An Example of Knowledge-Based Query Processing in a CAD/CAM DBMS*

Arnon Rosenthal
Sandra Heiler
Frank Manola

Computer Corporation of America
Four Cambridge Center
Cambridge, Massachusetts 02142

Keywords: Hierarchy, part explosion, bill of materials, transitive closure, data model, geometric search, knowledge-based question answering, knowledge-based output reduction, geometry in databases

Abstract

Queries to part hierarchies in CAD/CAM databases (and structures with similar semantics found in other applications) can be fundamentally different from queries to sets of parts having no underlying structure, and they can raise difficult issues in query language behavior and data management. Including explicit geometric information further complicates query processing by adding computational geometry to the list of issues to be considered. In this paper, we investigate the problems of querying part hierarchies and of using the special semantics associated with such structures to improve query performance and responsiveness to user requirements. In particular, we show how the hierarchy and geometry can interact to improve query processing, and how knowledge about the behavior of attributes stored in the hierarchy can be used to choose appropriate levels of detail for query output.

*This research is funded by General Dynamics, Data Systems Division.

1. Information Model

A basic data structure frequently found in databases for computer aided design and computer aided manufacturing (CAD/CAM) applications is some form of bill-of-materials (BOM). This structure defines the component parts and subassemblies that make up each product being designed or manufactured (an assembly is a part that is composed of other parts). The classic BOM structure has a schema of the form:

```
        -------------------------
PART   | PARTID                  |
       | COST                    |<<---
       | WEIGHT                  |    |
       | MANUFACTURER            |    |SUBPARTS
       | SHAPE                   |-----
        -------------------------
```

where each record represents a different type of part (e.g., a pump, bolt, or wing). Some integrated CAD/CAM database designs attempt to combine this data with geometric data, and separate records for each distinct instance of a given part (since different instances may differ in their geometric position and connectivity to other parts) to provide a more complete physical model of the product.

We refer to this general class of data structures (including the original BOM structure) as part hierarchies, because the basic relationship shown is hierarchical and has "part of" semantics.

An interesting feature of such structures is that frequently attributes are stored in the structure whose semantics "parallel" those of the part-of relationship itself. For example, the part-of relationship implies such things as physical inclusion (of geometry -- i.e., a component part's geometry is physically within the geometry of its containing assembly), as well as other types of "dominance" (e.g., weight -- a component part's weight is a component of its containing assembly's weight). Moreover, such structures are not confined to CAD/CAM applications. Examples of other applications exhibiting similar structures include:

- Data about geographic features wherein each level down the hierarchy represents more detailed information about the components of a given feature. Examples of features might be continents, countries within them or specific features such as cities that include industrial plants, that in turn include particular buildings, etc. Values of attributes such as geometry (area) or population preserve the "part of" semantics of the hierarchy.

- Work breakdown structures, in which each task has resources such as people or time. A subtask has a subset of those resources.

Queries to part hierarchies can be fundamentally different from queries to sets of parts having no underlying structure, and they can raise difficult issues in query language behavior and data management. Even queries to the basic BOM structure can involve computation of transitive closures [Aho & Ullman, Clemons, Zloof]. Including explicit geometric information further complicates query processing by adding computational geometry to the list of In this paper, we investigate the problems of querying part hierarchies, and we investigate how to use the special semantics associated with such structures to improve query performance and responsiveness to user requirements. In particular, we show how the hierarchy and geometry can interact to improve query processing, and how knowledge about the behavior of attributes stored in the hierarchy can be used to choose appropriate levels of detail for query output.

2. Querying a Part Hierarchy

Queries to part hierarchies can have very different semantics from queries to unstructured sets of parts. In this section, we show how these special semantics can be used by the DBMS to return query results at an appropriate level of detail and to reduce the amount of search it performs. For example, if a user queries the part hierarchy of an airplane for parts within 4 feet of the instrument panel, the user probably does not want the query to return "airplane", even though, strictly speaking, "airplane" is a part in the hierarchy (the root) and various parts of the airplane geometry are within 4 feet of the instrument panel. Nor is the user likely to want the query to return every bolt in the pilot's seat; instead, the seat will often suffice to represent all its subparts.

Specifically, we categorize predicates based on how the predicate result may differ between a part and its subparts. Using these categories, we describe general rules for preventing the user from receiving unwanted output. Turning to implementation issues, we discuss how the hierarchy itself is an appropriate structure for limiting the search on many queries, and we discuss a general algorithm for searching part hierarchies. Finally, we discuss the data model implications of these new facilities.

2.1 Monotonic Predicates

Queries to a part hierarchy can be expressed: "Find parts P [in hierarchy H] where <predicate>." The variable P ranges over parts in hierarchy H. Our output limitation and search reduction rules apply to a special class of predicates called monotonic predicates. The general idea behind the notion of monotonicity is that as one goes down the hierarchy, the chances of satisfying the predicate get better (downward monotonic) or worse (upward monotonic). Hence one need not test or display all members of the part hierarchy to determine the results of predicate evaluation.

The monotonicity of a predicate is dependent on both the part hierarchy itself and on the semantics of the attribute involved. We say that an attribute is order preserving (with respect to the hierarchy) if the value of the attribute is necessarily smaller for a given part than for an assembly that includes the part. Examples of order preserving attributes might be max_dimension(P) or weight(P). We say that an attribute is inverse order preserving (with respect to the hierarchy) if the value of the attribute is necessarily larger for a given part than for an that assembly includes the part. An example of an inverse order preserving attribute might be hierarchic_depth(P). We assume that the DBMS can be made aware of which attributes are order or inverse order preserving according to these definitions.

A predicate Pred is downward monotonic with respect to hierarchy H if whenever a part satisfies Pred, all its descendants (subparts) satisfy Pred. For many purposes it is not necessary to return subparts of a part that satisfies a downward monotonic query. Since the number of subparts can be enormous, omitting them can greatly reduce the output size and search time.

Some downward monotonic predicates (where P ranges over parts) are:

1. Attribute(P) < constant, where Attribute is order preserving.

2. Attribute(P) > constant, where Attribute is inverse order preserving.

3. P is a descendent (subpart) of assembly Q0 (for any fixed Q0).

4. AND or OR of downward monotonic predicates.

5. NOT (upward monotonic predicates).

6. Part P is contained in Region R (see Section 3, Geometric Queries).

A predicate Pred is upward monotonic with respect to hierarchy H if whenever a part satisfies Pred, its ancestors (superparts) satisfy Pred. Since no additional information is conveyed by returning the superparts, they can be omitted from the response to the user. This reduces the size of the output somewhat, but not as dramatically as with downward monotonic predicates.

Each downward monotonic predicate above has an upward monotonic analog (obtained by (1) reversing the inequality, (2) testing for superpart rather than subpart, etc.).

Some upward monotonic predicates are:

1. Attribute (P) > constant, where Attribute is order preserving.
2. Attribute (P) < constant, where Attribute is inverse order preserving.
3. P is a superpart (ancestor of) part Q (for any fixed Q0).
4. AND or OR of upward monotonic predicates.
5. NOT (downward monotonic predicates).
6. Part P contains Region R.
7. Part P (contains or properly intersects) Region R.

(The last two predicates are described further in Section 3, Geometric Queries.)

2.2 Output Limitation

A query to the part hierarchy identifies a set of nodes that satisfy the query predicate. Here we present a list of convenient rules for further limiting the query result. The rules take the form: "Do not show the descendants (or, less frequently, the ancestors) of parts that satisfy certain conditions." The user can specify how terse the output should be. This terseness specification determines which of the rules should be invoked. Usually, default rules are in effect.

A two-faced predicate is one that can be expressed as (pred_u OR pred_d), where pred_u is upward monotonic and pred_d is downward monotonic. We propose the following rules for preventing "uninteresting" parts in the hierarchy from being returned in response to a query:

Return Smallest satisfactory part: If pred is upward monotonic and P satisfies pred, do not show ancestors of P.

Return Largest satisfactory part: If pred is downward monotonic and P satisfies pred, do not show descendants of P. More generally, if pred is two-faced, do not search below parts which satisfy pred_d.

Any order preserving or inverse order preserving attribute can be treated as a measure of a part's importance (e.g., weight, maximum dimension, hierarchic level, cost). A part is important if the measure exceeds (or falls below) some threshold value.

Return Important Parts: When the search encounters an unimportant part P that satisfies Pred, the search stops and returns P but not its subparts.

Return Consistent Levels: If Pred is two-faced and P satisfies pred_d, do not search below any siblings of P that satisfy pred.

Other rules are possible. For example, Return Smallest could be modified to provide context information by outputting a specified number of levels above the smallest part returned. Alternately, it could be modified to suppress parts that were too important to be included for context (e.g., "airplane").

A particular system could choose terseness defaults to avoid swamping the interactive user with data and to give responses quickly. Because rules Return Largest and Return Consistent Levels greatly reduce the data volume, they might default to "on" (so they would be in force if the predicate is appropriate). Similarly, Return Important Parts might default to "on" if an importance threshold has been defined.

On the other hand, Return Smallest reduces output volume and the search is accelerated, somewhat, but the user receives no context information. It probably should default to "off".

2.3 Access Structures for Part Hierarchy Searches

Like the determination of appropriate query output, the design of access structures also can involve consideration of the special semantics implied by the part hierarchy. We consider here the particular case of part hierarchies in CAD/CAM databases that we call instance hierarchies. In an instance hierarchy, each appearance of a part in the final product is separately represented. Hence one can attach information to instances that will identify the part globally (e.g., date manufactured, serial number, and position in the end product).

In an instance hierarchy, there is a choice between representing a part's position with respect to its immediate ancestor, or its (unique) end item. The latter choice means that moving a large assembly may require hundreds of database updates for changing the positions of each of its component subparts, the positions of their components, etc. This is infeasible in an interactive environment. As a result, the former choice is common in many CAD/CAM systems (although in geographic systems the opposite choice often is made, using absolute latitude and longitude).

If geometric information about a part is available relative only to the immediate superassembly, information about the global location of the part (with respect to the overall product) would have to be reconstructed by traversing the hierarchy. In such a case, it is not feasible to use spatial index structures, since these require global information. The hierarchy itself is the only usable search structure.

The hierarchy has more levels than a B-tree, but the number of levels still is moderate. A brief analysis suggests that the search time for most cases will be substantial but bearable. For example, consider an airplane with 100,000 parts in the hierarchy. Assuming that each assembly is composed of 10 subassemblies, the leaves are five levels below the root. Each of these levels requires accesses to one Part record (in an instance hierarchy), or to one Part record and a cluster of (contiguously stored) Part_instance records for that part. The search time will be substantial, but the alternatives are sequential search of the database or manual search through drawings.

Use of the hierarchy as the search structure also facilitates application of the output limitation rules. If, instead, positions were stored relative to a top level assembly, one would then need a storage structure for a geometric search that permitted the output limitation rules to be applied. This problem is not addressed even by storage structures designed for searching for non-point objects [Guttman].

2.4 The Search Algorithm

Our search algorithm exploits the fact that for any assembly only a few of its next level parts are likely to be needed in processing the query. The algorithm tries to detect two types of situations. The first type includes situations in which no member of a subtree can possibly satisfy the search predicate. For example, it is sometimes possible to determine that no part of the elevator hinge satisfies the predicate "is within 3 feet of the pilot's seat" by testing the positions of higher level parts (e.g., "tail assembly") that contain the elevator hinge. If the tested part is not within 3 feet of the pilot's seat, subparts of the tested part need not be individually checked.

When a predicate is evaluated on a part P, the search algorithm sometimes acquires the answers to several questions at the same time by making deductions from previous tests and monotonicity. The information exploited by the query processor is: does P satisfy the query predicate? (yes/no); does any descendant of P satisfy the predicate? (yes/no/don't know); do all descendants of P satisfy the predicate? (yes/no/don't know). In addition, when testing a two-faced predicate, both pred_d and pred_u are evaluated. This helps the algorithm detect situations in which an output limitation rule inhibits the return of subparts.

We conceive of query processing as going on in two stages. First, the algorithm walks the part hierarchy, avoiding further search below a part if neither the part nor any of its descendents could possibly appear in the output. Such parts are called <u>irrelevant</u>. The algorithm builds a hierarchical structure that includes all relevant parts, together with flags that indicate whether the parts seemed to satisfy the predicates.

The nodes of the result hierarchy will include pointers to parts in the original hierarchy, and annotations about whether the part or its subparts are known to satisfy the predicate or could possibly satisfy the query predicate (if the subtree was not searched), and about whether an output limitation rule asserts that the node (or its descendants) should not be returned.

Note that the algorithm must simultaneously test all children of a part, because in <u>Return Consistent Levels</u> the decision about searching below one child may depend on the predicate result for a sibling. Also, some storage structures keep all the information about children together, so it would be desirable to use this information before it is paged-out. During the search the predicate can be modified to remove downward monotonic conjuncts that are satisfied by some child. Each relevant child node that is searched must be added (with annotations) to the result hierarchy.

Once the result hierarchy has been produced, output is prepared by walking the result with the following restrictions.

1. If no output limitation rules are in force, then walk the result hierarchy in preorder, printing each part that satisfies the predicate, with appropriate format.

2. If the <u>Return Smallest</u> rule is in force, then walk the result hierarchy in postorder, printing parts that satisfy the predicate <u>and</u> that have not had a descendant printed.

3. If the Return Largest rule is in force, then walk the result hierarchy in preorder, but stop the downward traversal when encountering a part that satisfies the predicate.

2.5 Data Model Implications

Database researchers have devoted very little attention to defining appropriate semantics for queries to part hierarchies. This reluctance stems not from lack of user interest, but from the well known difficulty of using a conventional language to specify queries that involve arbitrary-length paths through a set of instances. In fact, [Aho and Ullman] showed that the query formulations in conventional languages (e.g., SQL, QUEL) cannot express the query "Is P1 descended from P2 in the hierarchy?"

Systems implementors have provided some facilities for manipulating bill-of-materials data [Zloof]. Instead of determining level of detail based on properties of part hierarchies, this approach specifies in advance how many levels of data should be considered. A more general enhancement would support transitive closures as part of a facility for defining and manipulating recursively defined structures.

The output limitation rules and query processing strategy we have described are based on monotonicity. The system components doing query optimization and query execution must be informed about monotonicity of each predicate or each attribute.

As noted in Section 1, the structures in which our output limitation rules make sense seem to have two important characteristics:

a. They involve a partially ordered set whose ordering relation is based on a "part-of" relationship.

b. The entities in the hierarchy possess one or more attributes that are order preserving or inverse order preserving, as defined above.

Most conventional data models do not support an adequate indication of the recursive structure of the part hierarchy, or the fact that the hierarchic order is preserved by attributes such as weight, or by predicates such as "set intersection".

This suggests a study of enhanced data models that include declarations of part hierarchies and information about order preservation. We have shown how this kind of knowledge can improve performance for queries that involve these hierarchies. But it would be desirable to more fully integrate the hierarchical and ordinary processing in a query. The DBMS also could use knowledge about hierarchies to ease declaration and enforcement of integrity constraints, such as the fact that a part hierarchy has no directed cycles.

The output limitation rules can be implemented by predicates that are added at the source level to restrict a query's output. The query processor would be augmented with strategies for optimizing access via such predicates. Further research is needed to determine how well these special strategies can be integrated with an existing query optimizer's cost model and exploitation of predicates.

3. Geometric Queries

Geometric queries are useful for returning a list of parts or drawings relevant to a specified volume of space (here called a region). They also may be used for interference checking and for general orientation when performing CAD/CAM. The geometric predicates we propose support queries about the positions of parts relative to each other or to spatial regions. For example, one could ask: "What are all the parts in region R?" or "What are the parts that intersect the cross-sectional plane P?". In addition, one could ask queries that combine geometric information (which often is available only in drawings) with data usually available only from a database. For example: "What are all the plastic parts contained in region R?" or "What are all the third-level assemblies in region R?". Asking the query with only one kind of restriction can produce an unmanageably large set of parts.

This section describes how we represent and manipulate the shapes (called envelopes) of parts and regions. It then illustrates the output limitation rules on an example query that includes geometric predicates. Finally, it considers the impact of using an approximate rather than exact representation of the shapes of parts. It proposes a particular approximate geometry and describes the effects on query processing of using the approximation.

3.1 Envelopes

Each part's shape is represented by an envelope, a region expressed in the coordinate system of some part in the hierarchy. Each part type has a local coordinate system, together with an envelope that represents the shape of the part defined in that local coordinate system. This envelope is called the part's basic envelope. Normally a part's basic envelope is input to the DBMS. If the part is an assembly composed of other parts, its basic envelope may be either directly input or computed from the envelopes of its component parts. An envelope for a part must always wholly contain the part.

The position of a part within a next-level assembly is represented by the transformation that places the part's basic envelope in the assembly's coordinate system. In general, the transformation may involve translation, rotation, and scaling operations.

3.2 Operations on Geometric Envelopes

An envelope is defined as an abstract data type. Thus the actual representation of envelopes is known only to the envelope type operators. We provide operators to:

1. Retrieve a part's basic envelope.
2. Transform the coordinate system of an envelope (usually to the coordinate system of the basic envelope of another part).
3. Return the smallest xyz-box that contains envelope E, in the current coordinate system (an xyz-box is a rectangular box whose edges parallel the coordinate axes).
4. Extend an envelope by a specified distance along each axis.
5. Return the distance between the closest points in two envelopes.
6. Test geometric predicates on envelopes (as described in the next section).

The basic manipulations of geometric objects are unaffected by the part hierarchy. However, the output limitation rules and control over change between different coordinate systems require knowledge of the hierarchy, as described in Section 2.3.

Operators such as union, intersection, and difference, which explicitly combine envelopes to form a new envelope, also could be provided.

3.3 Directional regions

We will supply functions that implement the concepts "forward" and "back" (along any axis), in order to support queries such as "find all plastic parts forward of part xyz". The user must specify the kind of region desired. For example, suppose we are defining a box to represent the concept of "forward" along the x axis from box B. One option has the resulting box begin at the leftmost point of B, while another begins at the rightmost point. Another choice made by the user is whether the resulting region "forward from B" is bounded in the y and z directions by the same boundaries as B, or whether it extends infinitely in those directions.

3.4 Geometric Predicates

The geometric predicates that are supported are:

1. x properly intersects ("cuts") y (i.e., x and y intersect, but neither is contained in the other).
2. x contains y.
3. x contained in y.
4. x intersects y [meaning: (x properly intersects y OR x contains y) OR x is contained in y].

   Note that "intersects" is two-faced. The parenthesized term is upward monotonic (in x), while "is contained in" is downward monotonic.

If x and y are envelopes in the same coordinate system, interpretation is straightforward. In other cases, the query processor must supply a transformation that puts the envelopes in the same coordinate system.

We also allow the geometric predicates to use a part as an argument. In that case, the predicate is applied to the part's envelope. For example:

- Find P in hierarchy H where P contains x.
- Find P in H, I in Inventory where (P intersects x) and (P.P#=I.P#) and (I.quantity_in_stock < 10).

3.5 Example of a Geometric Query

The following query shows all parts near (within 6 cm) of the eject switch.

```
Temp = Find parts P where P intersects
       Enclosing_envelope
       (eject_switch, 6 cm along each axis)
```

Suppose the entire altimeter, a piece of the radio, and a piece of the pilot's seat are within 6 cm of the switch. Then:

- Return Smallest prevents the return of "airplane" or "cockpit", because some smaller parts are being returned.
- Return Largest means don't search below the altimeter.
- Return Consistent Levels says the radio is a sibling of the altimeter, so don't search below the radio.

- Return Important Parts says the front panel of pilot´s seat is unimportant, don´t decompose it.

3.6 Use of an Approximate Geometry

In a CAD/CAM environment, the most exact representations are available on paper drawings and on workstations that have facilities for solid modeling. These facilities cannot reasonably be duplicated in a DBMS. In addition, we want the facility to be useful even with crude approximations entered manually.

We find that a box provides a decent representation for most solid parts, even spheres. A part´s home coordinate system usually is chosen so that some axis parallels a long dimension of the part. As a result, we initially use relatively simple envelopes to represent part shapes. Initially, each envelope will be a single rectangular box that contains the part, and each basic envelope will be a box whose edges parallel the axes of the home coordinate system (termed an xyz-box).

The decision to use a box as the basic representation was influenced by the fact that eventually we may represent envelopes as sets of boxes. The generalized representation would permit an assembly to be represented as the union of the boxes of its components. The representation of a part could be refined to any desired accuracy, and one could implement intersection and difference of sets of boxes (oriented along the same set of axes). If necessary, a tilted box could be replaced by an xyz-box that enclosed it and that had axes parallel to the axes of the current coordinate system. A box can be displayed along any of its axes.

Other reasonable representations include xyz-boxes from a fixed set of sizes, spheres, or more general shapes with appropriate manipulation primitives from constructive solid geometry. xyz-boxes that have a fixed set of alternative sizes probably will yield a poorer approximation (especially if one wants to use a single box for each part), but storage may be more efficient, and difference operations between boxes will yield fewer slivers.

Spheres can be specified without knowing the coordinate axes, and changes to the coordinate system, rotations, and predicate testing (e.g., intersection) are extremely easy. But spheres are an even poorer fit than boxes to many shapes; approximation cannot be refined as a union of disjoint spheres; a display of a sphere will supply little visual information.

One also could provide for alternative representations of a shape. One alternative might be used for display, another for fast predicate testing (spheres excel here), and another for more detailed tests. Algebraic expressions over envelopes (union, difference) could provide even better approximations. (For example, difference would be useful in representing hollow shapes.)

3.7 Interpretation of Geometric Predicates

Because a part´s basic envelope may contain space not included in the part, geometric predicates cannot always return definitive answers. The test of the predicate can use only the part´s envelope, not its exact representation. We can determine whether:

[Envelope E properly intersects (cuts) E´] (yes/no).

But the predicate:

[Part P properly intersects E´]

yields the results "no" or "don´t know", since P is possibly smaller than Envelope(P).

Similarly:

[P contains E´]

yields "no" or "don´t know".

If E´ was obtained as the envelope of a part P´, one cannot conclude anything about the relations between P and P´, but only about relations between their envelopes.

The predicate "P is contained in E" must be interpreted as yielding "yes" or "don´t know". Its usefulness seems limited, because returning only parts where the answer is "yes" means that one has omitted some of the desired parts. Fortunately, intersection is the predicate most useful for applications, so even approximate geometry will support useful facilities.

4. Conclusion

Part hierarchies (and similar structures) and approximate geometries that might be used in CAD/CAM database systems have special semantics that need to be taken into consideration in the design of such systems. These semantics create interesting problems in query interpretation, but they also offer the potential for improvements in query performance, and in the systems´ ability to return appropriate data to users. This paper has explored some of these issues, and has presented methods for dealing with the special semantics involved. Research continues in an effort to incorporate these methods in an operational database system for CAD/CAM data.

References

A. Aho and J. Ullman, "Universality of Data Retrieval Languages," Proc. Sixth ACM Symposium on Principles of Programming Languages, pp. 110-120.

E.K. Clemons, "Design of an External Schema Facility to Define and Process Recursive Structures," ACM TODS, 6(2), June 1981, 295-311.

A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," Proc. SIGMOD ´84, ACM, New York, 1984.

M.M. Zloof, "Query by Example: Operations on the Transitive Closure," Res. Rep. RC5526, IBM Research Center, Yorktown Heights, N.Y., Oct. 1976. (The essentials of this report appear in C.J. Date, Introduction to Database Systems, 1981).