Int J Adv Manuf Technol (2005) 27: 305–312
DOI 10.1007/s00170-004-2193-5

ORIGINAL ARTICLE

Yanwei Wang · Junjun Wu · Liping Chen · Zhengdong Huang

Identity propagation method for tracing alterations of a topological entity in a history-based solid modeling system

Received: 14 January 2004 / Accepted: 01 April 2004 / Published online: 2 February 2005

Abstract In a history-based parametric modeling system, the mechanism for naming topological entities is a key component for generating design variants by re-evaluation. During the process of re-evaluation, topological entities may be split, merged, or obliterated. Thus, it is a principal functional request for a naming mechanism to trace such alterations of topological entities. A data structure called a name propagation graph (NPG) is introduced to represent the identity propagation of topological entities. Rules and algorithms are also presented to identify the genetic entities, which originate from the entities on the original version of a part model. Our approach has been implemented in a history-based and feature-based modeling system.

Keywords Computer-aided design · Constraint-based design · Feature-based modeling · History-based modeling · Topological entity

1 Introduction

1.1 Topological entity naming mechanism

Feature-based design paradigms are by now well established in current solid modeling systems. Nearly all of the commercial CAD systems have adopted this paradigm. In addition, there are some research systems that adopt this design paradigm to some degree [1, 2].

Nearly all of the systems that adopt a feature-based design paradigm are history-based. In these systems, the design process is often sequential and history-based. Models are created via a sequence of attachments that apply design features to an incremental version of a part model. The attachment sequence of features is often referred to as a *design history*, which can be represented by a binary tree [3]. In a history-based and feature-based modeling system, a part model is usually defined through dimensions, constraints and features, which make it easy to generate a new version of the part model.

When designers want to get a new version of the designed part, they can edit the designed model using editing tools and the design history is re-evaluated automatically; thus, a new version is obtained. However, sometimes designers may not be able to get the satisfactory results this way [4] due to the following typical reasons:

- the ability of the geometric constraint solver is not strong enough to handle some odd cases; and
- some related entities cannot be consistently retrieved or identified from the information in the part model.

The handling of the first issue needs a robust geometric constraint solver. In fact, to some extent, the solver is a mature technology to solve geometric constraint system in 2D space. There are even some commercial software component modules to solve geometric constraint system.

However, the second issue is relevant to referencing a topological entity (TE). The history of designing a part model is a sequential one in a parametric feature-based system. Usually, a base feature is designed first, and then the other features are attached to it. When a new feature is added to the part model, perhaps these TEs, such as faces, edges, and vertices, may be referenced. How the TEs are referenced is listed as follows:

- as an operation object of a feature, i.e., the removed face of a shelling feature;
- as a datum object of a feature, i.e., the datum plane of the sketch of a protrusion feature; and
- as an intermediate between a feature and its semantic, i.e., the entities in a feature referenced by dimensions.

For the existence of the reference relation in history-based and feature-based modeling systems, we need a mechanism for consistently naming the TEs. Thus, the correct new version of the part model can be achieved by re-evaluation of design history. Such a mechanism is called a topological entity naming mechanism (TENM). TENM is a kernel module for a history-based and feature-based modeling system. A generic and robust TENM

Y. Wang (✉) · J. Wu · L. Chen · Z. Huang
CAD Center,
Huazhong University of Science and Technology,
Wuhan, P.R. China 430074
E-mail: wangyw@hustcad.com

is a key for capturing design intent. The principal functional requests for a TENM include two main aspects:

- from the static viewpoint, a TENM should be capable of naming the TEs in a part model uniquely and consistently; and
- from the dynamic viewpoint, a TENM should be capable of tracing the alteration of TEs during the design process. Such alteration of TEs includes merging, splitting, and obliteration.

1.2 Identity propagation of TEs

Let's consider the first functional request for a TENM, which is from the static viewpoint, and have a look at how a TE such as face, edge, and vertex is named.

Firstly, an identity is given to each point in a 2D profile. Then, the identity of each edge in the 2D profile can be decided by the identities of the associated point set. For example, the identity of a line segment is decided by the identities of its starting point and end point; the identity of a circle is determined by the identity of its center. As is well known, a sweep feature is constructed by sweeping a profile along a specific path. Notice that element E is one of the edges in the profile, while P is the sweep path; hence, the identity of the face delineated by E and P can be decided by the identities of E and P. Wu et al. [4] also presented the mechanism to name the faces in some other types of features such as chamfer, round, shell and so on. With such a concisely described approach, each face in the original feature will have an identity, called its original name (ON).

After naming all of the faces, Wu et al. pointed out that the ONs of the faces in an original feature are unchanged during the Boolean operation. Therefore, the name of an edge can be decided by the identities of the two associated faces, and the name of a vertex is determined by the identities of the three associated faces. Consequently, each TE in a part model such as the face, edge and vertex will have its own identity. Due to a lack of space, we cannot give a detailed description of the mechanism adopted; refer to [4] for details.

Now let's take the second functional request into consideration. It is well-known that, in a history-based solid modeling system, after adding new features to the original version or re-editing the original version, a new version of a part model can be achieved by re-evaluating the design history.

In the process of re-evaluation, a Boolean operation is necessary. During such a process, perhaps the TEs will be split, merged, or even obliterated. Such alteration of a TE is called topological entity propagation (TEP). In the new version of the part model, if a topological entity TE is constructed by propagation of a topological entity TE' that is on the original version of the part model, there must be a relation between TE and TE'. The relation is called a genetic relation (GR), and TE is the genetic entity (GE) of TE'.

For the existence of GRs between the original version and a new version of the TEs, the identity of the original version should be propagated to the new version in the same manner. We name such a process, in which the identity of the original version is propagated to the new version of a TE, a topological name propagation (TNP).

There are four types of TNP [4]:

- Propagation of deleting a topological name When a topological entity is deleted during re-evaluation, the identity of the entity should be deleted too. This is called propagation of deleting a topological name.
- Propagation of keeping a topological name If a topological entity is still alive and not split or merged with other entities during re-evaluation, no change will happen to the identity of the entity. This is called propagation of keeping a topological name.
- Propagation of splitting a topological name If a topological entity is split into two or more entities during Re-evaluation, the topological name of the entity will be carried to all of the entities that originate from it. This is called propagation of splitting a topological name.
- Propagation of merging a topological name If two or more entities are merged into one entity, one of the identities of these entities will be used to name the new resulting entity. This is called propagation of merging a topological name.

The following is a simple example to illustrate the importance of TNP modeling.

In Fig. 1, a column is created as the base feature of a part model. Then, a pocket is added, and finally, a round is added to the top face of the column. Thus, the original version of the part is achieved and is shown in Fig. 1b. After the achievement of the original version, the pocket is re-edited and turned into a through-all slot. According to the design history tree shown in Fig. 1a, after reevaluation, we will obtain a new version of the part. During the re-evaluation, the top face f is split into f1 and f2, and consequently, the identity of f should be propagated to f1 and f2. If the propagation can't be traced by the modeling system, perhaps the correct resulting part shown in Fig. 1c can't be achieved. Some incorrect results are shown in Figs. 1d, 1e and 1f.

In reference [4], after pointing out the phenomenon of identity propagation, Wu et al. did not give any algorithms or details to trace the identity propagation. In this paper, we elaborate on a mechanism to fulfill the second functional request for a TENM. First, a data structure called a name propagation graph (NPG) is introduced to represent the identity propagation process of topological faces. Then, based on an NPG, some rules and algorithms are given to identify the genetic entity such as genetic face, genetic edge and genetic vertex.

This paper is organized as follows: in Sect. 2, after defining NPG, we give some properties of an NPG. In Sect. 3, an approach to identify a genetic face is described. Section 4 describes how to identify genetic edges. And in Sect. 5, a rule is presented to identify genetic vertices. The implementation is briefly introduced in Sect. 6. Section 7 summarizes the work.

1.3 Review of related research on TENM

A topological ID system is presented by Kripac [5] to map the ID of topological entities on the original version of a part model to

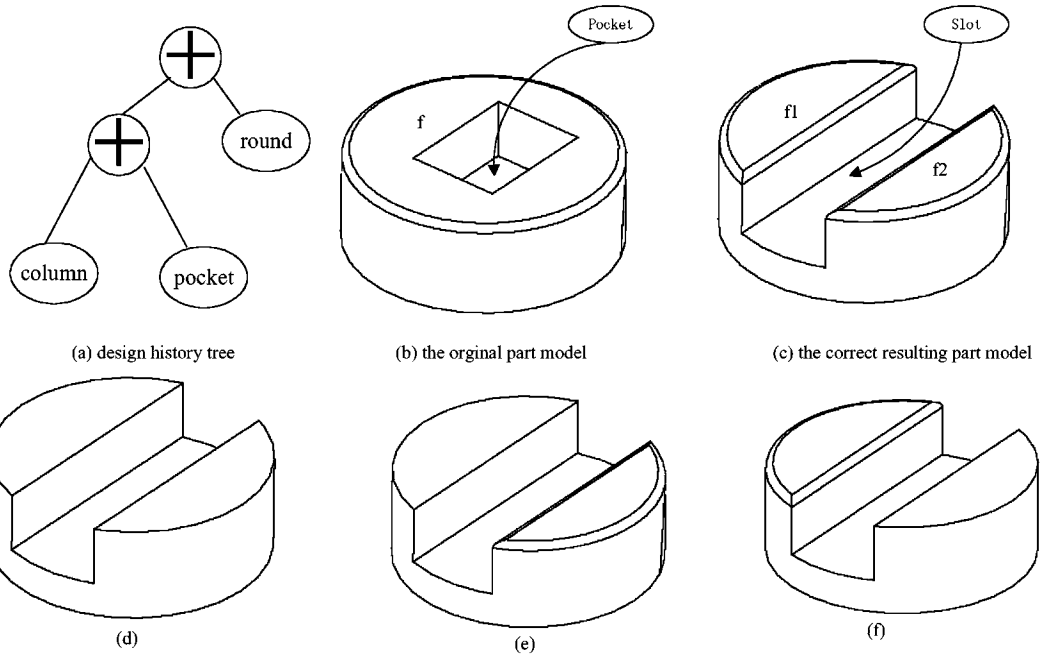


Fig. 1. A simple example of TNP modeling

the ID of entities on a new version. A graph-based approach is also presented to represent the alteration of entities such as merging, splitting, and so on. Some examples are given to illustrate the effectiveness of his approach. However, the details are omitted. Therefore, the algorithm adopted by Kripac is perplexing.

Capoyleas [6] presented a method to name the topological entities by giving each entity an unique identity. But Capoyleas didn't consider the alteration of entities during the re-evaluation of the part. Thus, only the functional request from the static viewpoint for a TENM is satisfied. However, the second functional request from a dynamic viewpoint can't be fulfilled. In order to improve the editability of feature-based design, Chen [7] presented matching algorithms for the vertex, edge and face, based on the method by Capoyleas. But it seems that the algorithms presented by Chen are not easy to be implemented.

Certainly, many commercial modeling systems have done much work on TENM because a robust TENM is a prerequisite for developing a feature-based modeling system. However, perhaps for proprietary reasons, they are unwilling to make their approach public.

2 Name propagation graph

A name propagation graph (NPG) is defined as $NPG = (E, R)$, where E is the node set and each node represents a complex code (CC) of a topological face. A CC is a triplet; that is, $\mathrm{CC} = \{\mathrm{ON}(f), seq, num\}$, where $\mathrm{ON}(f)$ represents the ON of a particular face (denoted f) and the process of generating an ON for a topological face has been described in Sect. 1.2; *seq* represents how many times f has been split or merged in the current part model; and *num* is used to identify the faces that are constructed by the splitting of f. For example, if f is split into two faces, f1 and f2, *num* corresponding to f1 is 1 and *num* corresponding to f2 is 2. The directed arc set of NPG, $R = \{ER\}$, and $ER = \left\{\langle CC_i, CC_j\rangle \,\middle|\, i \neq j, CC_i \neg CC_j\right\}$. The symbol $\neg$ indicates that the face corresponding to CC_j originates from the face corresponding to CC_i.

In order to describe the propagation of deleting a topological name, we introduce a special node called a rubbish node (RN) and the CC of the RN is (0, 0, 0). All of the nodes representing the deleted faces point to the RN.

The following is an example to illustrate how an NPG is constructed. The modeling process includes four steps: (a) create a block; (b) add a slot feature: Slot1; (c) add another feature: Pocket; (d) reevaluate the Pocket feature and turn it into a slot feature: Slot2. In order not to clutter the illustration, only the identity propagation process of the top face f1 is depicted.

In Fig. 2, assume that $\mathrm{ON(f1)} = 1$, then according to the definition of CC, $\mathrm{CC(f1)} = (1, 0, 1)$, $\mathrm{CC(f2)} = (1, 1, 1)$, $\mathrm{CC(f3)} = (1, 1, 2)$, $\mathrm{CC(f4)} = (1, 2, 1)$, $\mathrm{CC(f5)} = (1, 2, 2)$, $\mathrm{CC(f6)} = (1, 3, 1)$, $\mathrm{CC(f7)} = (1, 3, 2)$ and $\mathrm{CC(f8)} = (1, 3, 3)$. The texts on the arcs indicate the type of the topological name propagation.

The following are some basic properties of an NPG:

- Property 1: NPG is a directed acyclic graph (DAG). An NPG represents the process of splitting and merging of topological faces during the design process. In a history-based modeling system, the design process is unidirectional. Therefore, there must not be a cycle in an NPG, and the NPG is a DAG.
- Property 2: If a node whose CC= {0, 0, 0}, it must be an RN.
- Property 3: If a node contains no out-arc and it isn't an RN, the topological face corresponding to the node must exist on the current version of a part model. In Fig. 3, such a type of node includes $\mathrm{CC(f2)} = \{2.1.1\}$, $\mathrm{CC(f12)} = \{3.3.1\}$, $\mathrm{CC(f13)} = \{6.3.1\}$ and $\mathrm{CC(f14)} = \{6.3.2\}$. These faces, that

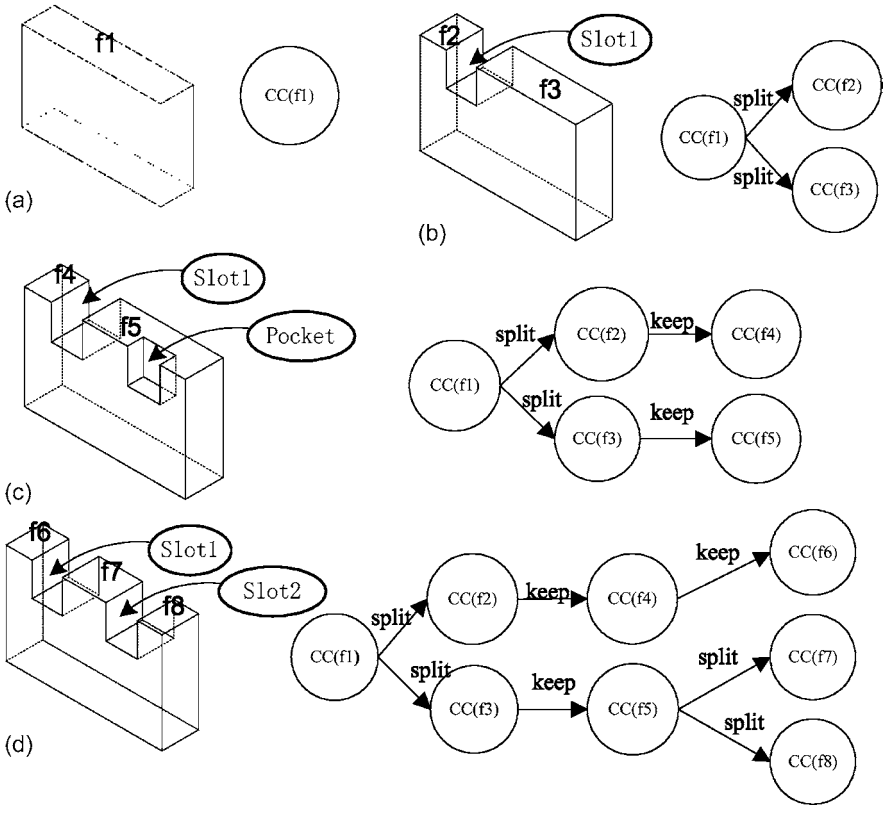


Fig. 2. The process of creating an NPG

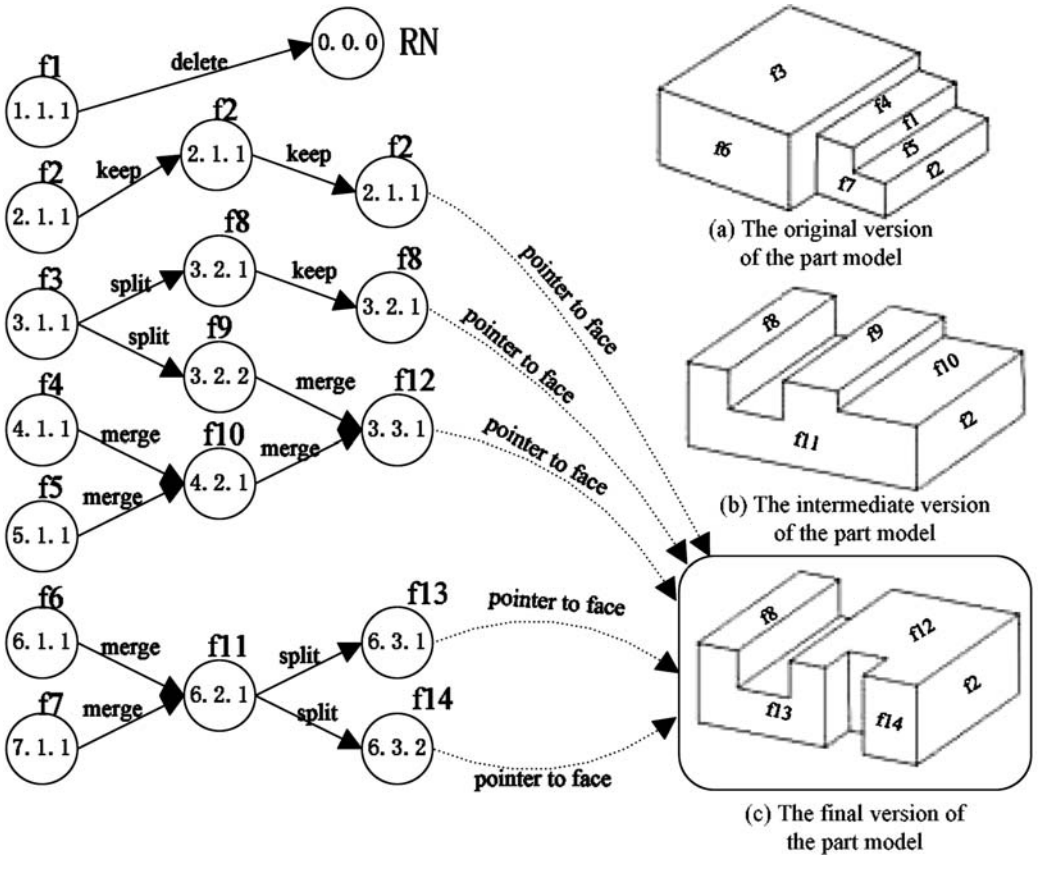


Fig. 3. The properties of an NPG

is, f2, f12, f13, and f14 are on the final version of the part model.

- Property 4: If a node contains no in-arc, the topological face corresponding to the node is a topological face on an original feature. In Fig. 3, such a type of node includes CC(f1), CC(f2), CC(f3), CC(f4), CC(f5), CC(f6) and CC(f7). f1, f2, f3, f4, f5, f6 and f7 are on original features, but these faces may be split, merged or deleted during the Boolean operation.
- Property 5: If a node contains more than on in-arc, the topological face corresponding to the node must be formed by a merging of some original topological faces. In Fig. 3, CC(f10) is a node that contains two in-arcs. One of the two in-arcs comes from CC(f4), and the other comes from CC(f5). Thus, during the Boolean operation, f4 and f5 are merged into f10.
- Property 6: If a node in an NPG contains more than one out-arc, the topological face corresponding to the node must be split into some new topological faces. In Fig. 3, CC(f3) is a node that contains two out-arcs. One points to CC(f8), and the other points to CC(f9). Thus, during the Boolean operation, f3 is split into f8 and f9.
- Property 7: If a node contain only one out-arc and the adjacent node, which contains only one in-arc, isn't an RN, the face corresponding to the node is still alive without splitting or merging. In Fig. 3, CC(f2) is such a type of node. Thus, f2 is still alive without splitting or merging during the Boolean operation.
- Property 8: If a node contains only one out-arc and the adjacent node, which contains only one in-arc, is an RN, the face corresponding to the node is deleted. In Fig. 3, CC(f1) is such a type of node. Thus, f1 is deleted during the Boolean operation.

In Fig. 3, first, f1, f2, f3, f4, f5, f6, and f7 are created. Then, f1 is deleted, and f3 is split into f8 and f9, as f4 and f5 are merged into f10. f6 and f7 are merged into f11. Finally, f9 and f10 are merged into f12, and f11 is split into f13 and f14. Only f2, f8, f12, f13 and f14 still exist on the final part model. In Fig. 3, the texts on the directed arcs indicate the propagation type, i.e., delete, merge, keep, or split.

As described in Sect. 1.2, the identification of topological faces is the basis of a TENM. An NPG represents the propagation process of the identities of faces. Thus, the propagation process of the corresponding faces can be identified from the NPG. Consequently, the genetic relation between the genetic face and the original face can be deduced. Based on the genetic relation between faces, the genetic relation between a genetic edge and an original edge can be clarified.

3 Approach of identifying a genetic topological face

As is well known, in a directed graph, the direction of an arc indicates a logic relation between the head node and the tail node. In an NPG, the direction of an arc represents the genetic relation between the head node and the tail node. The head node originates from the tail node. Based on the properties of an NPG that were mentioned before, some rules are presented to identify genetic faces.

3.1 Rules and algorithm for identifying a genetic topological face

For an arc of an NPG, let v_{head} represent the head node, v_{tail} represent the tail node. The number of in-arcs contained by a node v is called the in-degree of v and is denoted by $in\text{-}degree(v)$. Similarly, the number of out-arcs contained by a node v is called the out-degree of v and is denoted by $out\text{-}degree(v)$. Note that the v_{head} quoted in rule 1, rule 2, and rule 3 of the following isn't an RN.

- Rule 1: if $in\text{-}degree(v_{\text{head}}) = 1$ and $v_{\text{tail}} \cdot ON = v_{\text{head}} \cdot ON$, there is no topological alteration that occurs to the topological face corresponding to v_{tail}. The pattern of the sub-graph is depicted in Fig. 4a.
- Rule 2: if $out - degree(v_{\text{tail}}) > 1$, the face corresponding to v_{tail} is split into some new faces. The total num of the new faces are $out\text{-}degree(v_{\text{head}})$. The pattern of the sub-graph is depicted in Fig. 4b.
- Rule 3: if $in\text{-}degree(v_{\text{head}}) > 1$, the face corresponding to v_{head} is formed by merging some original faces. The pattern of the sub-graph is depicted in Fig. 4c.
- Rule 4: if $out\text{-}degree(v_{\text{tail}}) = 1$ and v_{head} is an RN, the face corresponding to v_{tail} is deleted during the Boolean operation. The pattern of the sub-graph is depicted in Fig. 4d.

Based on the rules discussed above, an algorithm is presented to identify the genetic topological faces in current version of a part model.

Algorithm 1: identifying genetic faces that originate from a face whose complex code is CC

Step 1 Construct a node set $\{v_s\}$ from an NPG; each node in $\{v_s\}$ contains no in-arcs.

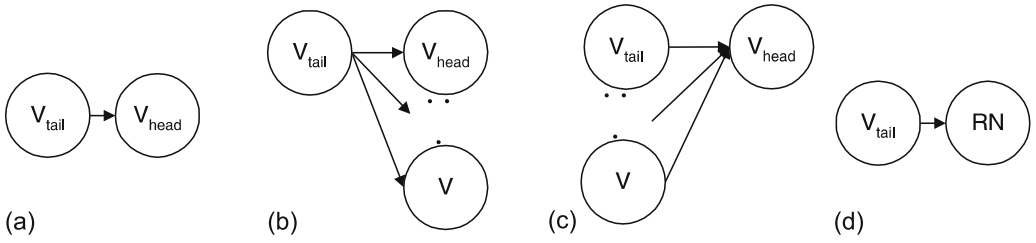


Fig. 4. Patterns of a sub-graph corresponding to the rules for identifying genetic faces

Step 2 Find the node v_0 from $\{v_s\}$ and $v_0 \cdot ON = CC \cdot ON$;
Step 3 Find a node v_1 from v_0 by a depth first search (DFS) and $v_1 \cdot seq = CC \cdot seq$, v_1 *cdotnum* $= CC$ cot *num*;
Step 4 Find a node set $\{v_{\text{dest}}\}$ from v_1 by DFS. Each node v in $\{v_{\text{dest}}\}$ isn't a RN and *out-degree*$(v) = 0$;
Step 5 Return $\{v_{\text{dest}}\}$.

The time complexity of algorithm 1 is approximately equal to that of depth first ergodicity (DFE), i.e. $O(n+m)$, where n is the number of nodes in an NPG and m is the number of arcs.

3.2 A case of identifying the genetic faces

In the example shown in Fig. 2, assume that the face f5 is chamfered in step (c). Thus, an original version of the part model is achieved, as shown in Fig. 5a. Then, in step (d), the pocket is re-edited and turned into a through-all slot. Therefore, in order to get a correct new version of the part model by re-evaluation, we should determine which faces are to be chamfered in step (d).

Now let's have a look at how to identify the genetic faces that originate from f5. As described in Sect. 2, CC(f5) = {1, 2, 2}. Therefore, according to algorithm 1, in the NPG corresponding to step (d), the node CC(f1) is first found because CC(f1) · *ON*= CC(f5) · *ON* = 1 and *in-degree*(CC(f1)) = 0. Then, using DFS, the node CC(f5) is obtained because CC(f5) · *seq* = 2 and CC(f5) · *num* = 1. Finally, the node set {CC(f7), CC(f8)} is obtained using DFS from the node CC(f5). Therefore, the faces f7 and f8 are the genetic faces that originate from f5, so both f7 and f8 should be chamfered in step (d).

The correct resulting part model is shown as Fig. 5b. The part model given by a commercial modeling system is shown as Fig. 5c. In Fig. 5c, the face f7 isn't chamfered. This case indicates that perhaps there are still some shortcomings of the TENM adopted by those commercial systems.

4 Approach of identifying genetic edges

As described in Sect. 1.2, the topological name of an edge is decided by its two adjacent faces. Thus, after the definition of complex code (CC) of a topological face, the complex code of an edge can defined by the CCs of its two adjacent faces. For example, in Fig. 6a1, the edge e is constructed by the intersection of f1 and f4. If CC(f1) = {1, 3, 2} and CC(f4) = {4, 3, 1}, the complex code of e, CC(e) = {1, 3, 2, 4, 3, 1}.

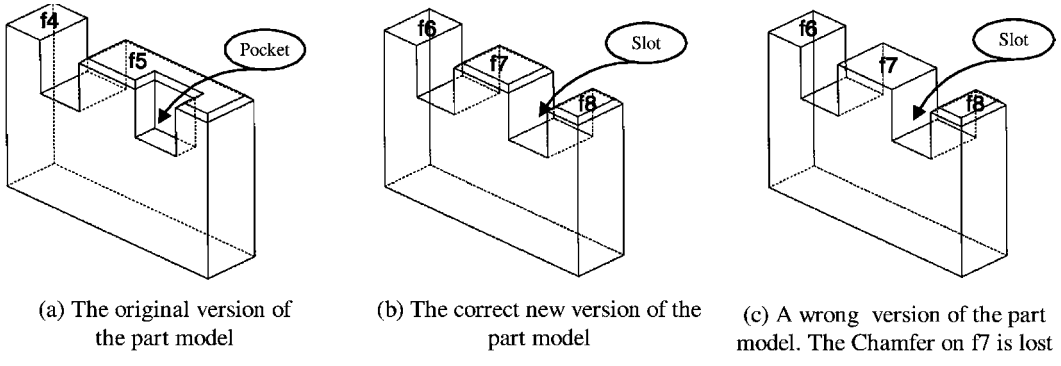
Based on the rules and the algorithm for identifying genetic faces, we present the rule and algorithm for identifying genetic edges.

4.1 Rule and algorithm for identifying genetic edges

In the original version of a part model, assume that CC is the complex code of a specific edge e and that f1, f2 are the adjacent faces of e. If e' is an edge on the current version of the part model, the following rule is presented to judge whether e' is a genetic edge of e.

- Rule 5: if the two adjacent faces of e' are the genetic faces of f1 and f2, e' is the genetic edge that originates from e.

For example, the pocket in the part model shown in Fig. 6a1 is re-edited and turned into a through-all slot; the new model is shown in Fig. 6a2. In Fig. 6a1, the adjacent face set of the edge e is {f1, f4}. In Fig. 6a2, it can be seen that the adjacent face set of $e2$ is {f3, f5}. Since f3 is a genetic face that originates from f1

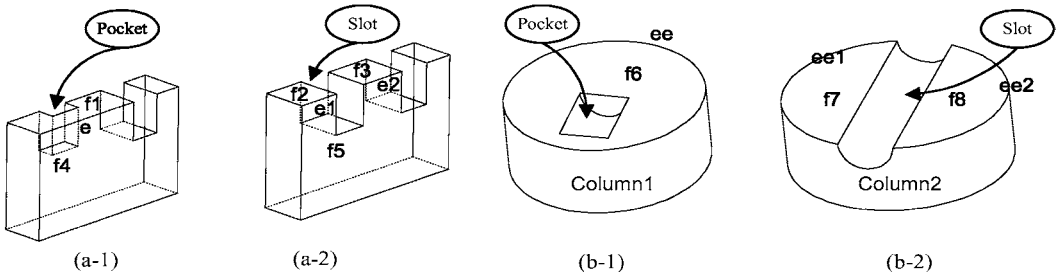


(a) The original version of the part model

(b) The correct new version of the part model

(c) A wrong version of the part model. The Chamfer on f7 is lost

Fig. 5. A case of identifying genetic faces



(a-1) (a-2) (b-1) (b-2)

Fig. 6. Two examples of genetic edges

and f5 originates from f4, according to rule 5, consequently, the edge $e2$ is a genetic edge that originates from e.

In Fig. 6b1, the edge ee represents the circle of the top face. The pocket is re-edited and turned into a through-all slot; the new model is shown in Fig. 6b2. In Fig. 6b2, the adjacent face set of the edges $ee1$ and $ee2$ is {f7, f8, $Column2$}. The faces f7 and f8 are the genetic faces of f6. The face $Column2$ is the genetic face of $Column1$. Thus, according to rule 5 described above, the edges $ee1$ and $ee2$ are the genetic edges of ee.

Based on rule 5, algorithm 2 is presented to identify genetic topological edges in the current version of a part model:

Algorithm 2: identifying the genetic edges that originate from an edge whose complex code is CC

Step 1 Obtain the complex code of the adjacent faces CC_{f1}, CC_{f2} from CC.

Step 2 If $CC_{f1} = CC_{f2}$, according to CC_{f1}, construct the genetic face set of f1: $\{f1_i \,|0 \le i \le k1\}$;
Else, according to CC_{f1} and CC_{f2}, construct the genetic face set of f1 and f2: $\{f1_i \,|0 \le i \le k1\}$ and $\{f2_i \,|0 \le i \le k1\}$.

Step 3 In the current part model, find the edge set $\{e_i \,|0 \le i \le k2\}$ that contains all of the edges formed by the intersection of the faces in $\{f1_i \,|0 \le i \le k1\}$ and $\{f2_i \,|0 \le i \le k1\}$.

Step 4 Return $\{ei \,|0 \le i \le k2\}$.

In algorithm 2, the time complexity of step 2 is far more than that of steps 1, 3, and 4, and essentially, step 2 involves algorithm 1, which finds the genetic faces according to a complex code of a face. If $CC_{f1} = CC_{f2}$, step 2 is executed only once; if $CC_{f1} \neq CC_{f2}$, step 2 is executed twice. Therefore, the time complexity of step 2 is the same as algorithm 1, i.e., $O(n+e)$. Consequently, the time complexity of algorithm 2 is $O(n+e)$ as well, where n is the number of nodes in the current NPG and e is the number of arcs.

4.2 A case of identifying the genetic edges

As described before, in the example shown in Fig. 6a1, the edge e is constructed by the intersection of f1 and f4. Thus, CC(e) is decided by CC(f1) and CC(f4). Assume CC(f1) = {1, 3, 2} and CC(f4) = {4, 3, 1}, then, the complex code of e, CC(e) = {1, 3, 2, 4, 3, 1}.

Assume that in Fig. 6a1, the edge e is chamfered and the original version of the part model is achieved, as shown in Fig. 7a. Then, in Fig. 6a2, the pocket is re-edited and turned into a through-all slot. Therefore, in order to get a correct new version of the part model by re-evaluation, we should determine which edges are to be chamfered in the part model shown in Fig. 6a2.

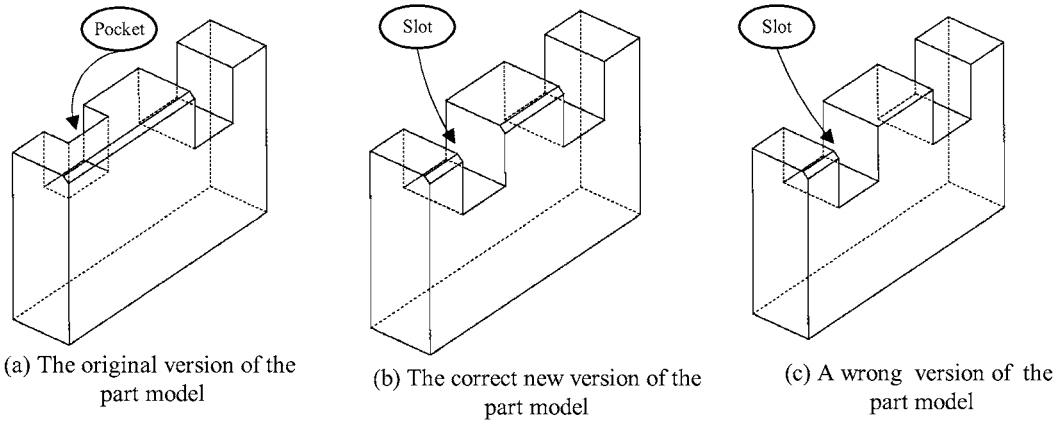
Now let's have a look at how to find the genetic edges that originate from e. As described before, CC(e) = {1, 3, 2, 4, 3, 1}. According to algorithm 2, the complex codes of the two adjacent faces is first obtained in step 1 from CC(e): one is {1, 3, 2} and the other is {4, 3, 1}. Then, in step 2, the genetic face sets that originate from f1 and f5 are found, i.e., {f2, f3} and {f5}. The edge $e1$ is constructed by intersection of f2 and f5, $e2$ are constructed by the intersection of f3 and f5. Therefore, the edge set constructed in step 3 is $\{e1, e2\}$. Consequently, the genetic edge set of e is $\{e1, e2\}$. So, in Fig. 6a2, both $e1$ and $e2$ should be chamfered. The resulting model is shown in Fig. 7b.

We also tested this case in some commercial modeling systems. However, the correct part model shown in Fig. 7b can't be fulfilled. The part model produced by those commercial systems is shown in Fig. 7c. In Fig. 7c, not all of the genetic edges that originate from e are chamfered. Obviously, it is an incorrect part model.

5 Approach of identifying a genetic vertex

Since the topological vertex won't be split or merged by the Boolean operation, it is not a very sophisticated task to identify the genetic vertex. The genetic vertex can be identified by its identity exclusively. Therefore, we only present a rule to identify genetic vertices.

Assume that CC is the complex code of a vertex $v1$ on the original version of a part model, and $v2$ is a topological vertex on current version of part model. The following rule can be used to identify whether $v2$ is a genetic vertex of $v1$.



(a) The original version of the part model

(b) The correct new version of the part model

(c) A wrong version of the part model

Fig. 7. A case of identifying genetic edges

Rule 6: If $CC(v2) = CC(v1)$, $v2$ is the genetic vertex that originates from $v1$.

6 Implementation

The algorithms discussed above have been implemented as a software module of our feature-based modeling system. The software module provides many interface functions that can be called to identify the GEs that originate from a TE.

If a feature in a design references some topological entities, the CCs of the referenced topological entities will be recorded. When a feature is added or re-edited, the part model will be re-evaluated. According to the recorded CCs, the interface functions provided by the software module will be called to identify the GRs between the new version TEs and those from the old version.

For example, consider a feature, *FILLET*, of a design that references a topological edge, EDGE. The complex code of EDGE, CC(EDGE), will be recorded. And when the part model is re-evaluated, CC(EDGE) is first retrieved, then the interface functions are called to determine the genetic entity set {NEW_EDGE} that originates from EDGE. Finally, each edge in {NEW_EDGE} will be filleted.

The software module is programmed with ANSI-C++, and thus, a different system will be supported. The software module is adopted by our history-based and feature-based modeling system. Our system is programmed with C++. The user interface toolkit adopted by our system is Microsoft Foundation Class (MFC), a software component module of Microsoft Corporation. And ACIS, a software product of Spatial Corporation, is the geometric kernel.

7 Conclusion

After analysis of the key position of a topological-entity-naming mechanism in a history-based and feature-based modeling system, two principal functional requests for a topological-entity-naming mechanism are found:

- From the static viewpoint, a naming mechanism should be capable of naming the topological entities uniquely and consistently.
- From the dynamic viewpoint, a naming mechanism should be capable of tracing the alteration of entities during the design process.

This paper presents an approach to satisfy the second functional request. Firstly, we introduce a data structure called an NPG to represent the process of identity propagation of topological entities. Next, some rules and algorithms are presented to identify genetic entities such as genetic faces, genetic edges, and genetic vertices. Several examples are also given to illustrate the effectiveness of our approach. Our approach has been implemented in our history-based and feature-based modeling system. It seems that our approach is more effective than those in some commercial modeling systems.

Acknowledgement The authors gratefully acknowledge the financial support of grant #50275060 from the National Natural Science Foundation of China (NSFC), under which the present research was possible.

References

1. Shah JJ, Rogers MT, Sreevalson PC, Hsiao DW, Mattew A, Bhatnagar A, Lieu BB, Miller DW (1990) The ASU features testbed: an overview. ASME Comput Eng 1:233–241
2. Rossignac JR (1990) Issues in feature-based editing and interrogation of solid models. Comput Graph 14:149–172
3. Venkataraman S, Shah J, Summers J (2000) An investigation of integrating design by features and feature recognition. Technical report, Department of mechanical and aerospace engineering, Arizona State University
4. Wu JJ, Zhang TB, Zhang XF, Zhou J (2001) A face based mechanism for naming, recording and retrieving topological entities. Comput Aided Des 33(10):687–698
5. Kripac J (2001) A mechanism for persistently naming topological entities in history-based parametric solid models. Comput Aided Des 29(3):113–122
6. Capoyleas V, Chen X, Hoffmann CM (1996) Generic naming in generative, constraint-based design. Comput Aided Des 28(1):17–26
7. Chen XP, Hoffmann CM (1995) On editability of feature-based design. Comput Aided Des 27(12):905–914