed in a subsequent paper proposing a semantic model difference identification method [5]. We provide details we believe will be helpful for similar future endeavors. In addition, the prototype is used to validate our AFR method by testing it with real-world samples.

2 DEFINITIONS AND METHODOLOGY

This section includes a short review of the definitions of the terms migrated from our earlier publications and illustrations and explanations of the 26 sample parts selected to validate the AFR method. These 26 parts cover all the features of all the parts studied in this research and are of various complexity levels. The main steps of the AFR method that were modified during implementation are briefly reviewed. Next, the data structure of the input STEP files, the intermediate enhanced B-rep elements manipulated to recognize features and the features are represented and explained. At the end, a few examples of the algorithms used to implement the main steps of the AFR method are explained in detail and represented in flowcharts.

2.1 Definitions

First, the definitions of the subtypes of the B-rep elements that were introduced in the original paper [30] are listed. Next, the definition of the features recognized by the original method as well as the features later modified in the work proposing the MDI method [5] are listed.

In the original method, it was detailed that faces are classified by their subtypes: trim_face, sheet_face, bend_face, internal_face, wall_face, connect_face and detained_face [30].

- The trim_faces are related to the faces of an ASM part that are created by trimming, and one of their dimensions is always equal to the part thickness.
- The sheet_faces are related to the faces of an ASM part that are not being trimmed, and they constitute the two sides of an ASM part.

The sheet_faces are primarily classified as web_faces, bend_faces, wall_faces, detained_faces and internal_faces.

- The planar sheet_faces with the two largest surface areas are called web_faces.
- The sheet_faces connecting web_faces and wall_faces via their face_outer_bounds are called bend_faces.
- The wall_faces are the sheet_faces that are connected to each other or to web_faces by bend_faces.
- The connect_faces are sheet_faces connecting bend_faces.
- The sheet_faces that are adjacent to web_faces and wall_faces via their non-face_outer_bounds are called internal_faces.
- The detained_faces are intermediate sheet_faces that are reclassified as connect_faces or wall_faces in the process of classifying sheet_faces by their subtypes.

In addition, edges are classified by their novel subtypes: trim_lin_edges, trim_nonlin_edges, untrim_lin_edges and untrim_nonlin_edges.

- A trim_lin_edge is an edge that is shared between any subtype of sheet_faces and a trim_face and is associated with a line.
- A trim_nonlin_edge is an edge that is shared between any subtype of sheet_faces and a trim_face and is associated with any subtype of curve other than a line.
- An untrim_lin_edge is an edge that is not shared with any trim_face and is associated with a line.
- An untrim_nonlin_edge is an edge that is not shared with any trim_face and is associated with any subtype of curve other than a line.

The face_bounds are also classified by their subtypes: face_outer_bound, bead_bound, internal_bound and hole_bound. The face_outer_bound are obtained from B-rep models.

- A face_bound that surrounds seven internal_faces is a bead_bound.
- A face_bound that is not a face_outer_bound and consists of edges associated with distinguishable curves is an internal_bound.
- A face_bound that consists of edges associated to undistinguishable curves is a hole_bound.

The features considered in this paper are web, cutout, hole, stringer cutout, bend relief, corner, lightening cutout, lightening hole, flange, lip, joggle, twin joggle, deformed flange, deformed web and bead.

- A web is the planar portion of an ASM part with the largest surface area.
- Cutouts are features formed by removing a portion of their parent feature, provided that the boundary of the parent feature is not changed.
- Holes are circular cutouts.
- Stringer cutouts are features formed by modifying the boundary of the parent feature and splitting flanges or the deformed flanges resulting from a twin joggle. Stringer cutouts are created to make room for a stringer to pass through.
- Bend reliefs are cutouts created to avoid immediate adjacency between flanges and facilitate manufacturing.
- Corners are features formed by rounding off sharp convex corners.

- Lightening cutouts and lightening holes are features created by removing a portion of the parent feature and forming stiffening lips at the boundary of the removed portion.
- Flanges are features materialized on the external boundary of their parent feature and are always the child of the web or another flange.
- Lips are features that are similar in shape to combined-open-immediate-stiffening flanges.
- Joggles and twin joggles are step-deformations that produce recesses on the web or flanges.
- The recessed portions are either deformed webs or deformed flanges depending on the parent feature of the joggle.
- Beads are features created by protruding a portion of the web.

The original taxonomy of the ASM part features was from a manufacturing perspective [30]. It was restructured to represent features' functionality, and a few features were also added [5]. Holes were divided into attachment holes and tooling holes, and flanges are divided into stiffening flanges and attachment flanges.

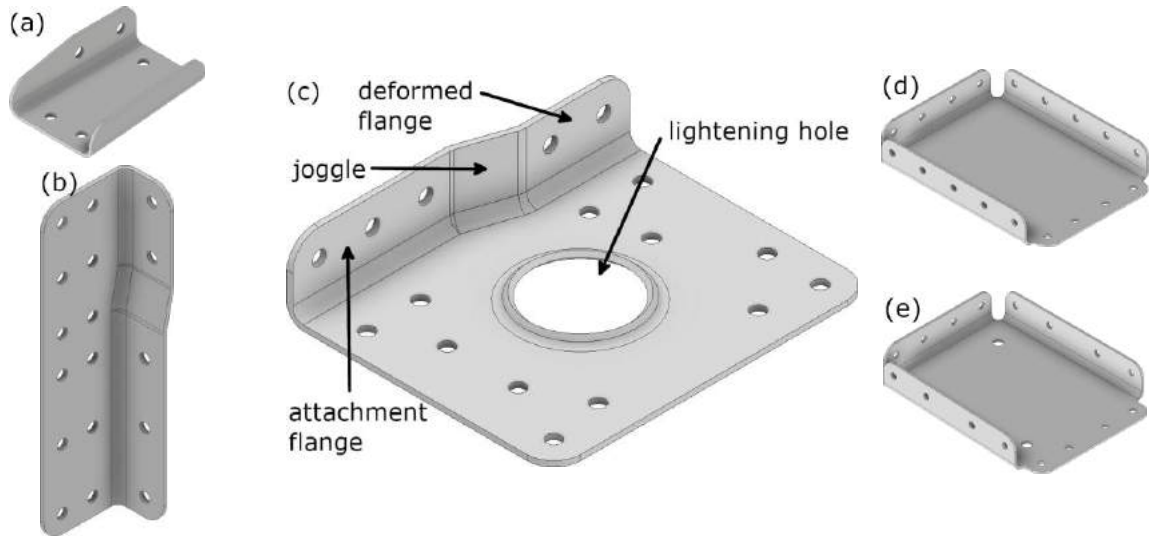
- Tooling holes are features created to facilitate manufacturing.
- Attachment holes are features created for attachment purposes. They are designed based on different design tables.
- Attachment flanges are flanges that have attachment holes.
- Stiffening flanges are flanges that do not attachment holes.

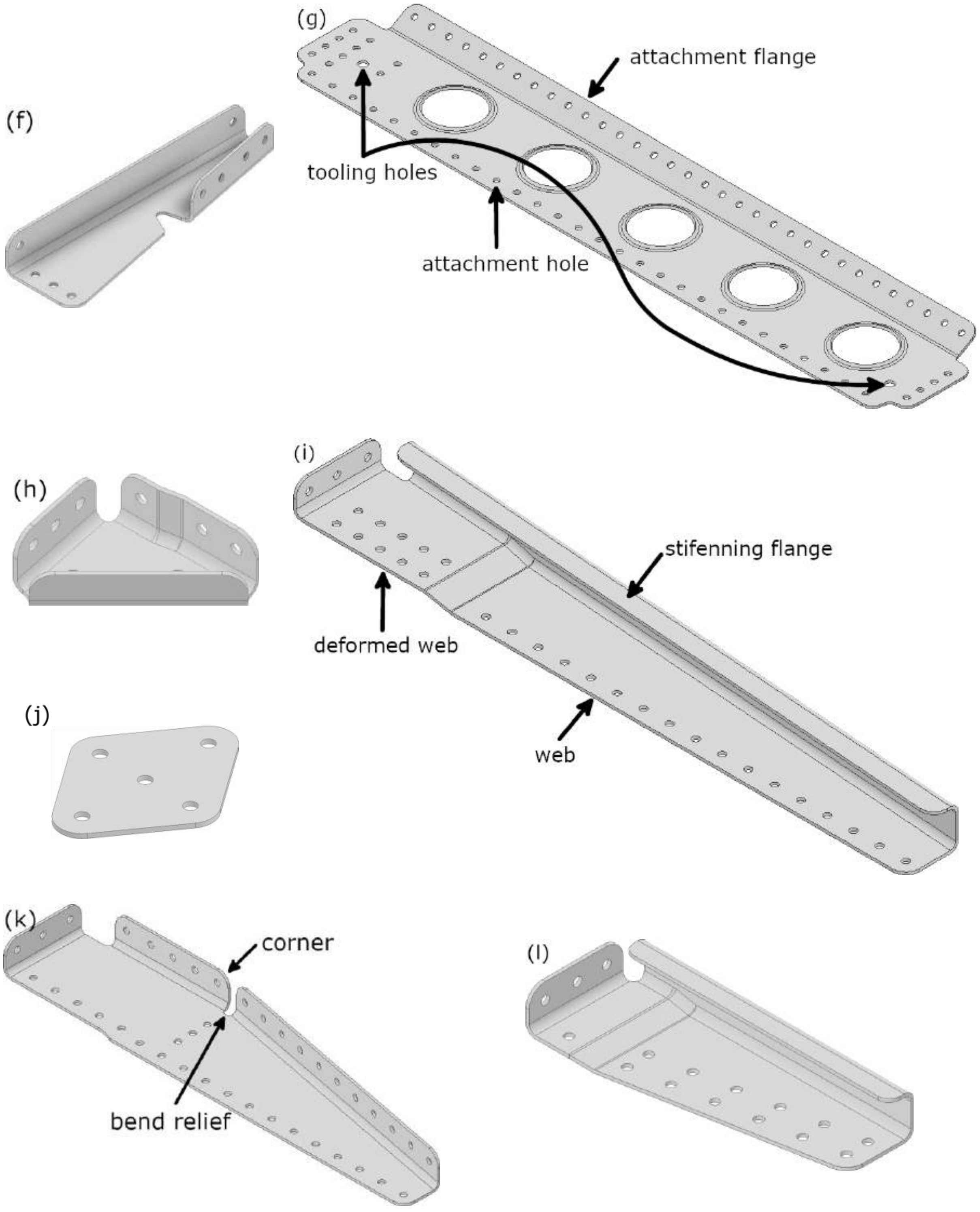
2.2 3D Models of Sample Parts

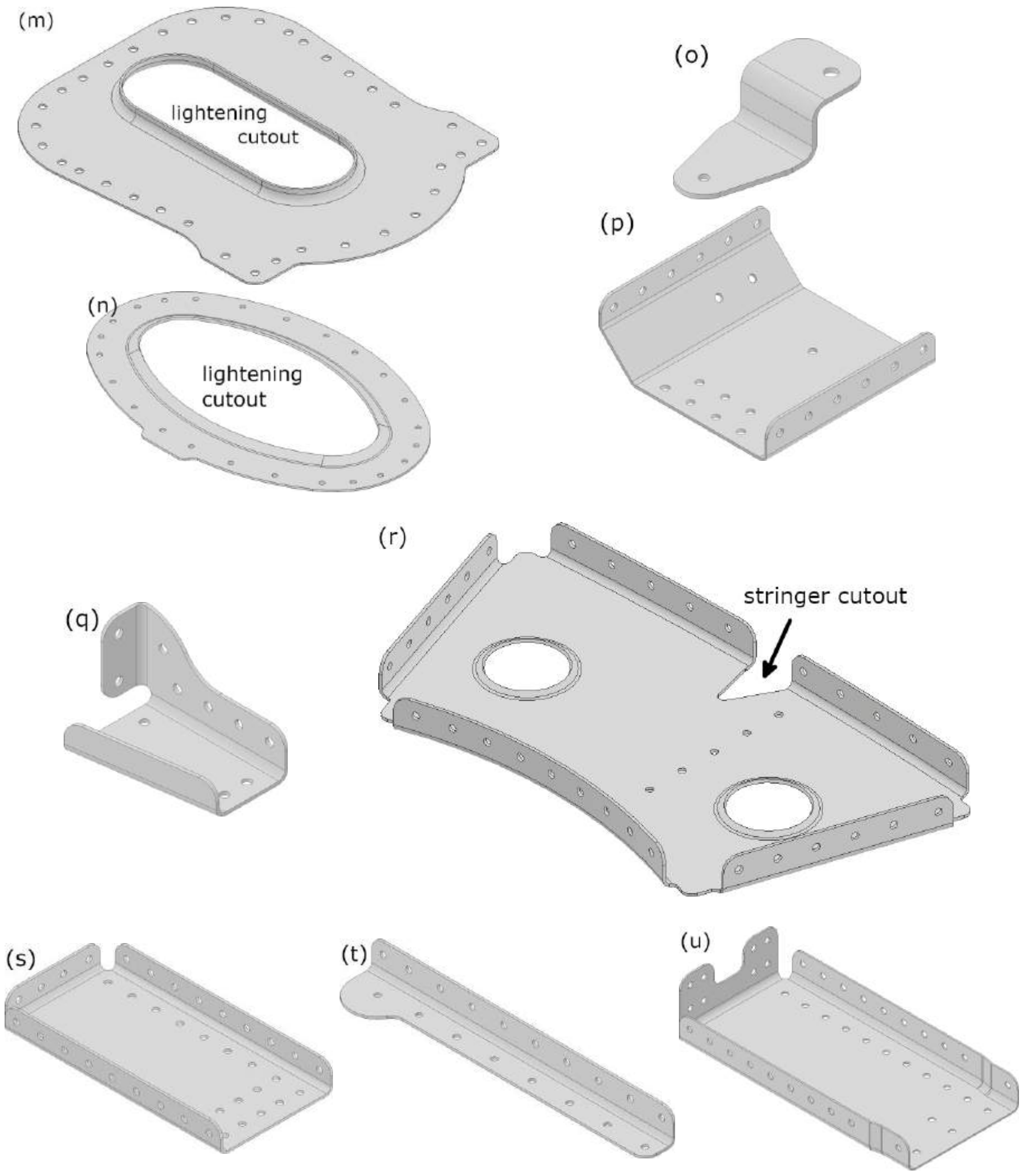
The sample ASM parts were chosen from a pool of parts belonging to frames, bulkheads, the floor and the cockpit of a Bombardier DHC-8-102. The parts were selected to satisfy the following criteria:

- Presence of all design features of ASM parts
- Presence of a broad spectrum of design complexity
- Presence of adequately similar designs to test output consistency

The native models were stored in STEP AP 242 files. Figure 1 shows the CAD models of the sample ASM parts.




(f)
(g)
attachment flange
tooling holes
attachment hole
(h)
(i)
stifenning flange
deformed web
web
(j)
(k)
corner
bend relief
(l)


(m)
lightening
cutout
(o)
(p)
(n)
lightening
cutout
(r)
stringer cutout
(q)
(s)
(t)
(u)

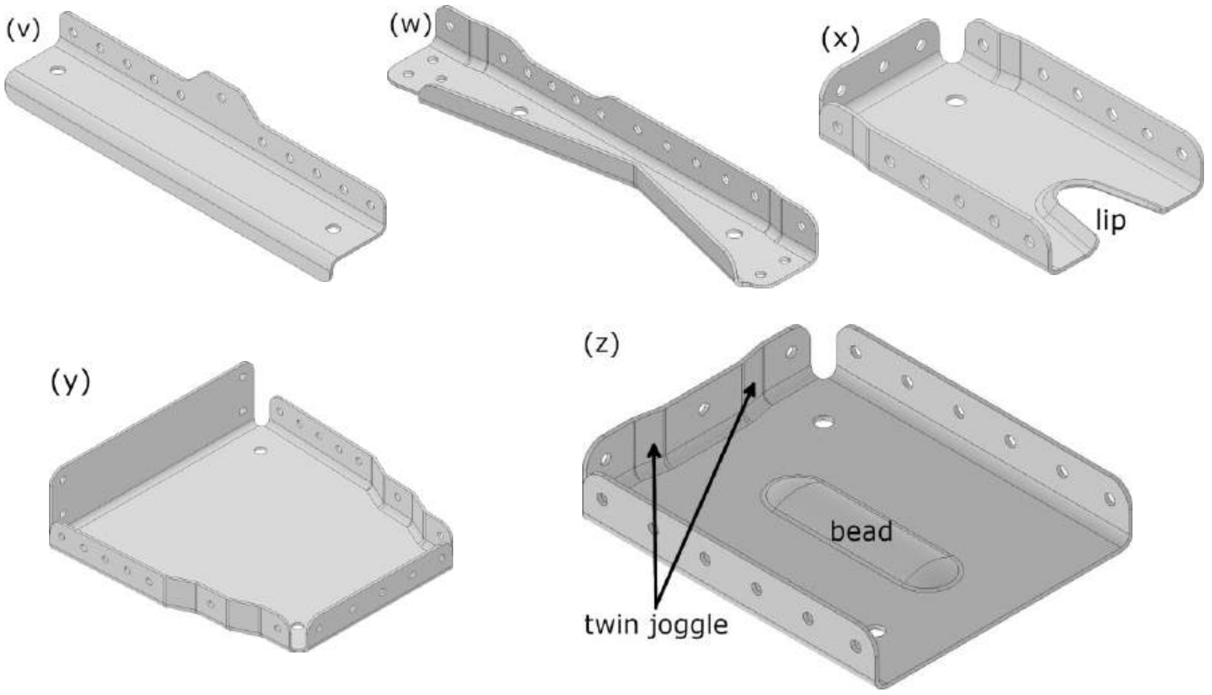


Figure 1: CAD models of the sample aerospace sheet metal parts.

2.3 Modified Steps of the AFR Method

The original AFR method was explained extensively in our previous paper [30]. Some modifications were made when implementing it in the prototype to either facilitate implementation or fix minor problems with the original method. Figure 2 highlights the major Steps 1 and 2 that were described in our previous paper and displays the steps implemented in the prototype with the modifications highlighted and denoted by the letters A to L. Also, each step whose algorithm is detailed in this paper is tagged with the number of the figure or letter of the appendix in which the algorithm can be found.

A. The B-rep elements are streamed from the STEP file and stored as C++ objects, which are themselves stored in lists.
B. To take advantage of the abstraction and encapsulation capabilities of object-oriented programming, the C++ objects storing the B-rep elements are enhanced by adding the geometric information to the topologic elements and directly connecting it to various B-rep elements via C++ pointers.
C. The web_faces are classified before sheet_faces and trim_faces are. This makes sense, as web_faces are sheet_faces.
D. The bend_faces and internal_faces adjacent to the web_faces are classified in one step.
E. Parts that do not have flanges, like those modeled in Figures 1 (j), 1 (m) and 1 (n), have all their sheet_faces classified by subtype (web_face and internal_face) by now and do not require step F. For those parts that have flanges and therefore have sheet_faces that have yet to be classified, step F is repeated until all their sheet_faces are classified by subtype.
F. The remaining sheet_face classification by subtype (wall_faces, detained_faced, bend_faces or internal_faces) continues until all sheet_faces are classified.

G. When classifying face_bounds, if the geometry of the non-lin edges is not identical, the hole_bounds are identified as internal_bounds. This change prevents incorrectly recognizing lightening cutouts like the one shown in Figure 1 (n) as lightening holes.
H. In the scope of this study, all joggles have faces that are adjacent to the web or a deformed web; therefore, joggles, deformed webs, flanges and deformed flanges are all recognized in one step performed after recognizing the web.
I. Although the stringer cutout recognition process is unchanged, bend reliefs are recognized between the web, deformed webs, flanges and deformed flanges, and are limited to having one non-lin edge between two of the above features.
J. A feature's ID and parent features are defined when it is recognized, and feature objects are instantiated accordingly. Its child features are also defined to complete its feature associations, and its parameters are calculated and instantiated to complete its definition.
K. Lips are features that have many similarities with flanges, so they are initially recognized as immediate-stiffening-open flanges. If any of said flanges have parameters that meet the standards defining lips, they are converted to lips.
L. In the last step, the flanges and holes are evaluated to classify them as attachment or stiffening flanges and attachment or tooling holes, respectively. Although these subtypes did not exist in the feature taxonomy of the original method, they are implemented to pave the way for the model difference identification applications intended for the output of this prototype. Attachment flanges and attachment holes carry design intentions that are pivotal for the semantic comparison of ASM parts.

At the end, the features are stored in a user-readable text file that we call a feature file. A couple of examples of feature files are presented in the results section.

2.4 Data Structures

When the B-rep elements are read from the STEP files, they are stored in the memory as C++ objects that have identical member attributes as the B-rep elements. These C++ objects are stored in lists implemented by C++ Vectors provided by the C++ Standard Template Library (STL). A vector is a sequence container that stores elements. C++ vectors are specifically used to work with dynamic data and can expand depending on the elements they contain. That makes them different from a fixed-size array. C++ vectors can automatically manage storage. Also, they are efficient when data is added and deleted often.

Figure 3 shows the class diagram representing the B-rep elements of the STEP files and their attributes and associations. Implementing the method based on the data structure of the STEP files' B-rep elements causes two issues: it is not possible to take full advantage of C++ class member methods, and objects must be repeatedly searched for.

To avoid these issues, the following changes were made to the data structure of the STEP files' B-rep elements:

1. The content of CartesianPoints, VertexPoints, Directions, Vectors, Axis2Placement3Ds, EdgeCurves and LoopEdges were added as member attributes of their associated B-rep elements.
2. B-splineCurveWithKnots, Circles, Ellipses and Lines were linked to OrientedEdges by raw pointers.
3. B-splineSurfaceWithKnots, BoundedB-splineSurfaces, Planes, CylindricalSurfaces, ToroidalSurfaces, SphericalSurfaces and ConicalSurfaces were linked to AdvancedFaces by raw pointers.
4. Both FaceOuterBounds and FaceBounds were stored as FaceBounds; however, FaceOuterBounds were signified by a true Boolean member attribute.
5. Faces, Bounds and Edges classes were added to manage AdvancedFaces, FaceBounds and OrientedEdges, respectively.

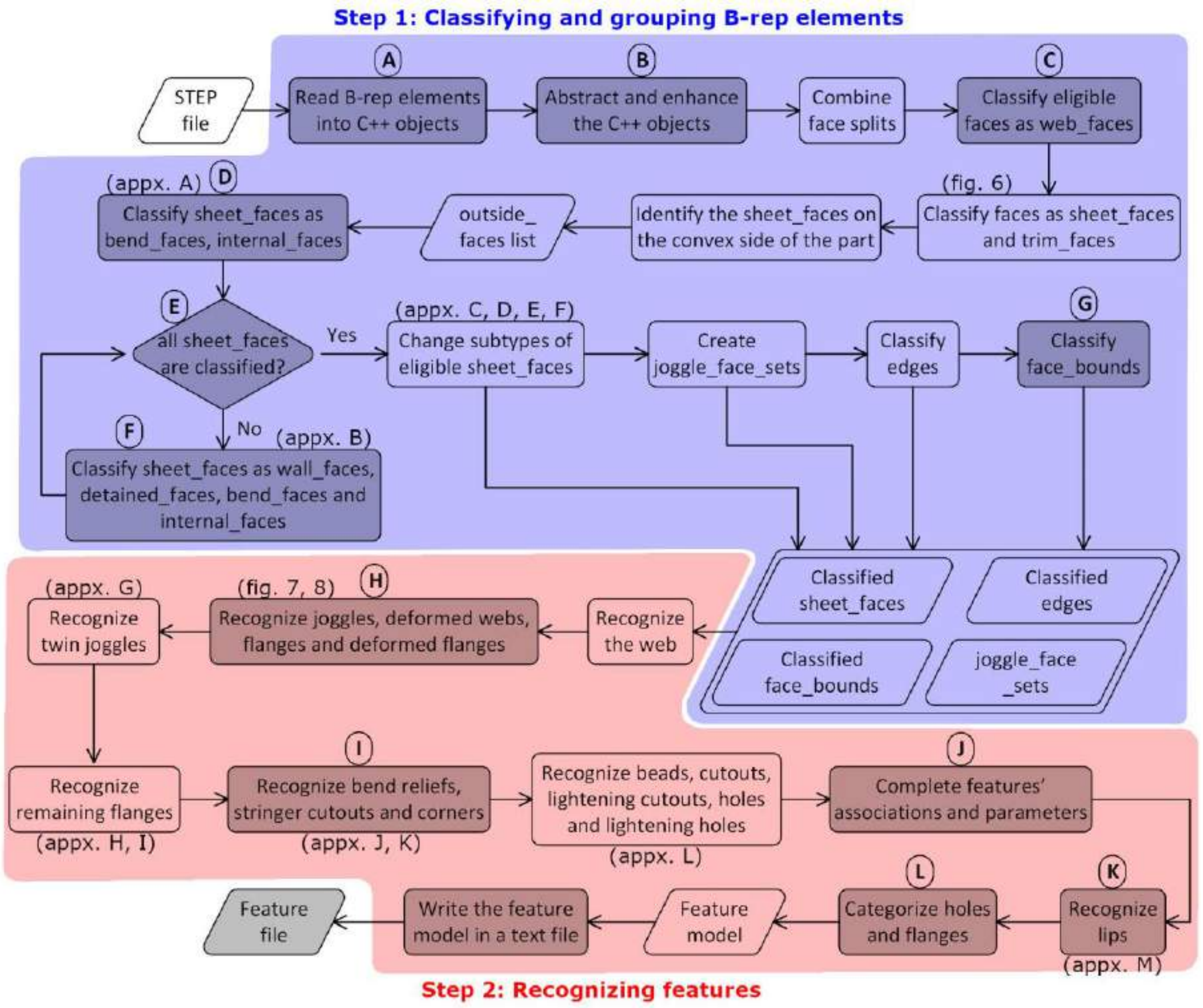

Step 1: Classifying and grouping B-rep elements
STEP file
A
Read B-rep elements into C++ objects
B
Abstract and enhance the C++ objects
Combine face splits
C
Classify eligible faces as web_faces
(fig. 6)
Classify faces as sheet_faces and trim_faces
Identify the sheet_faces on the convex side of the part
outside_ faces list
(appx. A) D
Classify sheet_faces as bend_faces, internal_faces
E
all sheet_faces are classified?
Yes
No
(appx. C, D, E, F)
Change subtypes of eligible sheet_faces
Create joggle_face_sets
Classify edges
G
Classify face_bounds
F
(appx. B)
Classify sheet_faces as wall_faces, detained_faces, bend_faces and internal_faces
Classified sheet_faces
Classified edges
Classified face_bounds
joggle_face _sets
(appx. G)
Recognize twin joggles
(fig. 7, 8) H
Recognize joggles, deformed webs, flanges and deformed flanges
Recognize the web
Recognize remaining flanges
(appx. H, I)
I
Recognize bend reliefs, stringer cutouts and corners
(appx. J, K)
Recognize beads, cutouts, lightening cutouts, holes and lightening holes
(appx. L)
J
Complete features' associations and parameters
Feature file
Write the feature model in a text file
Feature model
L
Categorize holes and flanges
K
Recognize lips
(appx. M)
Step 2: Recognizing features

Figure 2: Flowchart of the modified method implemented in the prototype.

Figure 4 shows the class diagram representing the further abstracted and enhanced data structure. This solution allows operations related to Edges, Bounds and Faces to be carried out via the classes method, which is more compliant with the abstraction and encapsulation basis of OOP. Another reason for this solution is to manage memory issues related to the use of raw pointers. When two objects (A and C) are pointing to another object (B), if object A is deleted (for example, local variable no longer in use), object B will also be deleted, and object C will have what is called a "dangling pointer" because it is pointing to a memory address (object B) that is no longer valid. This can cause several problems such as false results and lost data. C++ makes it possible to manage the lifetime of objects and change their default behavior.

When using raw pointers, it is best practice to allocate the memory dynamically using the "new" operator, and explicitly release the memory using "delete" and delete the object when the object is no longer in use. However, in the data structure proposed in this work, AdvancedFaces, OrientedEdges and FaceBounds are shared between different objects. For example, a pointer to a FaceBound is shared between an OrientedEdge and an AdvancedFace, so it is crucial to manage the memory and decide when each object should be deleted.

Creating classes to represent the edges_list, bounds_list and faces_list makes it easier to manage objects' lifetime, since in C++, each object is created and deleted using special methods (class member functions) that are called constructors and destructors, respectively. Each class has a default constructor and destructor, so customizing these two methods makes it easy to manage the memory.

Figure 5 shows the class diagram representing the features and their structure as well as their parameters (like position or profile) that are implemented in C++ classes. In the proposed feature data structure, all the features are inherited from a "Feature" class. This makes it possible to use one code to define the behavior that is common to all the features. The "Feature" class is defined as an abstract class. An abstract class is a class that cannot be instantiated; it is only used as a base class in inheritance hierarchies. C++ abstract classes are defined by pure virtual functions. A pure virtual function is a virtual function that has no implementation. For example, the following feature_type function (the Feature class's member function) is a pure virtual function:

```
Virtual std::string feature_type() const = 0;
```

This virtual function returns the feature type, for example, "Flange" or "Joggle", and doesn't have an implementation in the base class, since the base class "Feature" is an abstract class. There is no "feature" feature type, so it makes sense that the feature_type function is not implemented for the Feature class. Since, the Feature class serves as a base class from which all the other features are derived, it defines the behavior that is common to all the features.

2.5 The Algorithms

Here, we describe the main ideas in the algorithms designed to perform each step in classifying faces and recognizing features. The algorithms to classify boundaries and edges are not discussed due to their simplicity. The algorithm to identify sheet_faces and trim_faces is described in detail and illustrated in Figure 6. The algorithm to recognize joggles on the web, on deformed webs and on flanges is described in detail and illustrated in Figures 7 and 8. In the case of joggles on flanges, the algorithm also recognizes their related flanges and deformed flanges. The rest of the algorithms for further classifying faces and recognizing the other types of flanges are listed in Appendices A to M.

Identifying sheet_faces and trim_faces starts from one of the web_faces. The web_faces are the planar faces of the B-rep model with the largest surface area. Each web_face is stored in a list called output1 or output2, both of which are instances of the Faces class. First, the normal vectors of the web_face, which is initially the only face in the list, and each of its adjacent faces at a shared point are evaluated for codirectionality. If the normal vectors of a face and an adjacent face at a shared point are not codirectional, the adjacent face is classified as a trim_face and the next adjacent face is evaluated. If, however, the normal vectors are codirectional and the adjacent face is not already in the list of faces, it is classified as a sheet_face and added to the list. The sheet_face and trim_face identification function is recursive. As a result, if a face is classified as a sheet_face, before moving on to the next adjacent face, its own adjacent faces are evaluated for codirectionality between their normal vectors at a shared point. Figure 6 shows a detailed representation of the sheet_face and trim_face identification algorithm.

Most of the feature recognition algorithms start from a list of faces that are evaluated to check a few conditions and conclude whether or not a feature exists. These lists of faces could be the list of faces on the outside of the part (called outside_faces), joggle_face_sets, or faces linked to the shape of features. The outside_faces could be output1 or output2, depending on the summation of faces' surface area of which one is larger than the other one. For example, the algorithm to recognize joggles and their related parent or child features (illustrated in Figure 7) starts with a joggle_face_set. (The joggle_face_sets and their identification are detailed in [30].) Then, the bend_faces in the joggle_face_set are checked to find out if any of them is adjacent to the web_face, to assign true to a Boolean variable called cond1.

The connect_faces in the joggle_face_set are also checked to find out if any of them is adjacent to the web_face, to assign false to a Boolean variable called cond2. If cond1 is true and cond2 is false, the joggle must be on the web, and if both cond1 and cond2 are true, the joggle must be on a flange.

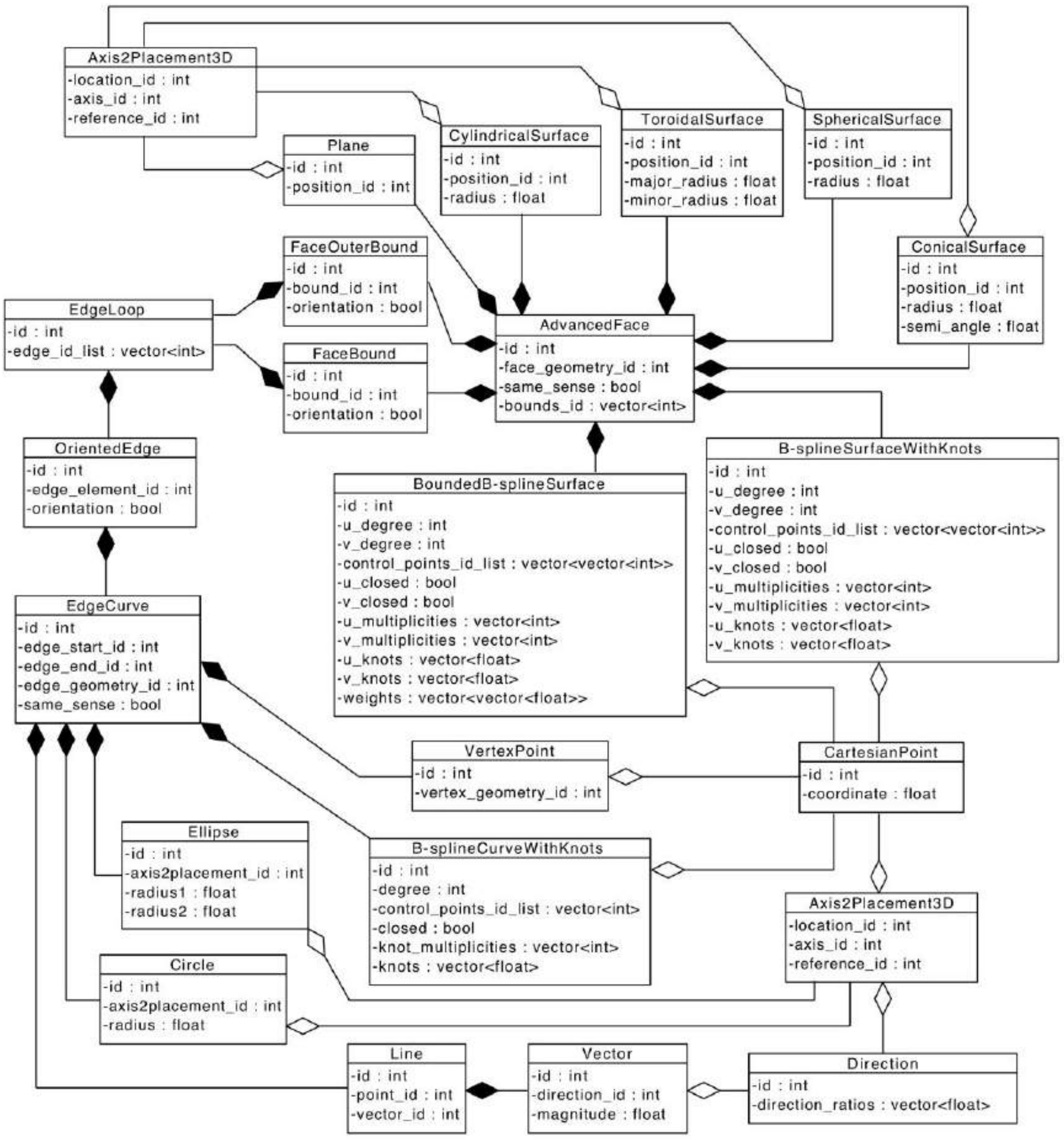


Figure 3: Class diagram representing the B-rep elements of the STEP files.

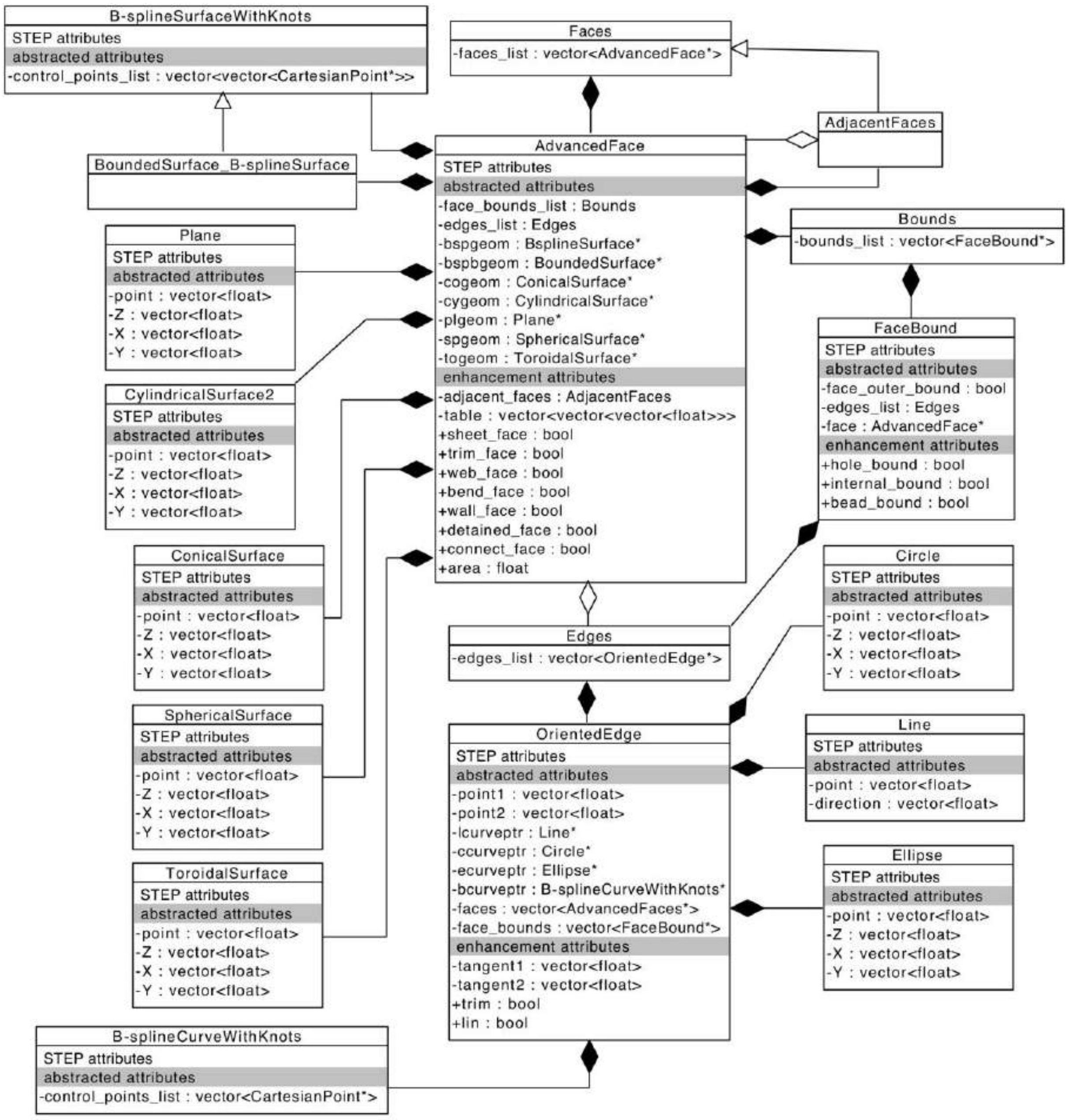


Figure 4: Class diagram representing the further abstracted and enhanced data structure.

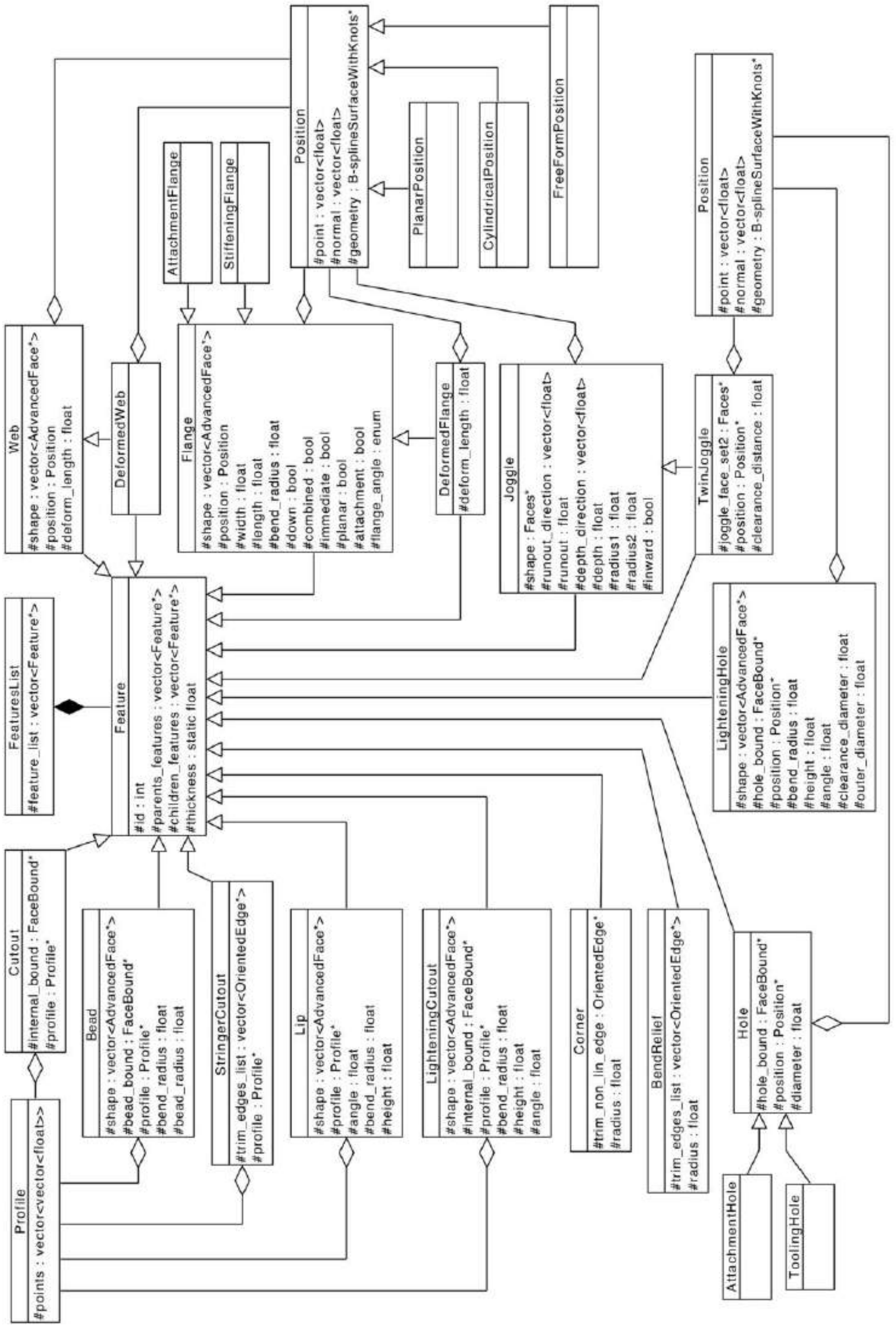


Figure 5: Class diagram representing the features and their data structure.

- If the joggle is on the web, the joggle is created with its associated sheet_faces. Then, each of the wall_faces in the outside_faces list is checked to find the one that is adjacent to one of the bend_faces and none of the connect_faces in the joggle_face_set. When found, the deformed web is created with its associated sheet_faces.
- If the joggle is on a flange, the joggle is created with its associated sheet_faces. Then, two temporary flanges are created. Note that this part of the algorithm is represented quite abstractly in the highlighted steps in Figure 7 to save some space and then detailed in a zoomed-in view in Figure 8. Next, the deformed temporary flange is identified and the deformed flange is created based on the temporary flange.

To recognize the temporary flanges, the algorithm represented in Figure 8 first checks the sheet_faces in the joggle_face_set to find the bend_faces. Then, the bend_faces are checked to find their adjacent wall_faces that are not in the joggle_face_set and not adjacent to any of the connect_faces in the joggle_face_set. When any of those wall_faces are found, a flange is created and then its faces are added to it. Note that the parent-child relationships between features are identified based on the relationships between the topological elements used in recognizing them. For example, a deformed web is linked to its parent joggle based on the adjacency of the wall_face used in its recognition and the bend_face of the parent joggle. Similarly, a hole is linked to a flange or web based on the topological relationships between the hole_bound and wall_face or web_face.

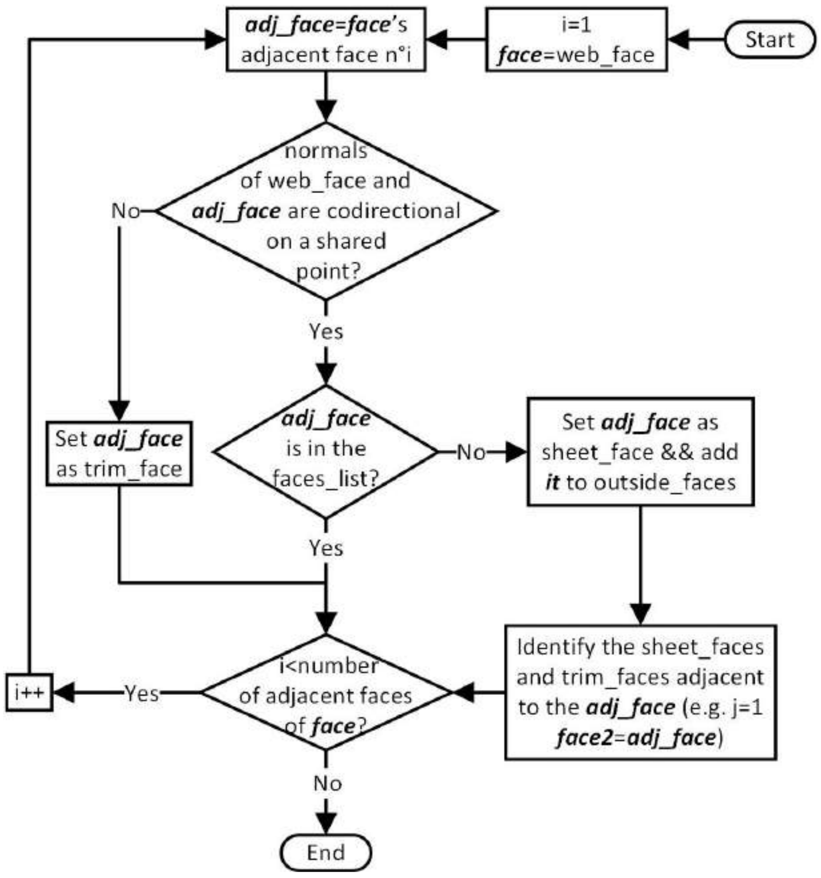

Start
i=1
face=web_face
adj_face=face's adjacent face n°i
normals of web_face and adj_face are codirectional on a shared point?
No
Yes
Set adj_face as trim_face
adj_face is in the faces_list?
No
Set adj_face as sheet_face && add it to outside_faces
Yes
Identify the sheet_faces and trim_faces adjacent to the adj_face (e.g. j=1 face2=adj_face)
i<number of adjacent faces of face?
Yes
i++
No
End

Figure 6: Flowchart representing the algorithm to identify sheet_faces and trim_faces.

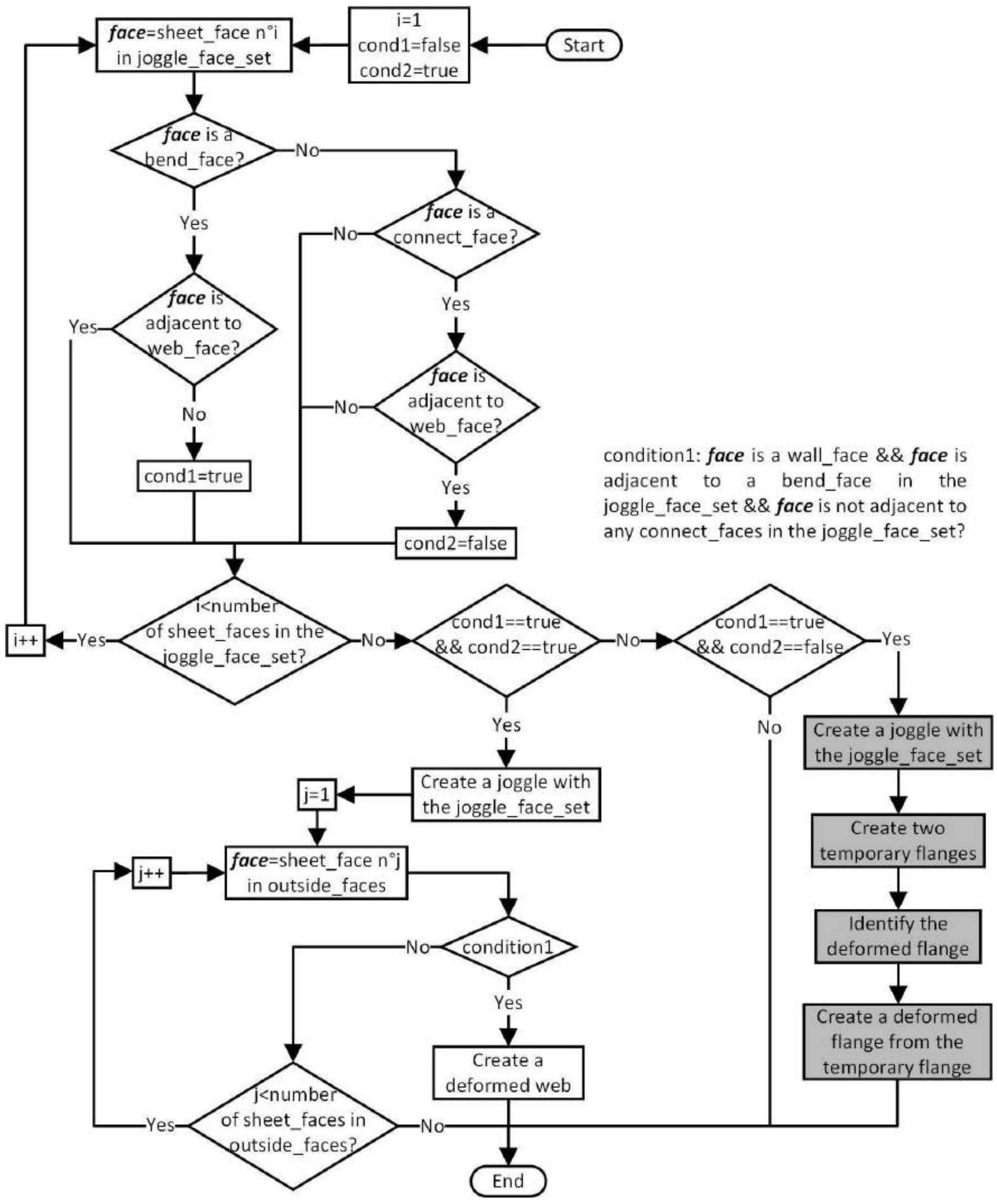

Start
i=1
cond1=false
cond2=true
face=sheet_face n°i in joggle_face_set
face is a bend_face?
No
Yes
face is a connect_face?
face is adjacent to web_face?
Yes
No
cond1=true
face is adjacent to web_face?
cond2=false
condition1: face is a wall_face && face is adjacent to a bend_face in the joggle_face_set && face is not adjacent to any connect_faces in the joggle_face_set?
i++
i<number of sheet_faces in the joggle_face_set?
cond1==true && cond2==true
cond1==true && cond2==false
Create a joggle with the joggle_face_set
Create two temporary flanges
Identify the deformed flange
Create a deformed flange from the temporary flange
j=1
Create a joggle with the joggle_face_set
j++
face=sheet_face n°j in outside_faces
condition1
Create a deformed web
j<number of sheet_faces in outside_faces?
End

Figure 7: Flowchart representing the algorithm to recognize joggles on the web, on deformed webs and on flanges as well as the flanges and deformed flanges related to joggles on flanges.

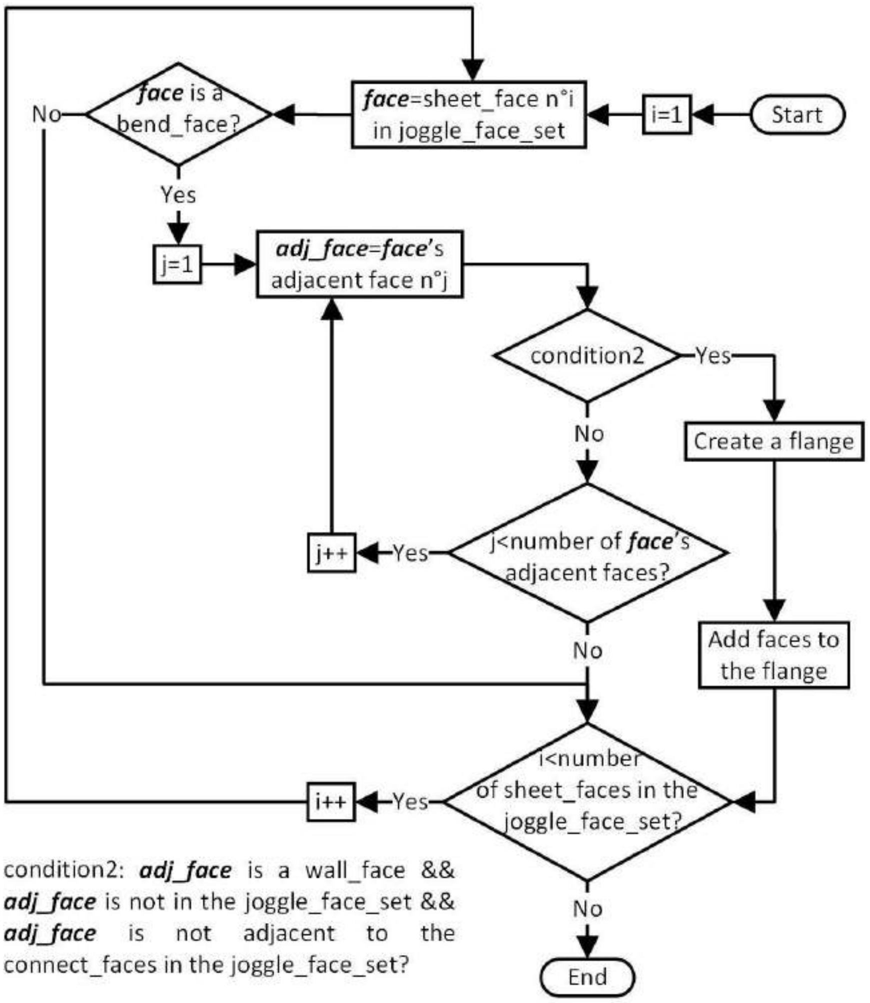
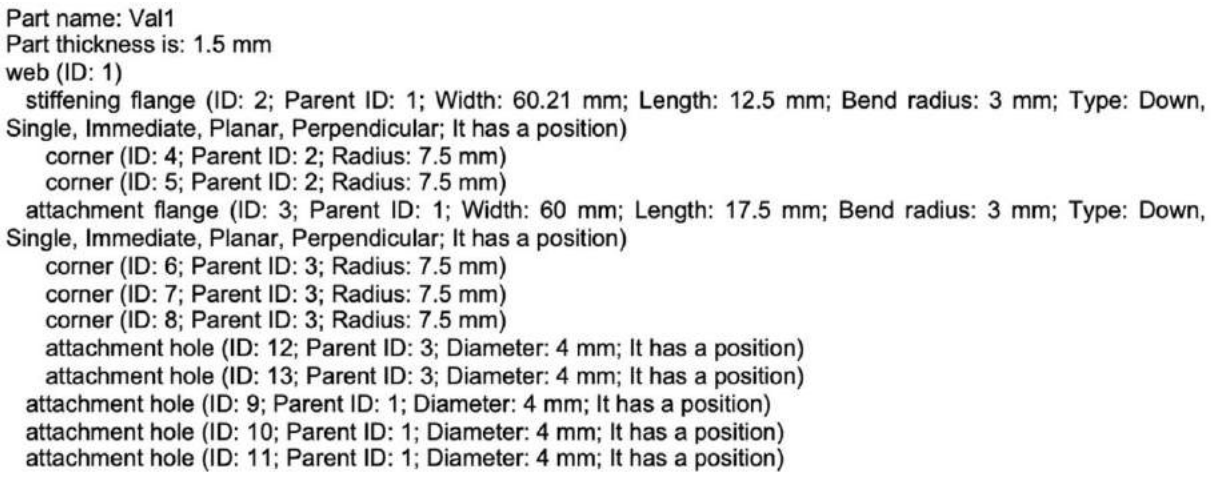

Start
i=1
face=sheet_face n°i in joggle_face_set
face is a bend_face?
No
Yes
j=1
adj_face=face's adjacent face n°j
condition2
Yes
No
Create a flange
j<number of face's adjacent faces?
Yes
j++
No
Add faces to the flange
i<number of sheet_faces in the joggle_face_set?
Yes
i++
No
End
condition2: adj_face is a wall_face && adj_face is not in the joggle_face_set && adj_face is not adjacent to the connect_faces in the joggle_face_set?

Figure 8: Flowchart representing the algorithm to create joggles and temporary flanges and recognize the deformed flange from the temporary flanges (a zoomed-in view of the highlighted steps in Figure 7).

3 RESULTS

The output of the prototype is a text file that is called a feature file. The feature file starts with the name of the part, followed by its thickness and then its features. The features are structured by parent features immediately followed by their children and are further indented as their level in the feature hierarchy increases. Each feature is defined by its name, ID, Parent ID (except the web) and parameters. For example, Figure 9 shows the content of the feature file for the part represented in Figure 1 (a).



```
Part name: Val1
Part thickness is: 1.5 mm
web (ID: 1)
  stiffening flange (ID: 2; Parent ID: 1; Width: 60.21 mm; Length: 12.5 mm; Bend radius: 3 mm; Type: Down,
Single, Immediate, Planar, Perpendicular; It has a position)
    corner (ID: 4; Parent ID: 2; Radius: 7.5 mm)
    corner (ID: 5; Parent ID: 2; Radius: 7.5 mm)
  attachment flange (ID: 3; Parent ID: 1; Width: 60 mm; Length: 17.5 mm; Bend radius: 3 mm; Type: Down,
Single, Immediate, Planar, Perpendicular; It has a position)
    corner (ID: 6; Parent ID: 3; Radius: 7.5 mm)
    corner (ID: 7; Parent ID: 3; Radius: 7.5 mm)
    corner (ID: 8; Parent ID: 3; Radius: 7.5 mm)
    attachment hole (ID: 12; Parent ID: 3; Diameter: 4 mm; It has a position)
    attachment hole (ID: 13; Parent ID: 3; Diameter: 4 mm; It has a position)
  attachment hole (ID: 9; Parent ID: 1; Diameter: 4 mm; It has a position)
  attachment hole (ID: 10; Parent ID: 1; Diameter: 4 mm; It has a position)
  attachment hole (ID: 11; Parent ID: 1; Diameter: 4 mm; It has a position)
```

Figure 9: The content of the feature file for the part represented in Figure 1 (a).

Note that if a feature needs a position or profile to be defined and the position or profile is successfully obtained from the B-rep model, "It has a position" or "It has a profile" holds the place of the position or profile information. Otherwise, "It does NOT have a position" or "It does NOT have a profile" indicates there is no position or profile information for the feature. The position and profile information is quite low-level data, described in the class model represented in Figure 5, and therefore not included in the content of feature files, which are supposed to be user readable.

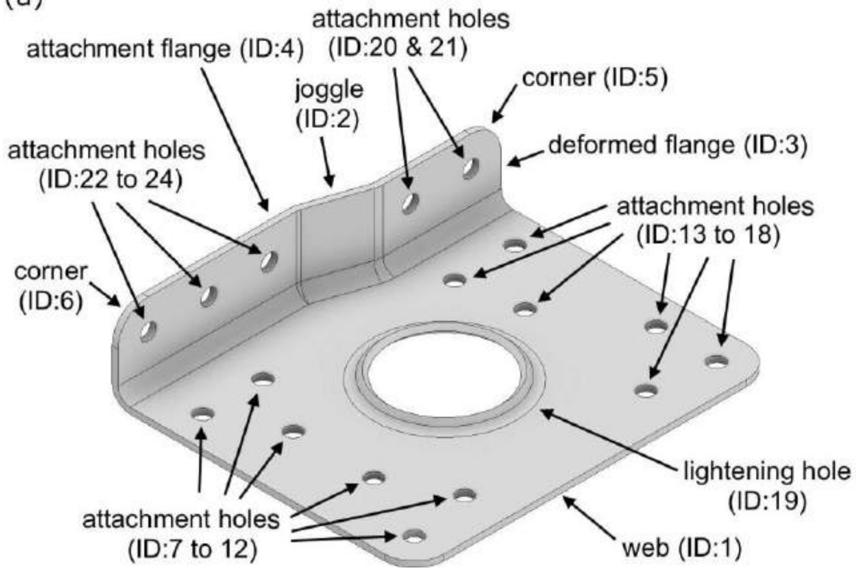
We have selected three parts to represent the prototype's results. Each of these parts has a number of features, some of which highlight interesting points about the prototype's output and are therefore explained. In Figures 10, 11 and 12, each part is displayed annotated with its features and followed by its feature file content.

To start with, Figure 10 (a) shows a rather typical design of a hydroformed ASM part that is annotated with its features. It has a lightening hole, attachment holes on the web, and an immediate flange deformed by a joggle with several attachment holes on the attachment flange and deformed flange. Figure 10 (b) shows the content of the part's feature file with "~ ~ ~" being used to avoid a long list of features having the same parameters.

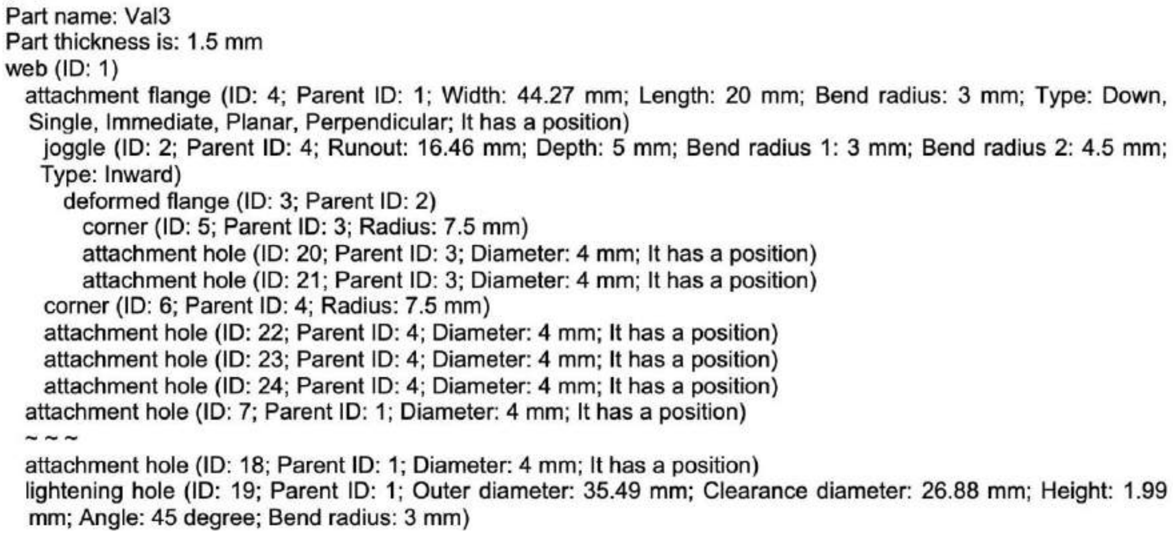
Figure 11 (a) shows another rather common design of a hydroformed ASM part that is annotated with its features. It has tooling and attachment holes on the web, and immediate flanges with several attachment holes on them. Due to the arrangement of the flanges, there are bend reliefs between them. The corners are not annotated in Figure 11 (a) for simplicity. Figure 11 (b) shows the content of the part's feature file with "~ ~ ~" being used to avoid a long list of features having the same parameters.

Figure 12 (a) shows a less common, more complex design of a hydroformed ASM part that is annotated with its features. It has attachment holes on the web, which is deformed by a joggle, and an immediate flange with several attachment holes and a stiffening flange on it. Due to the arrangement of the immediate flanges on the web and the deformed web, there is a bend relief between them. The corners are not annotated in Figure 12 (a) for simplicity. Figure 12 (b) shows the content of the part's feature file with "~ ~ ~" being used to avoid a long list of features having the same parameters.

(a)


attachment holes (ID:20 & 21)
attachment flange (ID:4)
joggle (ID:2)
corner (ID:5)
attachment holes (ID:22 to 24)
deformed flange (ID:3)
attachment holes (ID:13 to 18)
corner (ID:6)
lightening hole (ID:19)
attachment holes (ID:7 to 12)
web (ID:1)

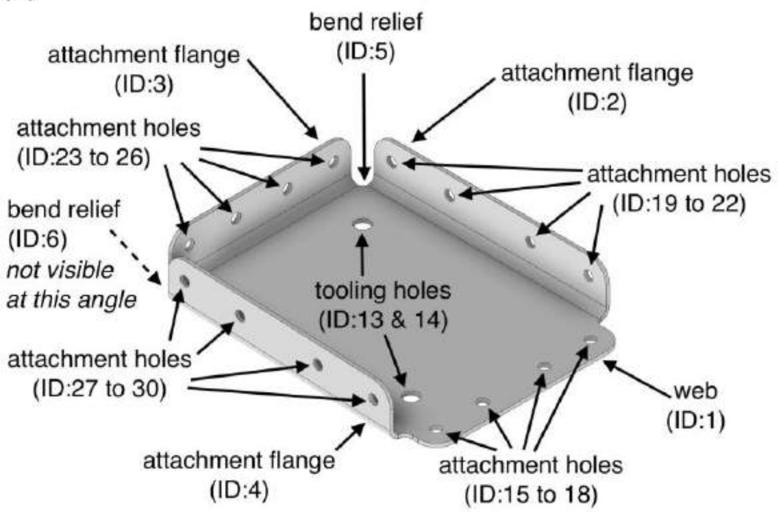
(b)



```
Part name: Val3
Part thickness is: 1.5 mm
web (ID: 1)
  attachment flange (ID: 4; Parent ID: 1; Width: 44.27 mm; Length: 20 mm; Bend radius: 3 mm; Type: Down,
  Single, Immediate, Planar, Perpendicular; It has a position)
    joggle (ID: 2; Parent ID: 4; Runout: 16.46 mm; Depth: 5 mm; Bend radius 1: 3 mm; Bend radius 2: 4.5 mm;
    Type: Inward)
      deformed flange (ID: 3; Parent ID: 2)
        corner (ID: 5; Parent ID: 3; Radius: 7.5 mm)
        attachment hole (ID: 20; Parent ID: 3; Diameter: 4 mm; It has a position)
        attachment hole (ID: 21; Parent ID: 3; Diameter: 4 mm; It has a position)
    corner (ID: 6; Parent ID: 4; Radius: 7.5 mm)
    attachment hole (ID: 22; Parent ID: 4; Diameter: 4 mm; It has a position)
    attachment hole (ID: 23; Parent ID: 4; Diameter: 4 mm; It has a position)
    attachment hole (ID: 24; Parent ID: 4; Diameter: 4 mm; It has a position)
  attachment hole (ID: 7; Parent ID: 1; Diameter: 4 mm; It has a position)
  ~ ~ ~
  attachment hole (ID: 18; Parent ID: 1; Diameter: 4 mm; It has a position)
  lightening hole (ID: 19; Parent ID: 1; Outer diameter: 35.49 mm; Clearance diameter: 26.88 mm; Height: 1.99
  mm; Angle: 45 degree; Bend radius: 3 mm)
```

Figure 10: A rather typical design of a hydroformed aerospace sheet metal part annotated with its features (a) as well as the content of its feature file (b).

(a)

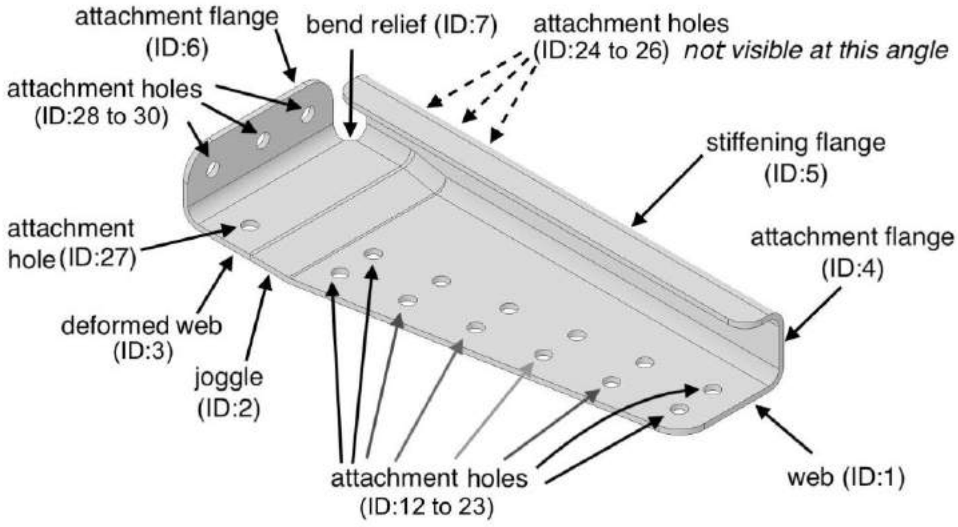

bend relief (ID:5)
attachment flange (ID:3)
attachment flange (ID:2)
attachment holes (ID:23 to 26)
attachment holes (ID:19 to 22)
bend relief (ID:6) *not visible at this angle*
tooling holes (ID:13 & 14)
attachment holes (ID:27 to 30)
web (ID:1)
attachment flange (ID:4)
attachment holes (ID:15 to 18)

(b)

```
Part name: Val5
Part thickness is: 1.5 mm
web (ID: 1)
  attachment flange (ID: 2; Parent ID: 1; Width: 100 mm; Length: 20 mm; Bend radius: 3 mm; Type: Down, Single,
  Immediate, Planar, Perpendicular; It has a position)
    bend relief (ID: 5; Parent IDs: 2, 3; Radius: 3 mm)
    corner (ID: 7; Parent ID: 2; Radius: 7.5 mm)
    corner (ID: 8; Parent ID: 2; Radius: 7.5 mm)
    attachment hole (ID: 19; Parent ID: 2; Diameter: 4 mm; It has a position)
    ~ ~ ~
    attachment hole (ID: 22; Parent ID: 2; Diameter: 4 mm; It has a position)
  attachment flange (ID: 3; Parent ID: 1; Width: 77.14 mm; Length: 20 mm; Bend radius: 3 mm; Type: Down,
  Single, Immediate, Planar, Perpendicular; It has a position)
    bend relief (ID: 5; Parent IDs: 2, 3; Radius: 3 mm)
    bend relief (ID: 6; Parent IDs: 3, 4; Radius: 3 mm)
    corner (ID: 9; Parent ID: 3; Radius: 7.5 mm)
    corner (ID: 10; Parent ID: 3; Radius: 7.5 mm)
    attachment hole (ID: 23; Parent ID: 3; Diameter: 4 mm; It has a position)
    ~ ~ ~
    attachment hole (ID: 26; Parent ID: 3; Diameter: 4 mm; It has a position)
  attachment flange (ID: 4; Parent ID: 1; Width: 100 mm; Length: 20 mm; Bend radius: 3 mm; Type: Down, Single,
  Immediate, Planar, Perpendicular; It has a position)
    bend relief (ID: 6; Parent IDs: 3, 4; Radius: 3 mm)
    corner (ID: 11; Parent ID: 4; Radius: 7.5 mm)
    corner (ID: 12; Parent ID: 4; Radius: 7.5 mm)
    attachment hole (ID: 27; Parent ID: 4; Diameter: 4 mm; It has a position)
    ~ ~ ~
    attachment hole (ID: 30; Parent ID: 4; Diameter: 4 mm; It has a position)
  tooling hole (ID: 13; Parent ID: 1; Diameter: 6 mm; It has a position)
  tooling hole (ID: 14; Parent ID: 1; Diameter: 6 mm; It has a position)
  attachment hole (ID: 15; Parent ID: 1; Diameter: 4 mm; It has a position)
  ~ ~ ~
  attachment hole (ID: 18; Parent ID: 1; Diameter: 4 mm; It has a position)
```

Figure 11: A rather common design of a hydroformed aerospace sheet metal part annotated with its features (a) as well as the content of its feature file (b).

(a)



(b)

```
Part name: Val12
Part thickness is: 1.5 mm
web (ID: 1; Deformation length: 99.55 mm)
  joggle (ID: 2; Parent ID: 1; Runout: 10.74 mm; Depth: 2 mm; Bend radius 1: 3 mm; Bend radius 2: 3 mm; Type:
  Inward)
    deformed web (ID: 3; Parent ID: 2)
      attachment flange (ID: 6; Parent ID: 3; Width: 43.5 mm; Length: 18 mm; Bend radius: 3 mm; Type: Down,
      Single, Immediate, Planar, Perpendicular; It has a position)
        bend relief (ID: 7; Parent IDs: 6, 4; Radius: 3 mm)
        corner (ID: 10; Parent ID: 6; Radius: 7.5 mm)
        corner (ID: 11; Parent ID: 6; Radius: 7.5 mm)
        attachment hole (ID: 28; Parent ID: 6; Diameter: 4 mm; It has a position)
        attachment hole (ID: 29; Parent ID: 6; Diameter: 4 mm; It has a position)
        attachment hole (ID: 30; Parent ID: 6; Diameter: 4 mm; It has a position)
      attachment hole (ID: 27; Parent ID: 3; Diameter: 4 mm; It has a position)
  attachment flange (ID: 4; Parent ID: 1; Width: 123.5 mm; Length: 20 mm; Bend radius: 3 mm; Type: Down,
  Single, Immediate, Planar, Perpendicular; It has a position)
    stiffening flange (ID: 5; Parent ID: 4; Width: 123.5 mm; Length: 11.5 mm; Bend radius: 3 mm; Type: Down,
    Single, Return, Planar, Perpendicular; It has a position)
      corner (ID: 8; Parent ID: 5; Radius: 5 mm)
      corner (ID: 9; Parent ID: 5; Radius: 5 mm)
    bend relief (ID: 7; Parent IDs: 6, 4; Radius: 3 mm)
    attachment hole (ID: 24; Parent ID: 4; Diameter: 4 mm; It has a position)
    attachment hole (ID: 25; Parent ID: 4; Diameter: 4 mm; It has a position)
    attachment hole (ID: 26; Parent ID: 4; Diameter: 4 mm; It has a position)
  attachment hole (ID: 12; Parent ID: 1; Diameter: 4 mm; It has a position)
  ~ ~ ~
  attachment hole (ID: 23; Parent ID: 1; Diameter: 4 mm; It has a position)
```

Figure 12: A less common design of a hydroformed aerospace sheet metal part annotated with its features (a) as well as the content of its feature file (b).

4 DISCUSSION AND CONCLUSION

A factor that made prototyping challenging was that the original proposed method was for a set of unprecedented features, ASM part features. This made the development work unique and hence necessitated creativity in solving problems. Of all the ASM part features, joggle was the main differentiator between the original method and similar literature. It was also pivotal to the development effort. The uniqueness of the original method's scope and by extension the development effort motivated the authors to present this work at a level of detail not seen in the existing literature.

In this work, we represented the successful implementation of an AFR method for ASM parts. The AFR method was implemented by first developing a data structure for the B-rep elements and a data structure for feature definitions. Then, algorithms were developed for classifying B-rep elements and recognizing features. When implementing the AFR method, certain steps in the originally proposed method [30] were changed to correct errors or make improvements. A collection of 26 parts was modeled and converted to STEP models to validate the AFR method and verify the accuracy of the AFR prototype.

The results from the AFR prototype show perfect accuracy in recognizing all the features of all 26 parts and confirm there is great potential for further development of AFR algorithms in rather specialized domains of application. Although feature recognition from B-rep models has been investigated for decades, it has not become an integral part of CAD solutions. The authors of this paper believe that works on AFR have been too general to be feasible or accurate enough for commercial solutions. A rather specialized approach like our originally proposed method whose implementation is represented in this paper makes it possible to break down the AFR problem and propose accurate specialized solutions.

Seyedmorteza Ghaffarishahri, http://orcid.org/0000-0001-9894-6539
Louis Rivest, http://orcid.org/0000-0003-0112-0090

REFERENCES

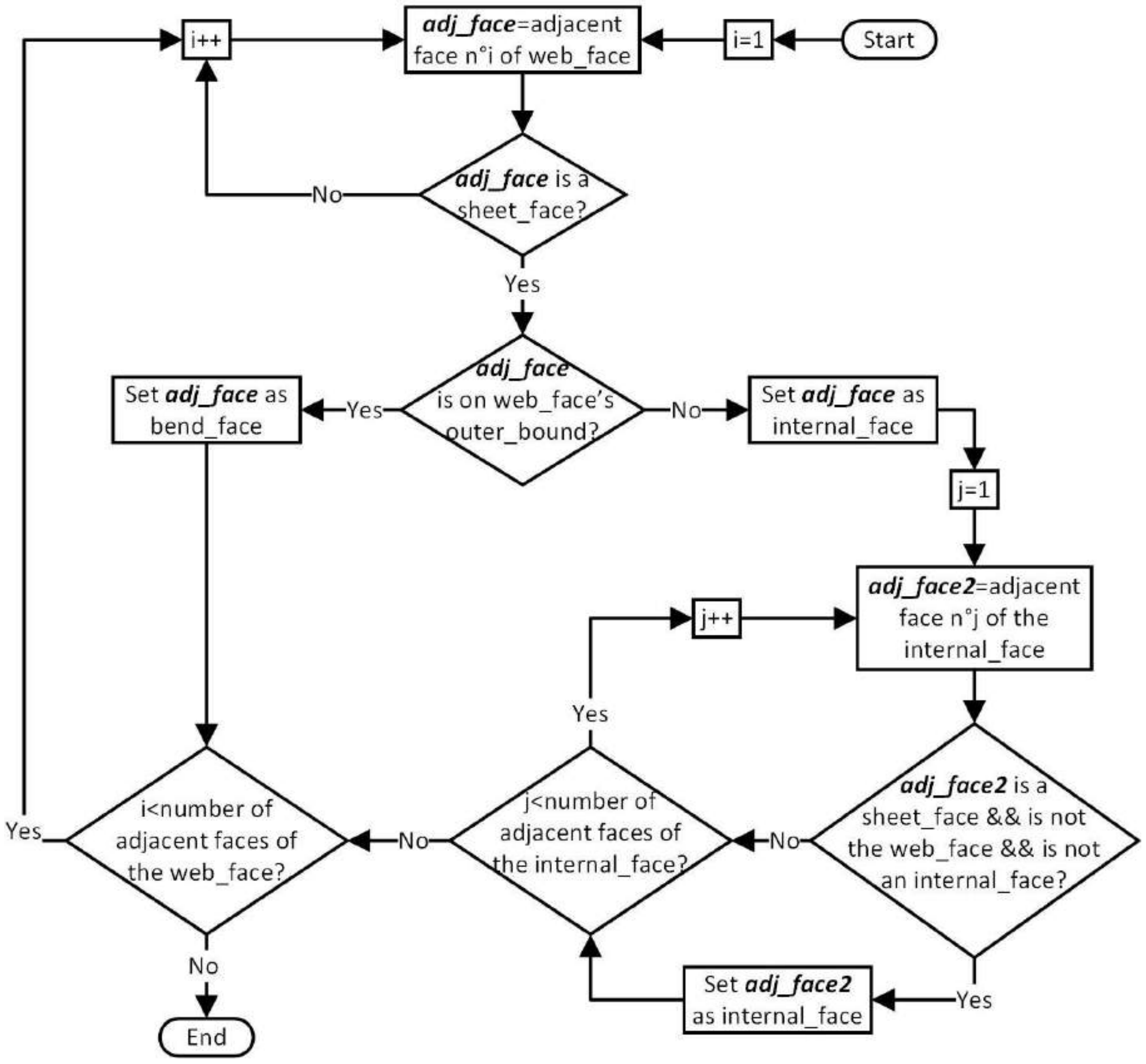
[1] CATIA 3DExperience 2017X, Dassault Systems.
[2] CATIA V5-R2016, Dassault Systems.
[3] NX for Design, Siemens PLM Software.
[4] Pratt, M. J.: Introduction to ISO 10303—the STEP standard for product data exchange, Journal of Computing and Information Science in Engineering, 1(1), 2001, 102-103.
[5] Ghaffarishahri, S.; Rivest, L.: Feature-Based Model Difference Identification for Aerospace Sheet Metal Parts, Computer-Aided Design & Applications, 18(3), 2021, 443-467. https://doi.org/10.14733/cadaps.2021.443-467.
[6] Babić, B.; Nešić, N.; Miljković, Z.: A review of automated feature recognition with rule-based pattern recognition, Computers in industry, 59(4), 2008, 321-337. https://doi.org/10.1016/j.compind.2007.09.001.
[7] Wang, J.; Zhou, L.: Algorithm for automatic boss feature recognition and ejector sleeve design, The International Journal of Advanced Manufacturing Technology, 97(1-4), 2018, 583-597. https://doi.org/10.1007/s00170-018-1918-9.
[8] Zubair, A. F.; Mansor, M. S. A.: Automatic feature recognition of regular features for symmetrical and non-symmetrical cylinder part using volume decomposition method, Engineering with Computers, 34(4), 2018, 843-863. https://doi.org/10.1007/s00366-018-0576-8.
[9] Lai, J.-Y.; Wang, M.-H.; Song, P.-P.; Hsu, C.-H.; Tsai, Y.-C.: Automatic recognition and decomposition of rib features in thin-shell parts for mold flow analysis, Engineering with Computers, 34(4), 2018, 801-820. https://doi.org/10.1007/s00366-017-0574-2.

[10] Li, Y. G.; Ding, Y. F.; Mou, W. P.; Guo, H.: Feature recognition technology for aircraft structural parts based on a holistic attribute adjacency graph, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 224(2), 2009, 271-278. https://doi.org/10.1243/09544054JEM1634.
[11] Langerak, T. R.: Local parameterization of freeform shapes using freeform feature recognition, Computer-Aided Design, 42(8), 2010, 682-692. https://doi.org/10.1016/j.cad.2010.02.004.
[12] Gupta, R. K.; Gurumoorthy, B.: Automatic extraction of free-form surface features (FFSFs), Computer-Aided Design, 44(2), 2012, 99-112. https://doi.org/10.1016/j.cad.2011.09.012.
[13] Sunil, V.; Agarwal, R.; Pande, S.: An approach to recognize interacting features from B-Rep CAD models of prismatic machined parts using a hybrid (graph and rule based) technique, Computers in Industry, 61(7), 2010, 686-701. https://doi.org/10.1016/j.compind.2010.03.011.
[14] Kim, B. C.; Mun, D.: Enhanced volume decomposition minimizing overlapping volumes for the recognition of design features, Journal of Mechanical Science and Technology, 29(12), 2015, 5289-5298. https://doi.org/10.1007/s12206-015-1131-9.
[15] Šormaz, D. N.; Tennety, C.: Recognition of interacting volumetric features using 2D hints, Assembly Automation, 2010, https://doi.org/10.1108/01445151011029763.
[16] Venu, B.; Komma, V. R.; Srivastava, D.: STEP-based feature recognition system for B-spline surface features, International Journal of Automation and Computing, 15(4), 2018, 500-512. https://doi.org/10.1007/s11633-018-1116-0.
[17] Wang, Q.; Yu, X.: Ontology based automatic feature recognition framework, Computers in Industry, 65(7), 2014, 1041-1052. https://doi.org/10.1016/j.compind.2014.04.004.
[18] Nnaji, B. O.; Kang, T.-S.; Yeh, S.; Chen, J.-P.: Feature reasoning for sheet metal components, International Journal of Production Research, 29(9), 1991, 1867-1896. https://doi.org/10.1080/00207549108948055.
[19] Jagirdar, R.; Jain, V. K.; Batra, J. L.; Dhande, S. G.: Feature recognition methodology for shearing operations for sheet metal components, Computer Integrated Manufacturing Systems, 8(1), 1995, 51-62. https://doi.org/10.1016/0951-5240(95)92813-A.
[20] Devarajan, M.; Kamran, M.; Nnaji, B. O.: Profile offsetting for feature extraction and feature tool mapping in sheet metal, International Journal of Production Research, 35(6), 1997, 1593-1608. https://doi.org/10.1080/002075497195146.
[21] Kannan, T. R.; Shunmugam, M. S.: Processing of 3D sheet metal components in STEP AP-203 format. Part II: feature reasoning system, International Journal of Production Research, 47(5), 2009, 1287-1308. https://doi.org/10.1080/00207540701510063.
[22] Liu, Z.; Li, J.; Wang, Y.; Li, C.; Xiao, X.: Automatically extracting sheet-metal features from solid model, Journal of Zhejiang University-Science A, 5(11), 2004, 1456-1465. https://doi.org/10.1631/jzus.2004.14.
[23] Kannan, T. R.; Shunmugam, M. S.: Processing of 3D sheet metal components in STEP AP-203 format. Part I: feature recognition system, International Journal of Production Research, 47(4), 2009, 941-964. https://doi.org/10.1080/00207540701510055.
[24] Jagirdar, R.; Jain, V. K.; Batra, J. L.: Characterization and identification of forming features for 3-D sheet metal components, International Journal of Machine Tools and Manufacture, 41(9), 2001, 1295-1322. https://doi.org/10.1016/S0890-6955(01)00006-2.
[25] Gupta, R. K.; Gurumoorthy, B.: Classification, representation, and automatic extraction of deformation features in sheet metal parts, Computer-Aided Design, 45(11), 2013, 1469-1484. https://doi.org/10.1016/j.cad.2013.06.010.
[26] Sunil, V.; Pande, S.: Automatic recognition of features from freeform surface CAD models, Computer-Aided Design, 40(4), 2008, 502-517. https://doi.org/10.1016/j.cad.2008.01.006.
[27] Zhang, C.-j.; Zhou, X.-h.; Li, C.-x.: Automatic recognition of intersecting features of freeform sheet metal parts, Journal of Zhejiang University-SCIENCE A, 10(10), 2009, 1439-1449. https://doi.org/10.1631/jzus.A0820705.
[28] FeatureWorks, Dassault Systems.

[29] Sheet Metal Feature Recognition Library, HCL Technologies.
[30] Ghaffarishahri, S.; Rivest, L.: Feature Recognition for Structural Aerospace Sheet Metal Parts, Computer-Aided Design & Applications, 17(1), 2020, 16-43. https://doi.org/10.14733/cadaps.2020.16-43.
[31] Niu, C.: Airframe structural design: practical design information and data on aircraft structures, Conmilit Press Limited, Hong Kong, 1999.

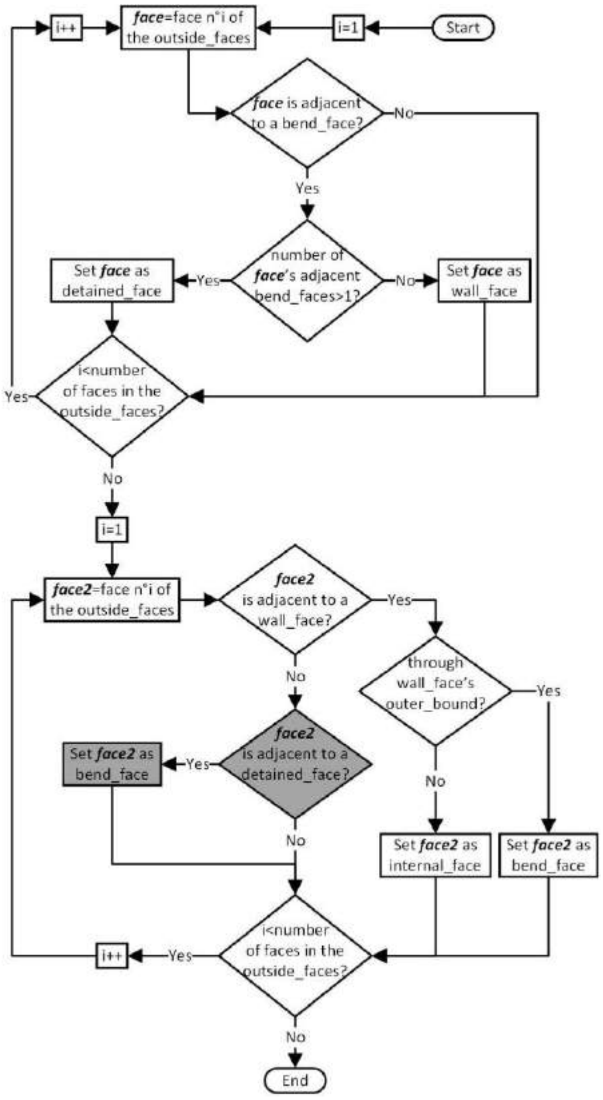
APPENDIX A

The algorithm for classifying sheet_faces adjacent to web_face starts with checking if each of the faces adjacent to the web_face is a sheet_face. If the sheet_face is adjacent to the web_face through its outer_bound, it is classified as a bend_face, otherwise it is classified as an internal_face and all its adjacent faces that are sheet_faces are also classified as internal_faces.


Start
i=1
adj_face=adjacent face n°i of web_face
adj_face is a sheet_face?
No
i++
Yes
adj_face is on web_face's outer_bound?
Yes
Set adj_face as bend_face
No
Set adj_face as internal_face
j=1
adj_face2=adjacent face n°j of the internal_face
adj_face2 is a sheet_face && is not the web_face && is not an internal_face?
Yes
Set adj_face2 as internal_face
No
j<number of adjacent faces of the internal_face?
Yes
j++
No
i<number of adjacent faces of the web_face?
Yes
No
End

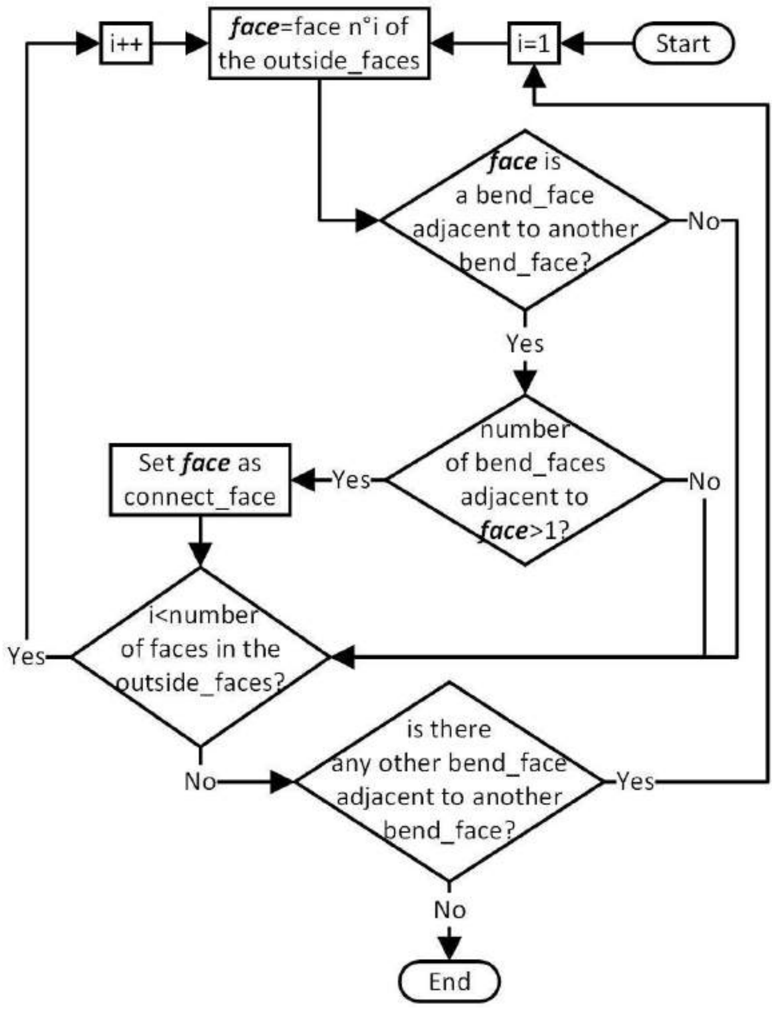
APPENDIX B

The algorithm for classifying the remaining sheet_faces as wall_faces, bend_faces, internal_faces or detained_faces starts with the outside_faces list. Some of these sheet_faces are already classified by their subtypes and will be used to classify their adjacent sheet_faces. If a face is adjacent to only one bend_face, it is classified as a wall_face, and if it is adjacent to more than one bend_face, it is classified as a detained_face. Once all the faces in the outside_faces list have been checked to classify those that are adjacent to a bend_face, the second round of the algorithm starts. In the second round, each face in the outside_faces list is checked to determine whether it is adjacent to a wall_face. If a face is not adjacent to a wall_face but is adjacent to a detained_face, it is classified as a bend_face. This condition and its associated step are missing from the originally proposed method and highlighted in the flowchart below. If a face is adjacent to a wall_face through its outer_bound, it is classified as a bend_face, otherwise it is classified as an internal_face.


Start
i=1
face=face n°i of the outside_faces
i++
face is adjacent to a bend_face?
No
Yes
number of face's adjacent bend_faces>1?
Yes
No
Set face as detained_face
Set face as wall_face
i<number of faces in the outside_faces?
Yes
No
i=1
face2=face n°i of the outside_faces
face2 is adjacent to a wall_face?
Yes
No
through wall_face's outer_bound?
Yes
No
face2 is adjacent to a detained_face?
Yes
No
Set face2 as bend_face
Set face2 as internal_face
Set face2 as bend_face
i<number of faces in the outside_faces?
Yes
i++
No
End

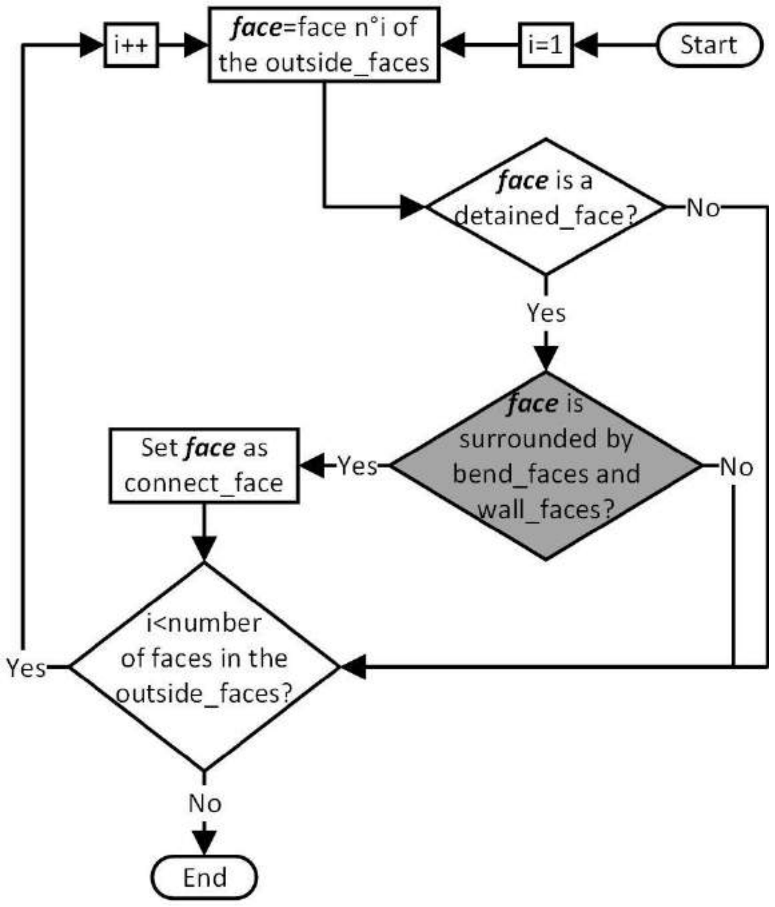
APPENDIX C

The algorithm for changing eligible bend_faces to connect_faces starts with the outside_faces list. If a face is a bend_face and is also adjacent to more than one other bend_faces, it is re-classified as a connect_face. The outside_faces list is rechecked until no new connect_face is identified.


Start
i=1
face=face n°i of the outside_faces
i++
face is a bend_face adjacent to another bend_face?
No
Yes
number of bend_faces adjacent to face>1?
Yes
No
Set face as connect_face
i<number of faces in the outside_faces?
Yes
No
is there any other bend_face adjacent to another bend_face?
Yes
No
End

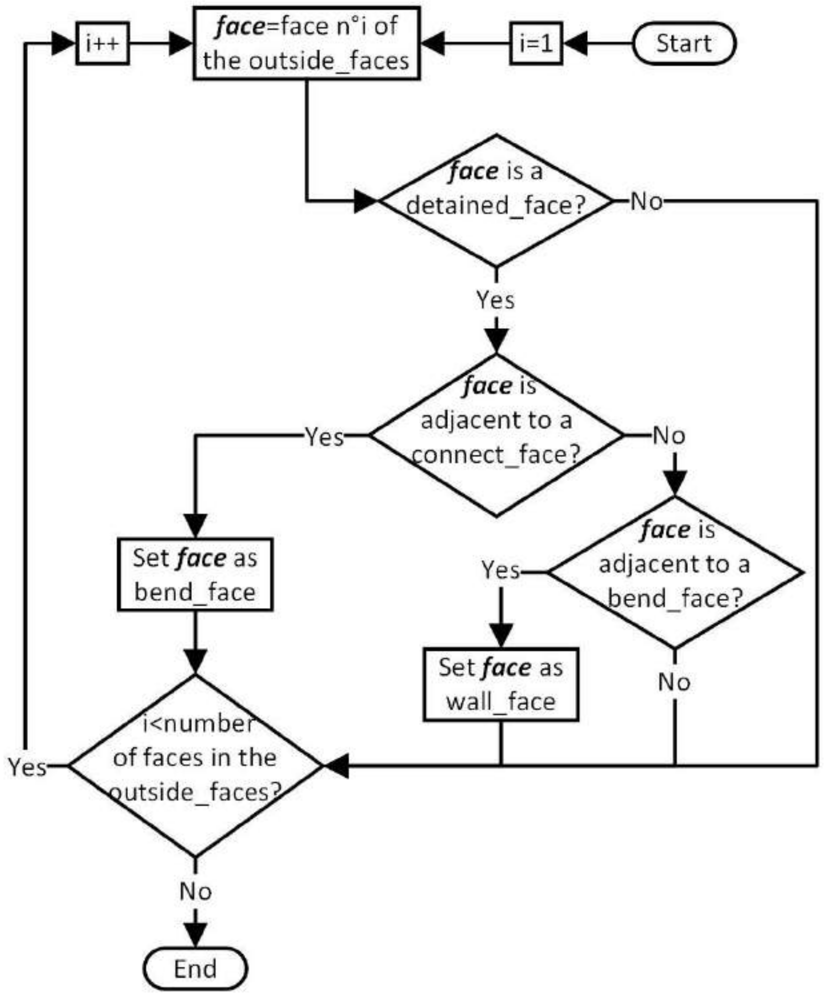
APPENDIX D

The algorithm for changing eligible detained_faces to connect_faces starts with the outside_faces list. If a face is a detained_face and is surrounded by bend_faces and wall_faces so it does not have any adjacent trim_face on its outer_bound, it is re-classified as a connect_face. The condition of being surrounded by bend_faces and wall_faces is missing in the original method and highlighted in the flowchart below.


Start
i=1
face=face n°i of the outside_faces
i++
face is a detained_face?
No
Yes
face is surrounded by bend_faces and wall_faces?
Yes
No
Set *face* as connect_face
i<number of faces in the outside_faces?
Yes
No
End

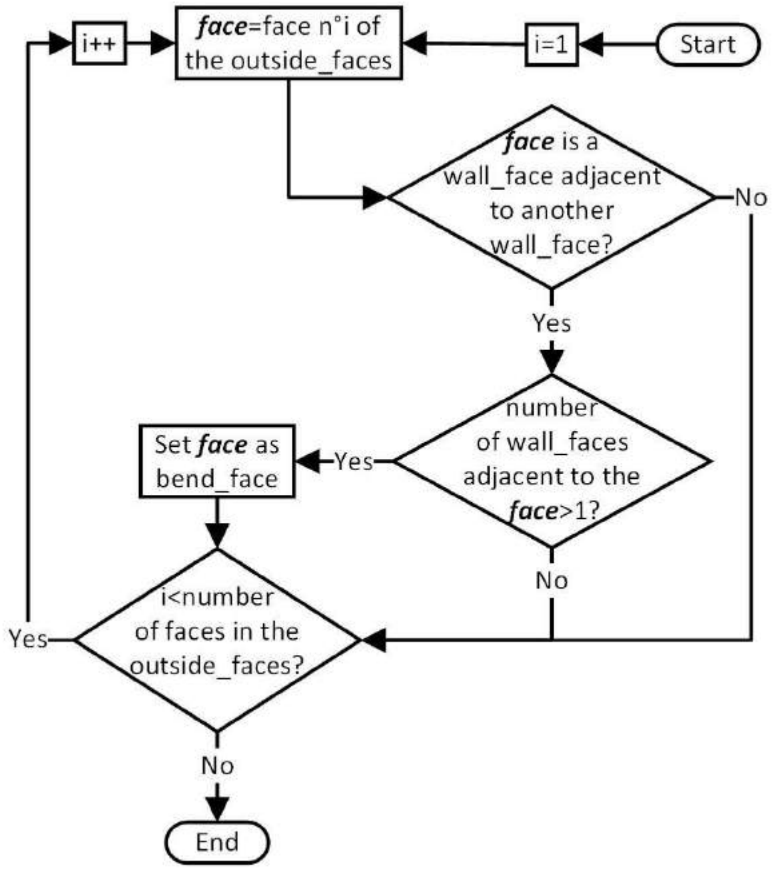
APPENDIX E

The algorithm for changing eligible detained_faces to bend_faces or wall_faces starts with the outside_faces list. If a face is a detained_face and is adjacent to a connect_face, it is re-classified as a bend_face. If a face is a detained_face and is not adjacent to a connect_face but is adjacent to a bend_face, it is re-classified as a wall_face.


Start
i=1
face=face n°i of the outside_faces
i++
face is a detained_face?
No
Yes
face is adjacent to a connect_face?
Yes
No
Set face as bend_face
face is adjacent to a bend_face?
Yes
No
Set face as wall_face
i<number of faces in the outside_faces?
Yes
No
End

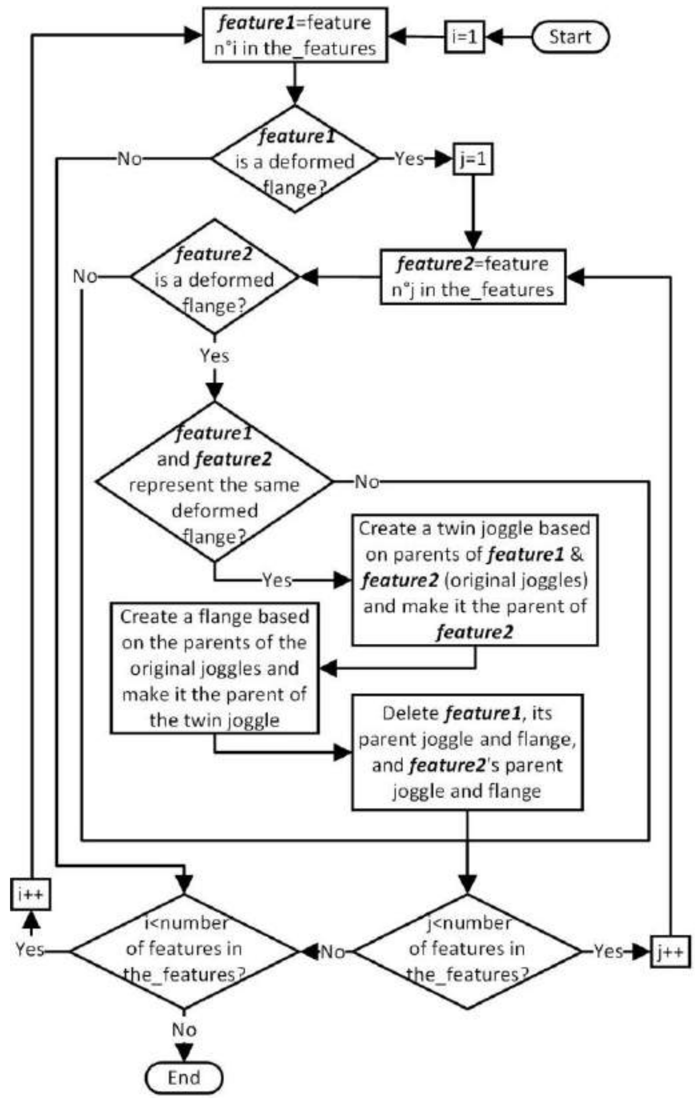
APPENDIX F

The algorithm for changing eligible wall_faces to bend_faces starts with the outside_faces list. If a face is a wall_face adjacent to more than one other wall_faces, it is classified as a bend_face.


Start
i=1
face=face n°i of the outside_faces
i++
face is a wall_face adjacent to another wall_face?
No
Yes
number of wall_faces adjacent to the *face*>1?
Yes
Set *face* as bend_face
No
i<number of faces in the outside_faces?
Yes
No
End

APPENDIX G

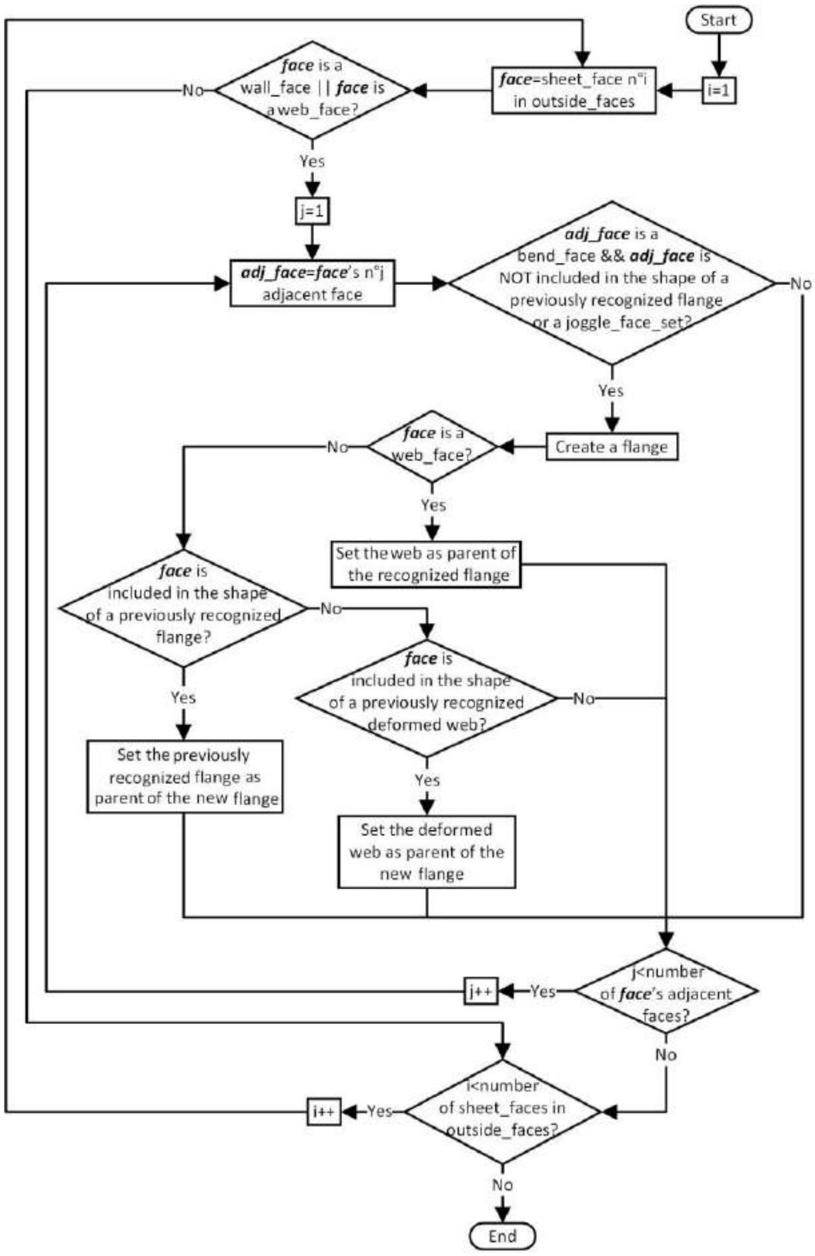
The algorithm for recognizing twin joggles starts with the the_features list. If a feature is a deformed flange, the other deformed flanges in the list are checked to determine whether they represent the same deformed flange. They are checked by verifying whether they are linked to the same wall_face as the original deformed flange. If they are, a twin joggle is created based on the joggles that were the parents of these deformed flanges, and the twin joggle is made the parent of one of the deformed flanges (in the flowchart below, it is referred to as feature2). Also, a flange is created based on the flanges that were the original parents of the joggles that were combined to create the twin joggle. This new flange is made the parent of the twin joggle. At the end, the deformed flange that was NOT made the child of the twin joggle, the original joggles (NOT the newly created twin joggle) and the original flanges (NOT the new flange created based on the original flanges) that were the parents of the original joggles are all deleted from the the_features list.



Start
i=1
feature1=feature n°i in the_features
feature1 is a deformed flange?
No
Yes
j=1
feature2=feature n°j in the_features
feature2 is a deformed flange?
No
Yes
feature1 and feature2 represent the same deformed flange?
No
Yes
Create a twin joggle based on parents of feature1 & feature2 (original joggles) and make it the parent of feature2
Create a flange based on the parents of the original joggles and make it the parent of the twin joggle
Delete feature1, its parent joggle and flange, and feature2's parent joggle and flange
j<number of features in the_features?
Yes
j++
No
i<number of features in the_features?
Yes
i++
No
End

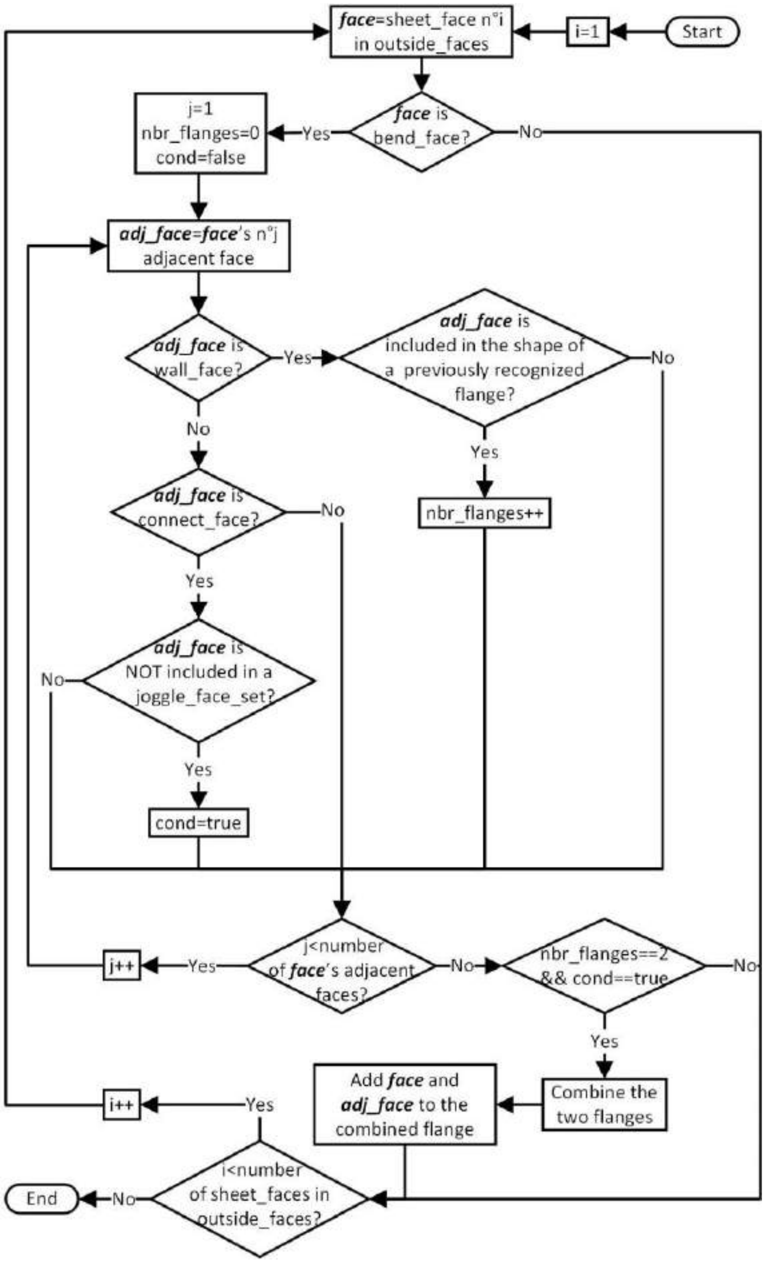
APPENDIX H

The algorithm for recognizing the remaining flanges starts with the outside_faces list. If a face is a web_face or wall_face that has an adjacent bend_face that is not included in a previously recognized flange, deformed flange or joggle_face_set, a new flange is created. If the face is a web_face, make the web the parent of the newly created flange. If it is not a web_face, therefore a wall_face, and it is included in a previously recognized flange, the previously recognized flange is made the parent of the newly created flange. If the face is not a web_face and is included in a previously recognized deformed web, the previously recognized deformed web is made the parent of the newly created flange.


Start
i=1
face=sheet_face n°i in outside_faces
face is a wall_face || face is a web_face?
No
Yes
j=1
adj_face=face's n°j adjacent face
adj_face is a bend_face && adj_face is NOT included in the shape of a previously recognized flange or a joggle_face_set?
No
Yes
Create a flange
face is a web_face?
No
Yes
Set the web as parent of the recognized flange
face is included in the shape of a previously recognized flange?
No
Yes
Set the previously recognized flange as parent of the new flange
face is included in the shape of a previously recognized deformed web?
No
Yes
Set the deformed web as parent of the new flange
j<number of face's adjacent faces?
j++
Yes
No
i<number of sheet_faces in outside_faces?
i++
Yes
No
End

APPENDIX I

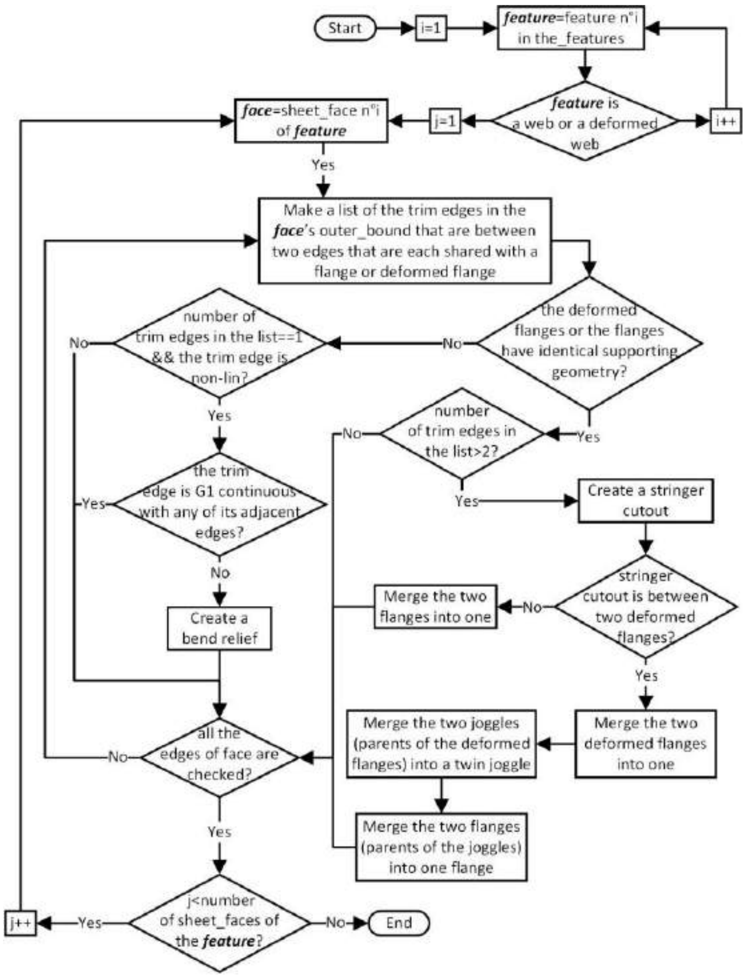
The algorithm for recognizing combined flanges starts with the outside_faces list. Each of the bend_faces in the list is checked to determine whether any of its adjacent faces is a wall_face that is included in a previously recognized flange. Such adjacent faces are counted. Also, the face's adjacent faces that are not wall_faces are checked to determine whether they are connect_faces that are not included in a joggle_face_set. If such an adjacent connect_face is found and the number of adjacent wall_faces included in previously recognized flanges is 2, a new flange is created by combining the previously recognized flanges, and the face and its adjacent connect_face are added to the newly created flange.

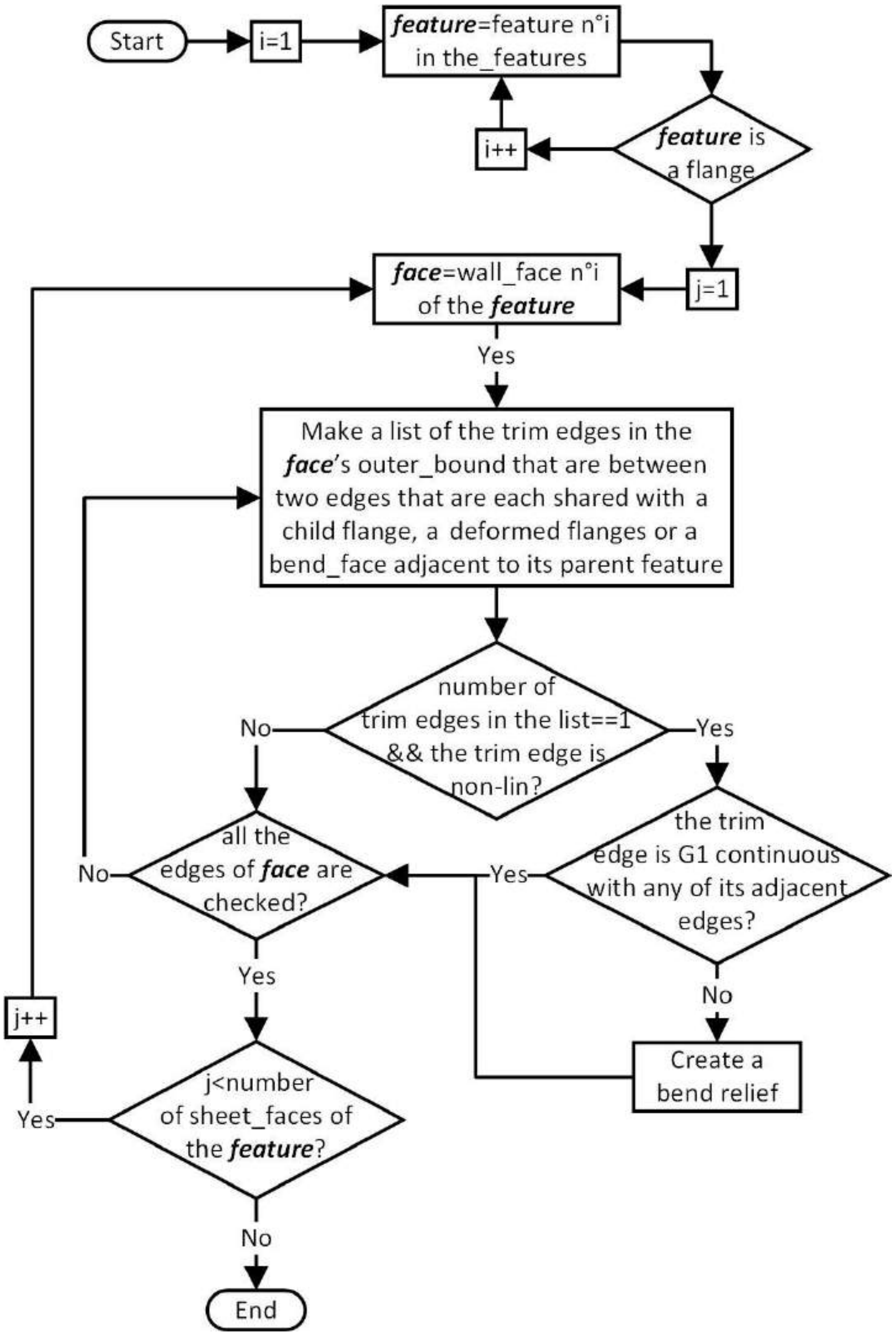

Start
i=1
face=sheet_face n°i in outside_faces
face is bend_face?
Yes
No
j=1
nbr_flanges=0
cond=false
adj_face=face's n°j adjacent face
adj_face is wall_face?
Yes
No
adj_face is included in the shape of a previously recognized flange?
Yes
No
nbr_flanges++
adj_face is connect_face?
Yes
No
adj_face is NOT included in a joggle_face_set?
Yes
No
cond=true
j<number of face's adjacent faces?
Yes
No
j++
nbr_flanges==2 && cond==true
Yes
No
Combine the two flanges
Add face and adj_face to the combined flange
i<number of sheet_faces in outside_faces?
Yes
No
i++
End

APPENDIX J

The algorithm for recognizing stringer cutouts and bend reliefs can be divided into two parts. The first part starts with the outside_faces list and a web or deformed web. Each sheet_face of these features that is a web_face or wall_face is checked to make a list of trim edges in its outer_bound that are between two edges that are each shared with a flange or deformed flange. If the flanges or deformed flanges have identical supporting geometries and the number of edges in the list is more than 2, a stringer cutout is created based on the list of edges. If the stringer cutout is between two flanges, they are merged into one, and if the stringer cutout is between two deformed flanges, they are merged into one, their parent joggles are merged into one twin joggle, and the joggles' parent flanges are merged into one flange. If the flanges or deformed flanges do not have identical supporting geometries, there is only one edge in the list, and the edge is a non-lin edge that is not G1 continuous with its adjacent edges, a bend relief is created based on the trim edge.

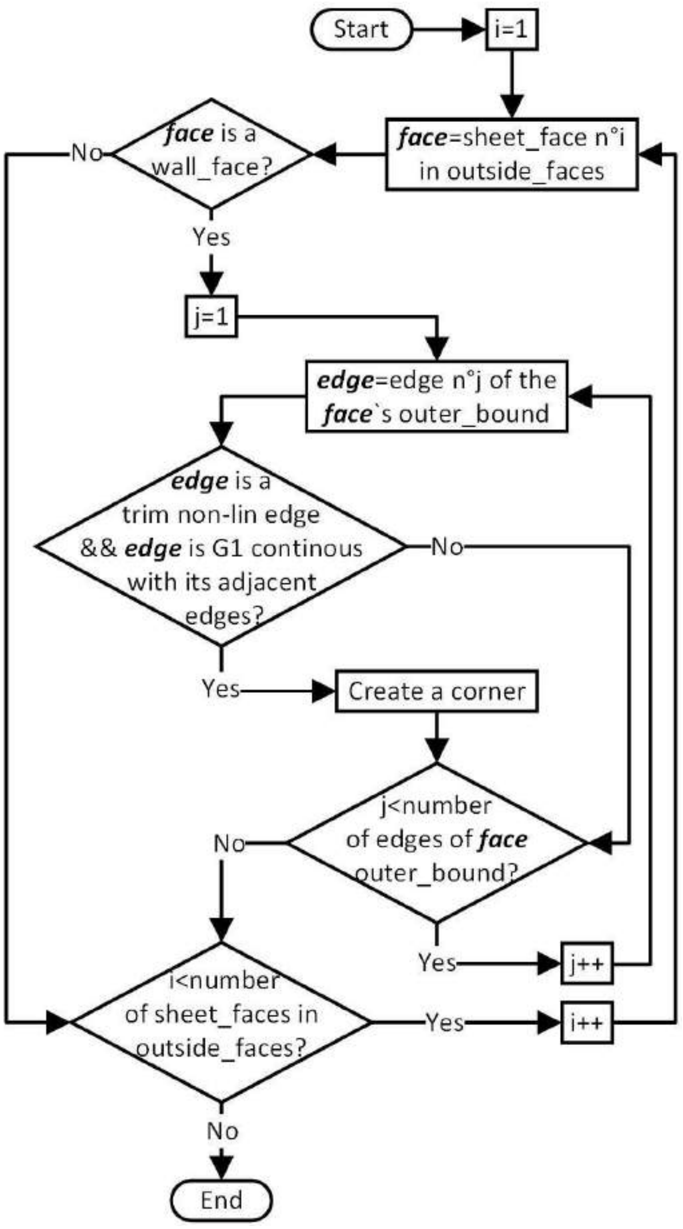
The second part starts with the the_features list and a flange. Each wall_face of the flange is checked to make a list of trim edges in its outer_bound that are between two edges that are each shared with a child flange, a deformed flange or a bend_face that is adjacent to its parent feature. If the child flange, the deformed flange or the parent feature do not have identical supporting geometries, there is only one edge in the list, and the edge is a non-lin edge that is not G1 continuous with its adjacent edges, a bend relief is created based on the trim edge.


Start
i=1
feature=feature n°i in the_features
feature is a web or a deformed web
i++
j=1
face=sheet_face n°i of *feature*
Yes
Make a list of the trim edges in the *face*'s outer_bound that are between two edges that are each shared with a flange or deformed flange
the deformed flanges or the flanges have identical supporting geometry?
No
number of trim edges in the list==1 && the trim edge is non-lin?
No
Yes
the trim edge is G1 continuous with any of its adjacent edges?
Yes
No
Create a bend relief
number of trim edges in the list>2?
Yes
No
Yes
Create a stringer cutout
stringer cutout is between two deformed flanges?
No
Merge the two flanges into one
Yes
Merge the two deformed flanges into one
Merge the two joggles (parents of the deformed flanges) into a twin joggle
Merge the two flanges (parents of the joggles) into one flange
all the edges of face are checked?
No
Yes
j<number of sheet_faces of the *feature*?
Yes
j++
No
End


Start
i=1
feature=feature n°i in the_features
feature is a flange
i++
j=1
face=wall_face n°i of the feature
Yes
Make a list of the trim edges in the face's outer_bound that are between two edges that are each shared with a child flange, a deformed flanges or a bend_face adjacent to its parent feature
number of trim edges in the list==1 && the trim edge is non-lin?
No
Yes
the trim edge is G1 continuous with any of its adjacent edges?
Yes
No
Create a bend relief
all the edges of face are checked?
No
Yes
j<number of sheet_faces of the feature?
Yes
j++
No
End

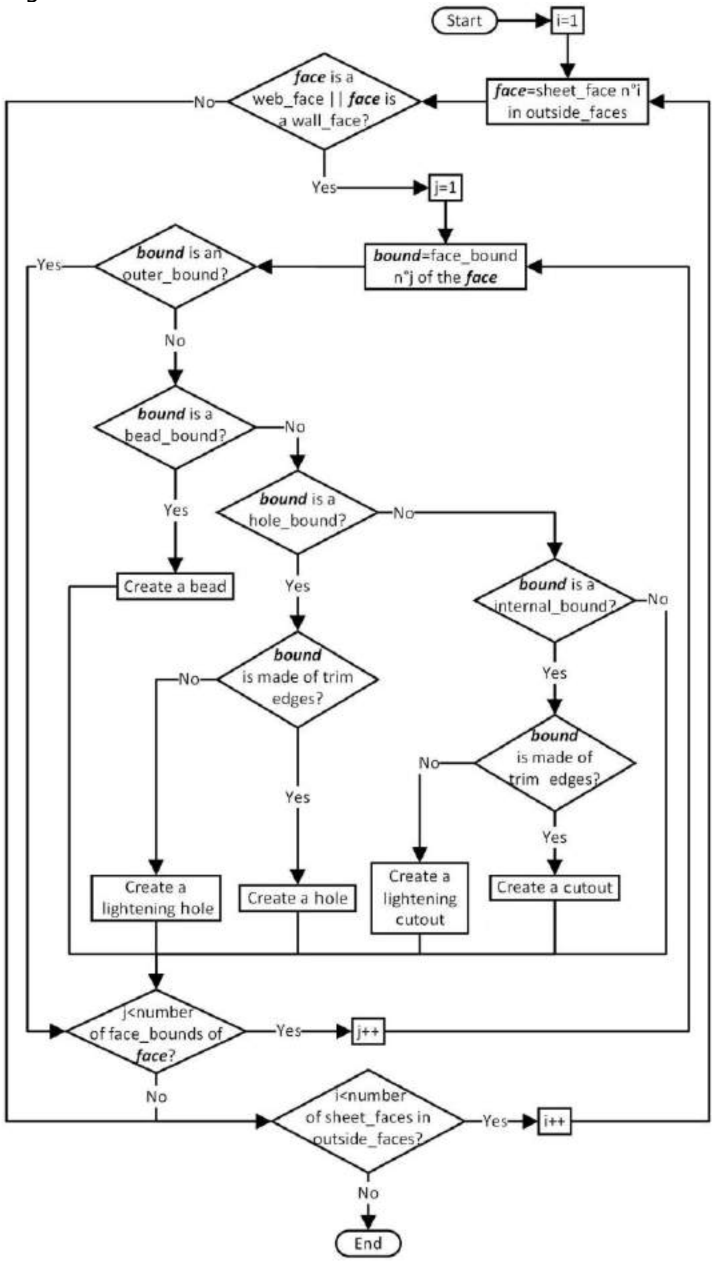
APPENDIX K

The algorithm for recognizing corners starts with the outside_faces list. The edges in the outer_bound of each wall_face in the list are checked to determine whether they are trim non-lin edges and G1 continuous with their adjacent edges. If an edge is both a trim non-lin edge and G1 continuous, a corner is created based on the edge.


Start
i=1
face=sheet_face n°i in outside_faces
face is a wall_face?
No
Yes
j=1
edge=edge n°j of the face`s outer_bound
edge is a trim non-lin edge && edge is G1 continous with its adjacent edges?
No
Yes
Create a corner
j<number of edges of face outer_bound?
No
Yes
j++
i<number of sheet_faces in outside_faces?
Yes
i++
No
End

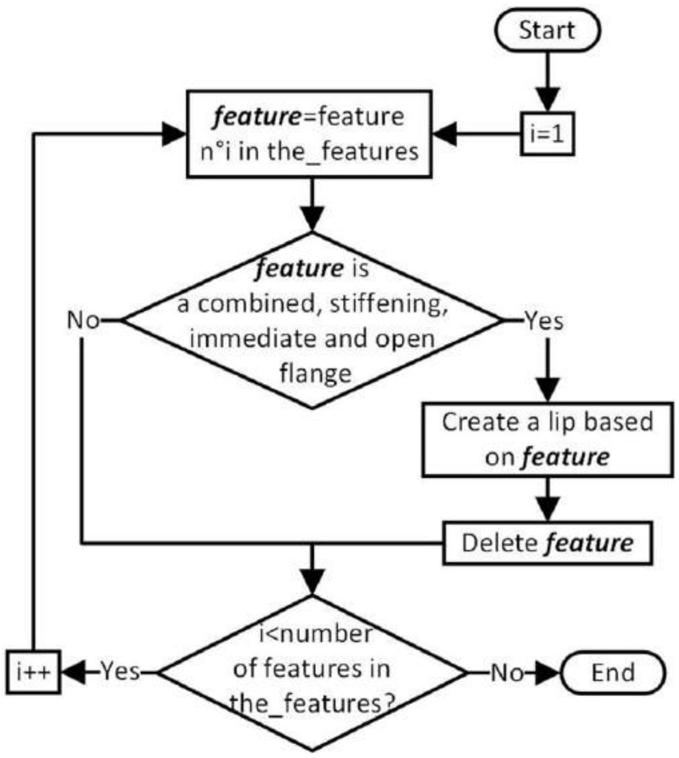
APPENDIX L

The algorithm for recognizing holes, cutouts, lightening holes, lightening cutouts and beads starts with the outside_faces list. Each bound of each web_face and wall_face in the list is checked to determine whether it is an outer_bound. If a bound is a bead_bound, a bead is created based on it. If a bound is a hole_bound and is made of trim edges, a hole is created based on it, or if it is instead made of non-trim edges, a lightening hole is created based on it. If a bound is an internal_bound and is made of trim edges, a cutout is created based on it, or if it is made of non-trim edges, a lightening cutout is created based on it.


Start
i=1
face=sheet_face n°i in outside_faces
face is a web_face || face is a wall_face?
No
Yes
j=1
bound=face_bound n°j of the face
bound is an outer_bound?
Yes
No
bound is a bead_bound?
No
Yes
Create a bead
bound is a hole_bound?
No
Yes
bound is a internal_bound?
No
Yes
bound is made of trim edges?
No
Yes
bound is made of trim edges?
No
Yes
Create a lightening hole
Create a hole
Create a lightening cutout
Create a cutout
j<number of face_bounds of face?
Yes
j++
No
i<number of sheet_faces in outside_faces?
Yes
i++
No
End

APPENDIX M

The algorithm for recognizing lips starts with the the_features list. Each feature that is a combined-open-immediate-stiffening flange is used to create a lip based on it and then deleted.


Start
i=1
feature=feature n°i in the_features
feature is a combined, stiffening, immediate and open flange
No
Yes
Create a lip based on *feature*
Delete *feature*
i<number of features in the_features?
Yes
i++
No
End