# Minimum Cost Aspect Classification: a Module of a Vision Algorithm Compiler

Ki Sang Hong Katsushi Ikeuchi
Keith D. Gremban

April 1990
CMU-CS-90-124$_3$

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was supported by the Defense Advanced Research Projects Agency (DOD) and monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFSC), Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-87-C-1499, ARPA Order No. 4976, Amendment 20. K.S. Hong was supported by KOSEF: Korea Science and Engineering Foundation.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, KOSEF or the U.S. government.

**Keywords:** computer vision, vision and scene understanding, automatic programming

## Abstract

An important issue in building a model-based vision system is how to extract and organize the relevant knowledge of an object, and systematically turn this knowledge into a working vision system. One approach is to use a vision algorithm compiler, which utilizes stored models of objects, sensors, and processing operations to automatically generate a working vision system. In this paper, we discuss the design of one module of an optimizing vision algorithm compiler which determines the minimum-cost sequence of operations needed to classify an object into an aspect. Given the costs of various feature extraction operations, the module searches over the space of possible classification strategies for the combination of operations that minimizes the expected cost. The optimal strategy is compiled in the form of an *aspect classification tree*. The classification tree may be expensive to compile, but this cost is incurred off-line, and may result in significant savings at run-time. The performance of the module is illustrated with several examples.

# Contents

# 1 Introduction

A critical issue in building a model-based vision system is how to extract and organize the relevant knowledge of an object, and systematically turn that knowledge into a vision program. Traditionally, vision systems have been developed based on the implementors' insight into the specific problem. Although the resulting system may be effective and efficient, this *hand-coding* method requires large amounts of time, experimentation, and deep vision expertise for building application systems. For complex applications, a prohibitive amount of resources may be required for development.

A vision algorithm compiler (VAC) represents a potential alternative to the resource intensive hand-coding method of vision system development. A VAC utilizes stored object models, sensor models, and analytic models of processing operations to construct programs to perform specific vision tasks.

It is unlikely that a single VAC can be built to compile programs for any arbitrary vision task. The range of possible vision applications, as well as the current state of the art, forces the development of specialized VACs that compile programs for fairly specific tasks. In particular, we are working on a VAC to perform object localization tasks, in which the object is known, but its configuration (position and orientation) must be determined. Our compiler is therefore specialized, and we refer to it as a VACL (Vision Algorithm Compiler for Localization). Many of the components of a VACL have been demonstrated. In particular, [15] discusses a geometric modeler developed for use in computer vision. Sensor models are discussed in [13], while the compilation of vision programs has been demonstrated by [9, 5, 11].

Although the programs generated by existing VACLs correctly solve the intended tasks, they are not guaranteed to do so efficiently. Yet, most applications are required to be cost-effective. Efficiency is particularly important in vision applications, which are computation-intensive to begin with. The computational costs of vision operations can vary dramatically; for example, the cost of computing the Gaussian curvature at every pixel in an image is several orders of magnitude more costly than computing the intensity gradient at every pixel. A practical VACL must be an optimizing compiler, and select the combination of operations that can perform the task for the minimum computational cost.

In this paper, we present the design of a VACL module that is used to construct efficient vision programs for a subtask of object localization, called aspect classification. An aspect is a class of topologically equivalent object appearances [16]. Intuitively, an aspect can be associated with a range of viewing positions. To perform object localization, we first classify an object into an aspect in order to obtain a rough estimate of an object's configuration; this is followed by a numerical minimization procedure to precisely locate the object [12]. Our compiler module generates an optimal strategy for aspect classification, in the sense that the average cost of classification is minimal.

Given the costs of various primitive operations, the module searches over the space of possible strategies for the combination of operations that minimizes the expected cost. The optimal strategy is stored in the form of a tree of operations, the *aspect classification tree*. Classification of an object into an aspect starts at the root node and works down to a leaf. At each node, a test on a feature is indicated - the feature is computed and the next node is selected based on the outcome of the test. The classification tree may be expensive

to compile, but this cost is incurred off-line, and results in significant savings at run-time, assuming that the application is performed much more frequently than compilation.

## 1.1 Background

A model-based vision system requires that the relevant knowledge about the task, environment, sensors, and objects be determined and programmed into the system as models. Traditionally, the process of extracting and programming models has depended on the insight of the implementor into the specific problem. Representative examples include systems built by Oshima and Shirai [17], Perkins [19], and Ikeuchi and Horn [14]. While the final system may be both effective and efficient, the construction of such *hand-coded* systems requires extensive investments of time from expert implementors.

The problem is made somewhat easier when a geometric model of an object is available. Such a model, as generated by a CAD system for instance, represents the 3D shape of an object by means of polyhedra, generalized cylinders, or other primitives. Given such a model, an object can be localized in space by matching features of the model to features found in an image. The abstract method for performing the match consists of minimizing a difference measure between the features of the model projected into the image, and the features extracted from the image [2]. The pioneering vision system by Roberts [20] can be viewed as an instance of such a method. Roberts reduced the problem of object matching to that of estimating the parameters of transformation (rotation, translation, size, and projection) by minimizing a matching error between model vertices and image joints.

Grimson and Lozano-Perez [10] formulated the problem of object localization measurements within a generic hypothesize-and-test search paradigm. Given a set of observations and a set of polyhedral object models, the possible matches of observations to model surfaces are expanded as a search tree. The matcher prunes the tree by using relational constraints between pairs of measurements which are imposed by the object models. For example, if the distance between two observations is inconsistent with an hypothesized match to an object, then that portion of the tree is pruned. The method has been applied to 2d and 3d object recognition using sparse range, touch, and orientation sensory inputs.

Faugeras and Hebert [8] proposed the *recognizing-while-positioning* paradigm, a general paradigm which combines the processes of object recognition and object localization. Given an input image, a standard set of operations is performed to extract linear primitives, and matching is performed between model and image primitives using a rigidity constraint to reduce the size of the search space. A tree search technique consisting of hypothesis formation, prediction, and verification phases is used to explore the search space.

The ACRONYM system by Brooks [6] modeled objects as generalized cylinders and their spatial relationships. Recognition or matching of the models to an input image is performed by using a symbolic algebraic reasoning system which reasons about projection and relational constraints on geometry. ACRONYM succeeded in recognizing airplanes in aerial images.

The systems cited above used geometric models of objects in conjunction with some generic reasoning mechanism: numerical optimization, tree search, or symbolic constraint satisfaction. A common characteristic is that each system uses the object model interpretively; that is, the knowledge is extracted from the model and used to formulate the

execution strategy at run time. The systems are general, but not necessarily efficient, even for specific objects.

One way to increase efficiency is to compile an execution strategy. That is, the relevant knowledge is extracted from the object models and compiled into an object recognition strategy off-line so that as little computation as possible is performed at run time. Interestingly enough, some early vision work can be viewed as examples of compilation. The generalized Hough transform of Ballard [1] and the direction coding method of Yoda, Motoike, and Ejiri [21] can be regarded as compiling the object shape in the appropriate transform so that the recognition reduces to peak finding in a histogram. However, these methods have limited applicability.

Goad [9] presented one of the first and most systematic methods for automatic generation and compilation of object localization programs. His basic localization procedure is to pick a model edge and try a match to an image edge. Based on that match, estimate the configuration of the object to constrain the search for additional matching edges, and iterate until the object configuration has been determined. The system compiles in advance the sequence of matches, and also pre-computes various conditions and data structures to be used at run time. However, the sequence of matches is specified by hand, and the system is not fully optimized for speed.

Bolles and Houraud [4] noted that one of the most important features of an industrial vision system is speed. To achieve high speed parts orientation, they proposed a compiled system using the *local-feature-focus* strategy [3]. A focus feature is any feature which is relatively distinct and easy to recognize. Matching begins with a focus feature. Once a hypothetical localization is established based on focus features, the localization is verified using other features. By compiling the choice of focus features and the features to be used in verification, the system is able to perform localization quickly. However, both focus features and secondary features are selected by hand.

We have been developing a completely automatic system that can compile a strategy for object localization [12]. Given a CAD model of the object, the system generates apparent shapes of the object and classifies these shapes into groups. A set of computable features is analyzed to generate an interpretation tree that can be used on-line to classify an object into one of the groups. A second interpretation tree, using an additional set of features is also generated to determine the attitude of an object within each group. No attempt was made to optimize the processing, however.

The current paper extends on the system of [12] by compiling a minimum-cost strategy for the first stage of object localization.

## 1.2 Organization

The organization of the paper is as follows. Section 2 presents the details of our method. We discuss our procedure for aspect classification and object localization, the available features and their associated costs, and the method used to generate a program for aspect classification. The method has been implemented in a VACL module, and section 3 presents the results for generation of minimum cost aspect classification trees on several example objects. A discussion and recommendations for further research conclude the paper.

# 2 Strategy Generation

Our goal is to automate the process of generating an aspect classification strategy that minimizes the expected computational cost of classification. Minimizing the computational cost has the effect of increasing the speed with which aspect classification can be performed. In order to proceed, certain information must be available. In specific, the following information must be available to the VACL:

- a definition of the aspects to be discriminated;
- the features available for classification;
- the costs to compute each feature.

In this section, we will discuss each of the points above. We begin by describing the problem of aspect classification and explain its role in the larger context of object localization. We describe a collection of features which can be extracted from images for use in aspect classification, and determine the computational cost of each feature. We conclude this section by presenting the method used in our VACL for generating minimum cost classification strategies. A VACL module has been implemented using the method discussed here; the performance of this module on several different objects is presented in section 3.

## 2.1 Aspect Classification and Object Localization

Aspects are defined to be topologically distinct classes of object appearances [16]. Less formally, we can consider an aspect to be a collection of similar views of an object. For example, consider the case of a convex polyhedron. As the polyhedron rotates with respect to the viewer, different faces appear and disappear. There will be ranges of viewing positions within which a particular collection of faces will be visible, and within which the polyhedron looks roughly the same. Such a collection of faces or views constitutes an aspect.

In our VACL, we are not concerned with having a fine partition of the set of object appearances. Instead, we combine distinct aspects into larger sets that still maintain the property of "looking roughly the same". In particular, we select a group of significant faces, and then choose our aspects from a set of images distributed uniformly around the unit sphere based on the visibility of the selected faces. (See [11].) Each aspect is represented by the average of the features extracted from the images within the aspect. The image generated from the average view position is used to illustrate an aspect.

Figure 1 illustrates a collection of aspects for a simple polyhedral object.

Object localization is the process of determining the precise position and orientation of an object. Our VACL generates a two step strategy for object localization:

1. classify an image of an object into an aspect (aspect classification);
2. using aspect information as a starting point, determine the precise position and attitude of the object (configuration determination).

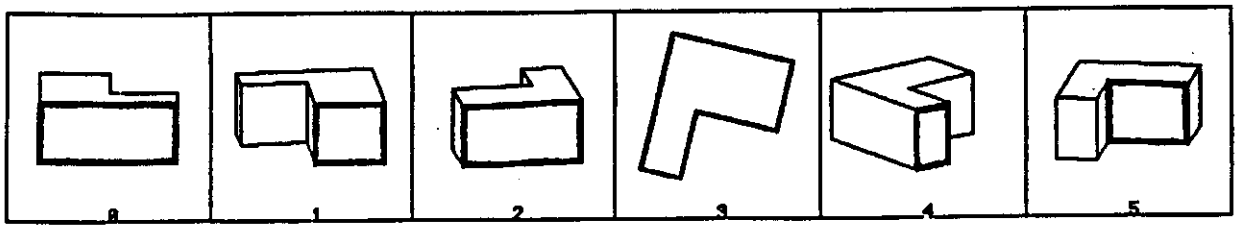


Figure 1: **Aspects of a Simple Polyhedral Object**

Aspects are topologically distinct classes of object appearances. An aspect is represented by a single image corresponding to the average view within the aspect. Objects are depicted as wire frame drawings. Dark lines distinguish detectable faces.

Aspect classification provides a rough guess of an object's configuration. By classifying the image of an object as an instance of a specific aspect, the object is constrained to lie within a relatively small range of configurations. Since an average view is associated with each aspect, the configuration of the object in the average view can be used as an initial estimate of the true configuration.

To perform configuration determination, the configuration parameters from the initial estimate are used as the starting parameters for a numerical optimization procedure that determines the precise attitude of the object [12]. Classifying an image of an object into an aspect limits the range of possible attitudes of the object, and makes the optimization more efficient, as well as reducing the likelihood of a converging to a false minimum.

## 2.2 Features and Costs

To determine the value of any feature, a certain amount of preprocessing is necessary. The cost of the preprocessing common to all features has not been included in the overall cost computation. Instead, we assume that a minimum amount of processing takes place prior to classification. In particular, the input image has been broken down into components, which are the visible regions in the image, and the following arrays have been computed:

- the *needle map* which contains the gradient space $(p, q)$ values at each pixel, and
- the *label map* which indicates the region to which each pixel belongs.

The computational cost of each feature can be obtained by analyzing the number of operations required to compute the feature value. We denote the largest visible region in the image as the *target region*; neighboring regions are *brother regions*. Our current implementation prepares the following collection of features for aspect classification:

**F1:** *the area of the target region*

The area of the target region is simply the number of pixels with that region label.

**F2:** *the dominant eigenvalue of the target region*

The inertia matrix of a region characterizes the distribution of the region in the image. The dominant eigenvalue of the inertia matrix is a measure of the spatial extent of the region.

**F3:** *the eigenvalue ratio of the target region*

The ratio of the eigenvalues of the inertia matrix of a region is a measure of the elongation of the region.

**F4:** *the number of brother regions*

**F5:** *the distance between the largest brother region and the target region*

The distance is defined to be the distance between the centroids of the regions.

**F6:** *the angle between the principal axes of the largest brother region and the target region*

The principle axes of a region are the eigenvectors of the inertia matrix of the region.

**F7:** *the surface orientation difference between the largest brother region and the target region*

The difference between the average surface orientation of the target region and that of the largest brother region is used.

**F8:** *the area of the largest brother region*

**F9:** *the dominant eigenvalue of the largest brother region*

**F10:** *the eigenvalue ratio of the largest brother region*

**F11:** *the surface characteristics of the target region*

The surface characteristics of a region are the average Gaussian and mean curvatures.

**F12:** *the surface characteristics of the largest brother region*

**F13:** *the surface characteristic distribution of the target region*

At each pixel, the surface characteristics are used to classify the local surface as being one of the following: hyperbolic, positive elliptic, negative elliptic, cylindrical, or planar. Comparing surface characteristic distributions requires comparing surface types pixel by pixel across matching regions.

**F14:** *the surface characteristic distribution of the largest brother region*

We examine the computational cost of the above features in terms of the number of operations involved. In order to calculate computational costs features, we use the following notation:

| Feature Number | Array Ref R | Test X | Add/Sub Y | Mult/div Z | Square S | Trigonometry T |
|---|---|---|---|---|---|---|
| F1 | A | A | a | 0 | 0 | 0 |
| F2 | A | A | 6a | 3a | 0 | 0 |
| F3 | A | A | 6a | 3a | 0 | 0 |
| F4 | B | B | 0 | 0 | 0 | 0 |
| F5 | B | B | 3a+3b | 0 | 0 | 0 |
| F6 | B | B | 6a+6b | 3a+3b | 0 | 0 |
| F7 | B+2a+2b | B | 3a+3b | 0 | 0 | 0 |
| F8 | B | B | b | 0 | 0 | 0 |
| F9 | B | B | 6b | 3b | 0 | 0 |
| F10 | B | B | 6b | 3b | 0 | 0 |
| F11 | A+5a | A | 19a | 25a | 3a | 0 |
| F12 | B+5b | B | 19b | 25b | 3b | 0 |
| F13 | 2A+8a | 2A+3a | 28a | 32a | 3a | 4a |
| F14 | 2B+8b | 2B+3b | 28b | 32b | 3b | 4b |

Table 1: **Computational Costs of Features**

*A*= the rectangular search area size of the target region

*B*= the rectangular search area size of the brother region

*a*= the area size of the target region

*b*= the area size of the brother region

*X*= unit computational cost of one test

*Y*= unit computational cost of one add/subtract operation

*Z*= unit computational cost of one multiply/divide operation

*R*= unit computational cost of one array reference operation

*S*= unit computational cost of one square root operation

*T*= unit computational cost of one trigonometric operation

Using these notations, the computational costs of the features are presented in table 1. Please refer to Appendix A for a detailed derivation of the costs.

## 2.3 Searching Possible Strategies

### 2.3.1 Determining the Cost of Aspect Classification

A *classification tree*, sometimes called a *decision tree*, is a tree in which each node represents a collection of classes and an associated test, and arcs represent the possible results of a test. Leaf nodes represent the final results of classification. A classification tree is a convenient framework for optimizing aspect classification, since using a classification tree permits comparisons (and hence feature extraction operations) to be performed sequentially. Using an aspect classification tree, each classification is a traversal of a path from the root to a leaf. Intermediate nodes specify features to be computed and tests to be performed, while leaf nodes represent the final aspect classification. Each path will in general contain a different sequence of tests, each of which requires a different feature to be computed.

We utilize aspect classification trees in the following way. In compile mode, the set of possible aspect classification trees are examined systematically, and the minimum cost classification tree is saved; this tree stores feature identifiers and threshold values at each node. At run time, the classification process is repeated by following the tree and comparing image features to threshold values. When a terminal node is reached, the image has been classified as an instance of a particular aspect.

Figure 2 is a simple example of a classification tree. In the figure, we have start with 4 possible classes. At the root node, all 4 classes are grouped together, and a test is indicated. The test partitions the classes into two sets. The set at the left child consists of a single class; if we reach this node, we are done after having performed only a single test. The set at the right child consists of three classes, and an additional test is needed to discriminate between these possibilities.

As shown in section 2.2, features vary in their computational costs. Hence, different paths through a tree will, in general, have different costs. Our goal is to construct classification trees in which the average cost of a classification (the average cost of a path from the root to a leaf) is minimal.

Computing the cost of a given path is straightforward. Each test requires a feature to be computed, and each such computation incurs a computational cost. Therefore, each node in the classification tree is assigned the cost of computing the feature needed for the associated test. The cost of a path from the root to a leaf is then the sum of the costs of the intermediate nodes.

For example, figure 3 illustrates the method for computing costs. Each node in the classification tree is labeled with the set of classes to be discriminated at the node and the feature to be used in the discrimination test. In addition, each node is marked with the cost of computing the feature. The darkened path through the tree has a cost of 35 units.

We define the cost of a given classification tree as the expected cost of a classification, or, equivalently, the average cost taken over all possible inputs. The expected cost can be computed by weighting the cost at each node by the proportion of the sample population that will pass through the node. The cost of every node in the tree is summed and divided by the population size to yield the expected cost of the tree.

Figure 4 illustrates the method for computing the expected cost of a classification tree. Each node is labeled with the set of classes and the associated feature. The cost of each node is the product of the feature cost and the node population. The expected cost of the

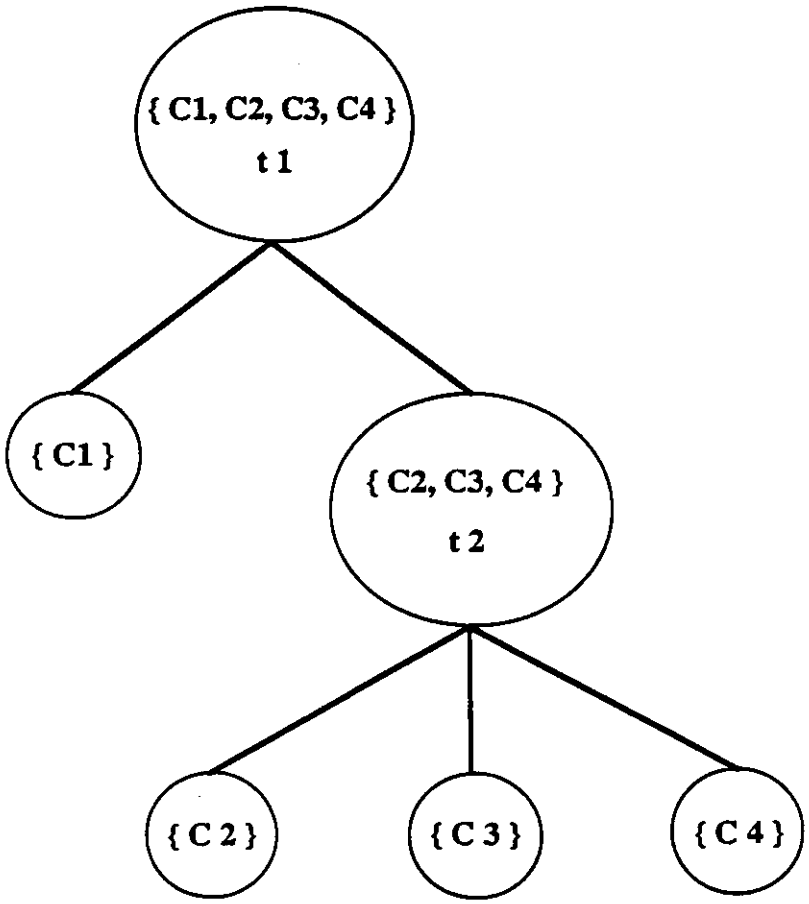


Figure 2: **Classification Tree**

In a classification tree, each node represents a collection of classes and an associated test. Arcs represent the possible results of tests.

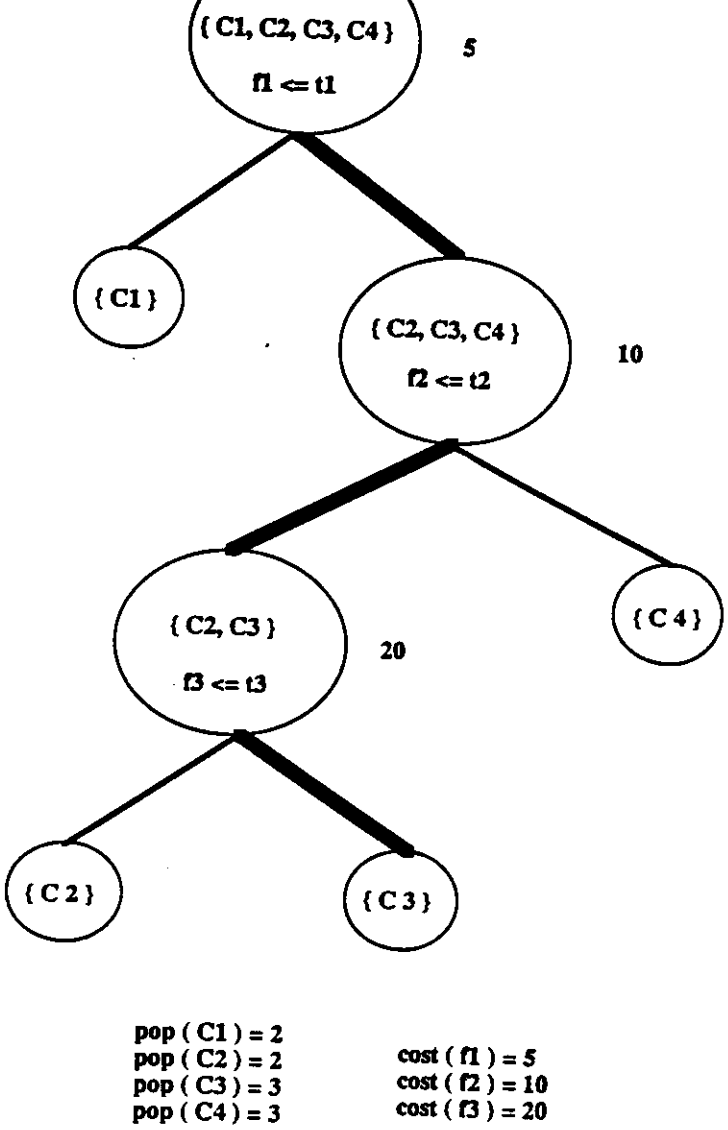


Figure 3: Cost of Classification

Each node is labeled with the classes to be discriminated and the cost of the associated test. The darkened path through the tree has a cost of 35 units.

tree is the sum of all the node costs divided by the population size. In comparing trees over the same population, the normalization factor can be deleted.

### 2.3.2 Strategy Trees and Branch-and-Bound Search

The brute force approach to finding the optimal classification tree would consist of generating all possible trees, and then selecting the one which minimizes the expected cost. However, for a complex object and many features, there are potentially a huge number of possible classification trees. Alternatively, finding a minimum cost classification tree can be formalized as a search problem over another kind of tree, which we call a *strategy tree*. A strategy tree for classification is a tree in which each path from the root to a leaf represents a complete strategy for classification (that is, each path can be expanded into a classification tree). We will formulate our strategy trees such that the search for an optimal classification strategy can be performed by searching for the leaf node of lowest cost.

In our strategy trees, nodes contain the results of applying a feature test, while arc represent feature tests. Each arc is labeled with the product of the feature cost and the expected number of samples to which the test will be applied. An arc is present when a feature can be used to break up a set of classes into smaller sets. The goal is to find a leaf node in which all the sets are singletons (containing a single class), and the cost of the path to the node is minimal. Each path from the root to a leaf represents a complete strategy for classification. The cost of the path is simply the sum of the costs of the constituent arcs.

Figure 5 illustrates a strategy tree for a simple case consisting of 4 classes, ($c1, c2, c3, c4$), and 3 features, ($f1, f2, f3$). The classes have populations of sizes $(2, 2, 3, 3)$ respectively, while the features have costs $(5, 10, 20)$ respectively. At the root, all the classes are grouped into a single set. An arc is present for every computable feature which can reduce the set size, so there are three arcs in this example. Since every sample must be tested at the root, the arc costs are the feature costs multiplied by 1; At the first child of the root, there are two sets: $\{c1\}$, and $\{c2, c3, c4\}$. The first set is a singleton, and the second set can only be subdivided by using features $f2$ or $f3$. Application of $f2$ leads to a node having three sets: $\{c1\}$, $\{c2\}$, and $\{c3, c4\}$. The cost of the arc is the cost of $f2$ multiplied by the expected number of samples tested (the sum of the population sizes of $c2$, $c3$, and $c4$) or $10 * 8 = 80$. The darkened path is the minimum cost classification strategy, and is expanded into a classification tree as the tree of figure 4.

Many different tree search algorithms exist in the literature. (See, for example, [18]) In our system, search is performed using a branch-and-bound method. The intuition behind the method is easily stated. Consider the process of exploring the strategy tree step by step. At each step, an arc is traversed and the cost of the path is increased. Rather than completely exploring one path before trying another, keep track of incomplete paths, and only explore an additional step along the path that is currently the cheapest. When a leaf node has been reached, continue exploring until all incomplete paths are at least as expensive as the minimum cost path to a leaf. The minimum cost path then represents the optimal strategy. In some cases, a node will be generated that has the same collection of sets as a previously generated node. In this case, the minimum-cost node is preserved, while the other node is pruned from the tree. The algorithm for branch-and-bound search is presented in figure 6.

There will be cases in which it is not possible, given the selection of features, to

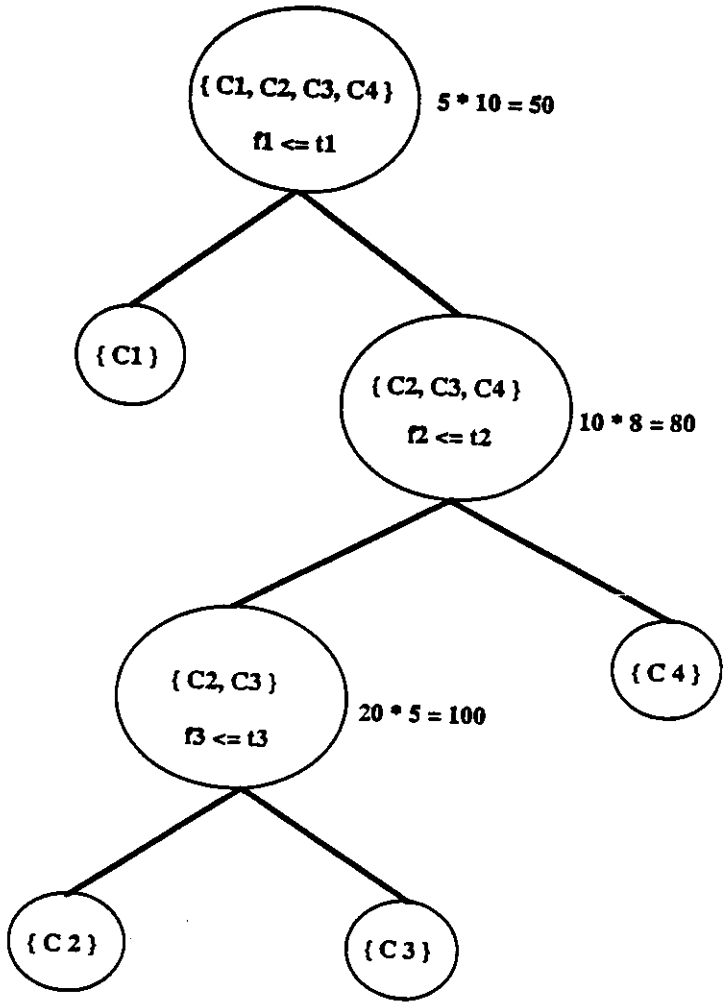


Figure 4: **Expected Cost of a Classification Tree**

The classification tree has been weighted to compute the expected cost. Each node is weighted by the cost of the feature times the population at the node.

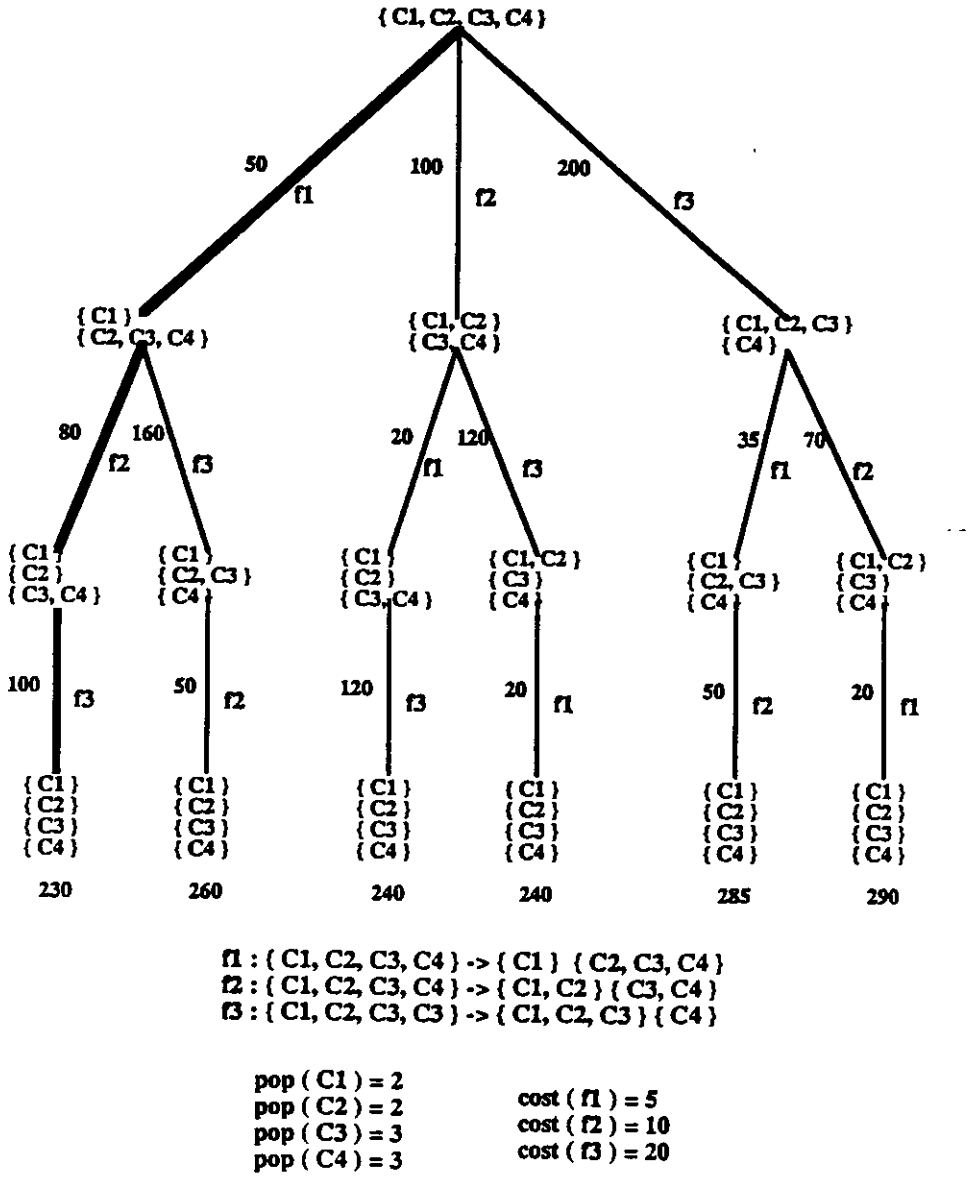


Figure 5: **Strategy Tree**

A strategy tree for a classification problem consisting of 4 classes, *C1*, *C2*, *C3*, *C4*, and 3 features, *f1*, *f2*, *f3*. *pop(Ci)* represents the population of class *Ci*, while *cost(fi)* represents the computational cost of computing feature *fi*. The darkened path is the minimum cost path, and is expanded into the classification tree of figure 4.

```
  Place the root node on OPEN.
* If ( empty(OPEN) ) then
     Failure
  else {
     Select a minimum cost node, n, from OPEN
     Place n on CLOSED
     If ( leaf(n) ) then
        Exit, with the path from root to n as the solution
     Else {
        Generate all children, m, of n
        For each child, m, of n {
              Compute the cost of m
           If ( not( member( m, OPEN ))
             or not( member( m, CLOSED ))) then
              Place m on OPEN, with a pointer back to n
           Else if ( member( m, OPEN )) then
     Redirect pointers along path of least cost
           Else if ( member( m, CLOSED )) then {
              Redirect pointers along path of least cost
              If ( pointers change ) then
                 Place m on OPEN
           }
        }
     }
  }
  Go to *
```

Figure 6: **Branch-and-bound Search Algorithm**

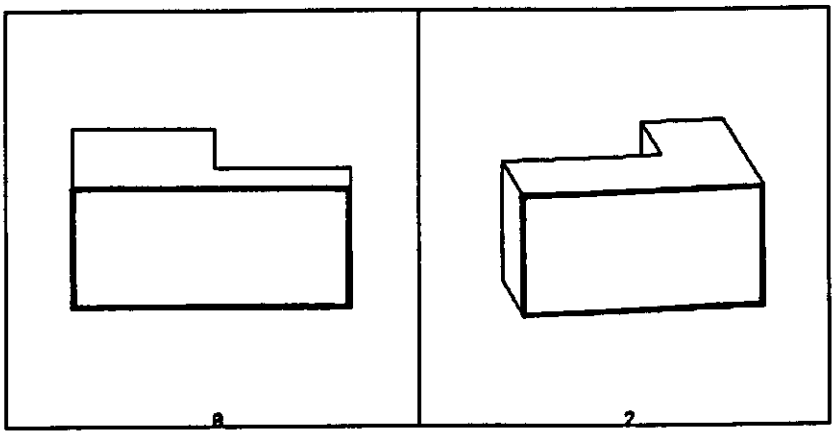

Figure 7: **Parallel Classes**

The aspects illustrated here cannot be distinguished using the available features. Objects are depicted as wire frame drawings. Dark lines distinguish detectable faces.

distinguish two classes. We refer to the corresponding classes as *parallel classes*, and the corresponding nodes in the classification tree as *parallel nodes*. For example, figure 7 shows the representative images from two classes that cannot be distinguished using available geometric features. This is not a failure of the search strategy, but represents a fundamental limitation of the features available. In our method for object localization, the configuration determination step corrects for ambiguous aspect classification and determines the correct object configuration. For purposes of search, parallel nodes are considered leaf nodes, and are simply assigned a large value for cost.

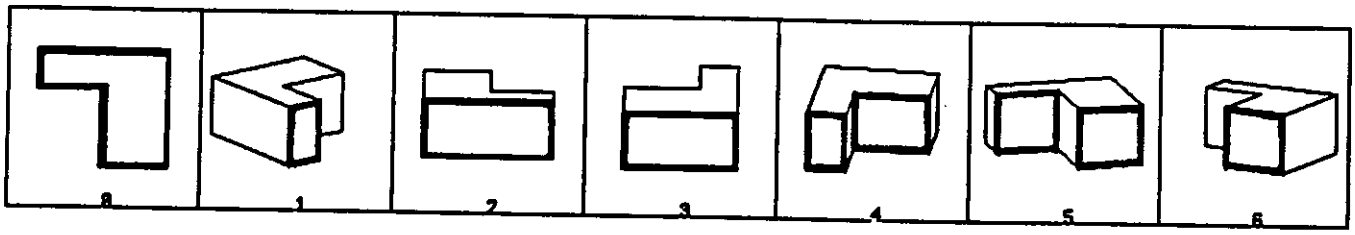


Figure 8: **Polyhedron Aspects**

# 3 Object Demonstrations

We have implemented a module of a VACL that generates minimum cost classification trees for aspect classification. The module uses a CAD object model, a model of a range sensor, and the cost and discrimination information for each of the features listed in section 2.2 to determine the optimal classification strategy for a given object. The range sensor modeled is incapable of detecting surfaces that are sloped more than 45 degrees from the line-of-sight.

In each of the demonstrations discussed below, the aspects were determined systematically (see [11]) and provided as input to the system. The aspects were not determined over the entire Gaussian sphere, but only over one hemisphere. The aspects are illustrated, along with the resulting classification tree. In each of the illustrations, the objects are depicted with wire frame drawings in which the boundaries of faces detectable by the sensor are darkened. Each classification tree is labeled with the expected cost relative to a 128 by 128 image. Each decision node in a tree contains the number of the feature on which a test is performed. Parallel nodes are labeled **N-PAR**.

## 3.1 Polyhedron Demonstration

In the first demonstration, a simple polyhedron was used. The aspects selected for the polyhedron are illustrated in figure 8. It can be immediately seen that aspects 2 and 3 will be impossible to ambiguate, since only a single face is visible in each aspect, and the faces are the same size and shape.

The corresponding classification tree is presented in figure 9. Note the presence of a parallel node; as predicted, aspects 2 and 3 are not distinguishable.

The polyhedron classification tree is used as follows. At the root node, RCH, the area of the largest brother region is computed and used to separate aspects 4 and 5 from each other, as well as from the aspects that have only a single region. At the next decision node, BH0, the dominant eigenvalue of the largest region (the only region, in this case) is tested; this is sufficient to distinguish aspect 1 from the rest. At node BH3, the area of the largest region is used to distinguish aspects 0 and 6 from the rest. Node BH7 is a parallel node combining aspects 2 and 3.

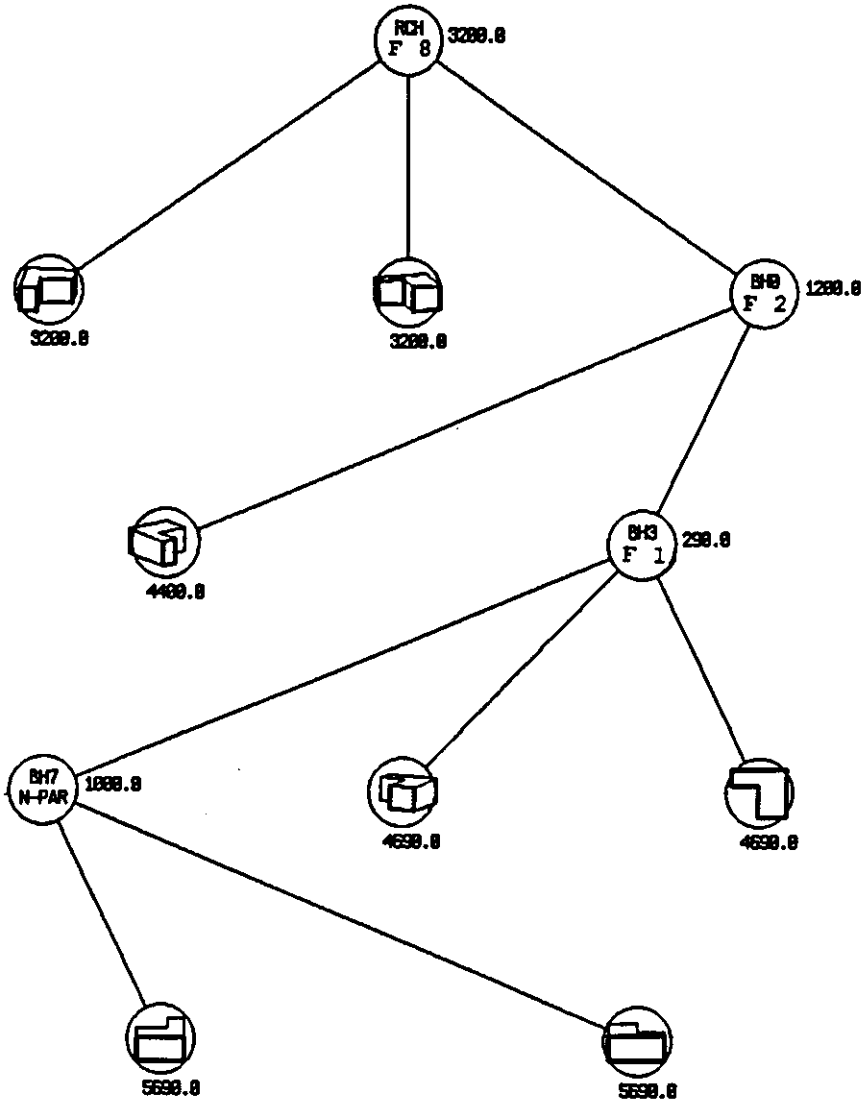


Figure 9: **Polyhedron Aspect Classification Tree**

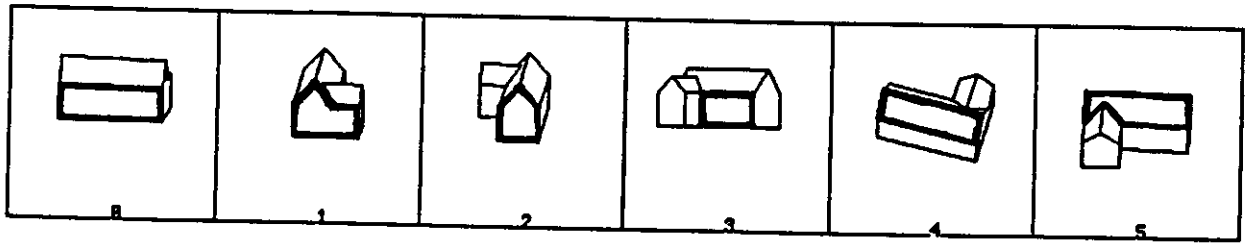


Figure 10: **House Aspects**

## 3.2 House Demonstration

In this demonstration, a simple model of a house was used. The aspects selected for the house are illustrated in figure 10.

The corresponding classification tree is presented in figure 11. There are two parallel nodes visible in this tree. The first parallel node, BH3, combines aspects 4 and 5, while node BH0 combines aspects 4 and 0. The interpretation here is that the area of the largest region (feature F1, used at the root node) is capable of distinguishing aspects 0 and 5, but the area of the largest region in aspect 4 lies somewhere between the values of the other two aspects, and the sensor is not sufficiently accurate to make the distinction. For example, the sensor might yield areas that are accurate to $\pm 10$ units, while the areas of the largest faces might be 100, 110, and 120 units.

## 3.3 Airplane Demonstration

In this demonstration, a much more complicated model was used. The model is that of a stylized airplane. The aspects selected for the airplane are illustrated in figure 12. The corresponding classification tree is presented in figure 13.

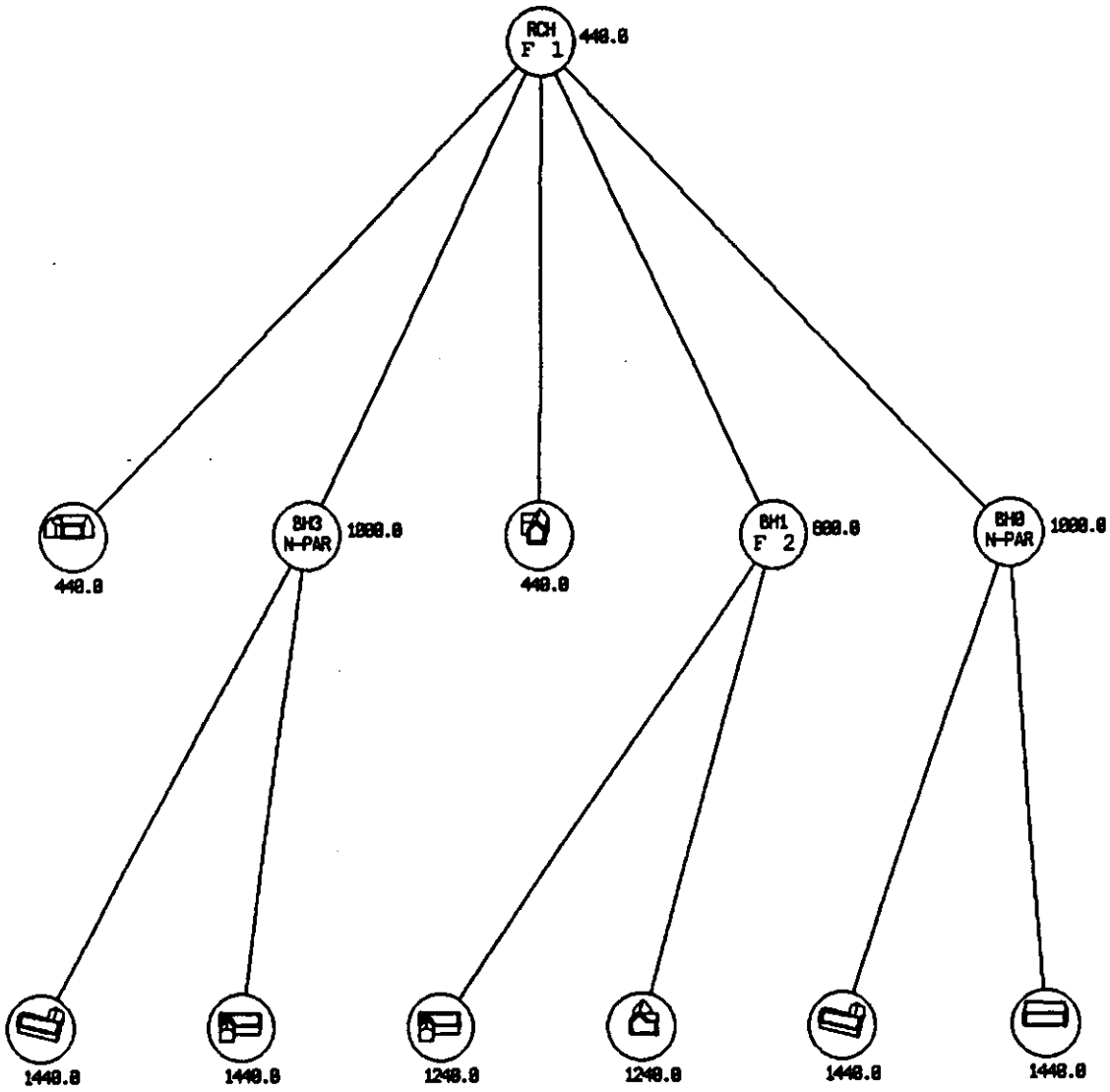


Figure 11: **House Aspect Classification Tree**

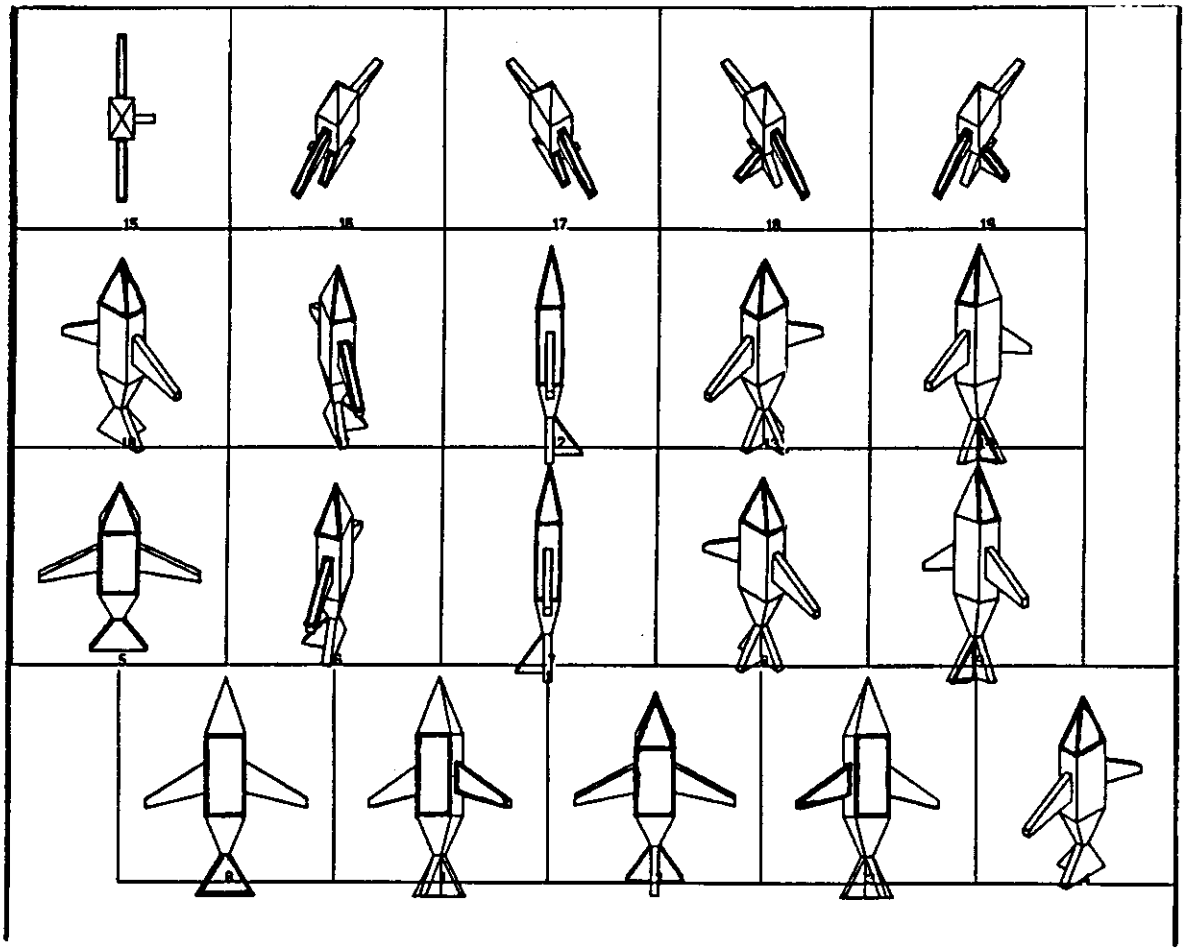

Figure 12: **Airplane Aspects**

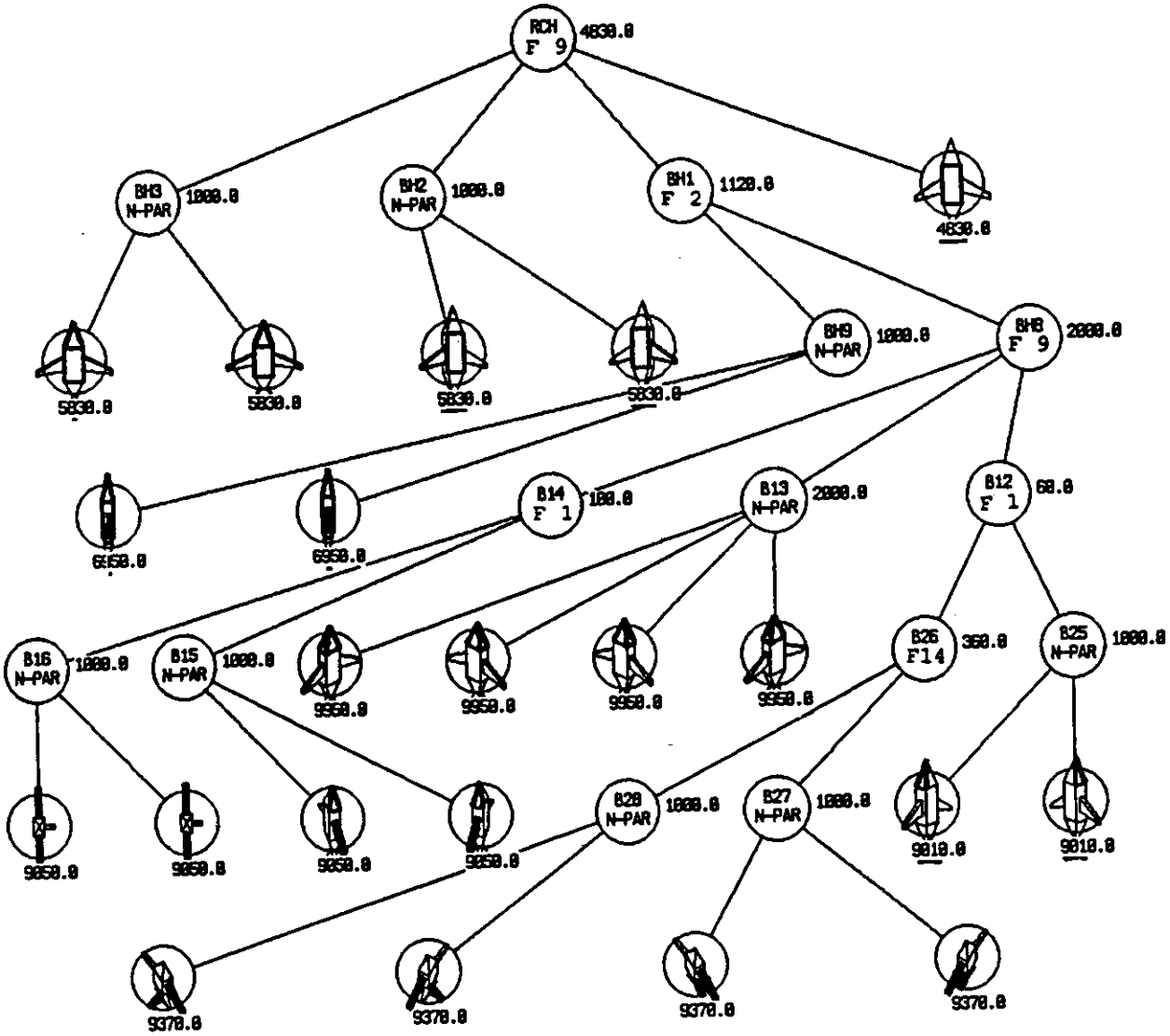


Figure 13: **Airplane Aspect Classification Tree**

# 4 Summary

Generating programs for model-based vision applications has traditionally involved hand-coding of the relevant object, sensor, and task knowledge. The process is therefore expensive due to the requirement for highly skilled individuals to extract and code the knowledge. An alternative to hand-coding is the use of a vision algorithm compiler (VAC), which uses stored object, sensor, and processing models to automatically construct programs.

The expense of development is only one of the cost considerations that must be met by a vision application program. It is also important for an application to execute quickly and efficiently. Therefore, a practical VAC must also be an optimizing compiler, and select the set of operations that can perform the given task for the minimum computational cost.

In this paper, we addressed part of the problem of constructing an optimizing compiler for vision tasks. In particular, we discussed a method used to optimize the process of aspect classification, which is one step in the process of object localization; we defined optimization to be minimizing the expected cost of classifying an image into an aspect.

We represented a strategy for aspect classification as a classification tree, in which nodes represent classes and tests, and arcs denote the results of the tests. Classification is performed by starting at the root of the tree, and proceeding from the root to a leaf, extracting the features and performing the tests indicated at each node. The cost at each node is therefore the cost of the operations required to extract the feature specified at the node. The cost of a classification is the sum of the costs of the nodes along the path. We determined the computational cost of the feature extraction operations for a specific collection of features.

The space of possible classification strategies was represented in the form of a strategy tree, in which each path from the root to a leaf corresponds to a complete classification strategy (and hence to a classification tree). Hence, the optimal classification strategy can be obtained using standard tree search procedures. We performed a variation of branch-and-bound search over the strategy tree to find the minimum cost path from the root to any leaf, and converted this path to a classification tree.

We demonstrated the feasibility of our approach by implementing a module for generating minimum cost aspect classification trees within an existing VACL (Vision Algorithm Compiler for object Localization). The performance of the module was illustrated on several example objects.

# 5 Acknowledgements

Takeo Kanade provided many useful comments and encouragements. The authors also thank the members of the Intelligent Modeling Laboratory of the Robotics Institute of Carnegie Mellon University for their valuable comments and discussions.

# References


[1] Ballard, D. H. *Generalizing the Hough transform to detect arbitrary shapes.* **Pattern Recognition**, vol. 13 (1981), pp. 111–122.

[2] Besl, P. *Geometric modeling and computer vision.* **Proc. of the IEEE**, vol. 76 (1988), pp. 936–958.

[3] Bolles, R. and Cain, R. A. *Recognizing and locating partially visible objects: the local-feature-focus method.* **The International Journal on Robotics Research**, vol. 1 (1982), pp. 57–82.

[4] Bolles, R. C. and Horaud, P. *3DPO: A three-dimensional part orientation system.* in: **Three-Dimensional Machine Vision**, edited by T. Kanade. Kluwer, Boston, MA, 1987, pp. 399–450.

[5] Bolles, R. and Horaud, P. *3DPO: A three-dimensional part orientation system.* **The International Journal of Robotics Research**, vol. 5 (1986), pp. 3–26.

[6] Brooks, R. *Symbolic reasoning among 3-D models and 2-D images.* **Artificial Intelligence**, vol. 17 (1981), pp. 285–348.

[7] doCarmo M. P. **Differential Geometry of Curves and Surfaces.** Prentice-Hall, Inc., Englewood Cliffs, NJ, 1976.

[8] Faugeras, O. D. and Hebert, M. *The representation, recognition, and locating of 3-D objects.* **The International Journal of Robotics Research**, vol. 5 (1986), pp. 27–52.

[9] Goad, C. *Special purpose automatic programming for 3D model-based vision.* in: **Proc. of DARPA Image Understanding Workshop**, DARPA. 1983, pp. 94–104.

[10] Grimson, W. E. L. and Lozano-Perez, T. *Model-based recognition and localization from sparse range or tactile data.* **The International Journal of Robotics Research**, vol. 3 (1984).

[11] Ikeuchi, K. *Generating an Interpretation Tree from a CAD Model for 3-D Object Recognition in Bin-Picking Tasks.* **International Journal of Computer Vision**, vol. 1 (1987), pp. 145–165.

[12] Ikeuchi, K. and Hong, K. S. *Determining Linear Shape Change: Toward Automatic Generation of Object Recognition Program.* in: **Proc. of IEEE Conf. on Computer Vision and Pattern Recognition.** San Diego, 1989. *a longer version, containing programs, is avaiable as CMU-CS-88-188.*

[13] Ikeuchi, K. and Kanade, T. *Modeling Sensors: Toward automatic generation of object recognition program.* **Computer Vision, Graphics, and Image Processing**, 1989. *(Accepted for publication).*

[14] Ikeuchi, K., Nishihara, H. K., Horn, B. K. P., Sobalvarro, P., and Nagata, S. *Determining grasp points using photometric stereo and the PRISM binocular stereo system.* **The International Journal of Robotics Research**, vol. 5 (1986), pp. 46–65.

[15] Kanade, T., Balakumar, P., Robert, J., Hoffman, R., and Ikeuchi, K. *Overview of geometric/sensor modeler VANTAGE.* in: **Proc. the International Symposium and Exposition on Robots**. The Australian Robot Association, Sydney, Australia, 1988.

[16] Koenderink, J. J. and Van Doorn, A. J. *Internal representation of solid shape with respect to vision.* **Biological Cybernetics**, vol. 32 (1979), pp. 211–216.

[17] Oshima, M. and Shirai, Y. *An Object Recognition System Using Three-Dimensional Information.* in: **Three-Dimensional Machine Vision**, edited by T. Kanade. Kluwer Academic Publishers, 1987, pp. 355–397.

[18] Pearl, J. **Heuristics: Intelligent Search Strategies for Computer Problem Solving.** Addison-Wesley, 1984.

[19] Perkins, W. A. *Model-based vision system for scene containing multiple parts.* in: **Proc. 5th International Joint Conference on Artificial Intelligence.** 1977, pp. 678–684.

[20] Roberts, L. G. *Machine perception of three-dimensional solids.* in: **Optical and Electro-Optical Information Processing**, edited by J. Tipplett. MIT Press, Cambridge, MA, 1965, pp. 159–197.

[21] Yoda, H., Motoike, J., and Ejiri, M. *Direction Coding Method and its Application to Scene Analysis.* in: **Proc. 4th Joint Conference on Artificial Intelligence.** International Joint Conf. on Artificial Intelligence, 1975.

# A Computational Costs of Features

This appendix calculates the computational costs of features for aspect classification. In this appendix, for simplicity's sake, we calculate the computational cost of 2D features derived from the image rather than 3D features derived from the 3D face. Namely, we ignore the operational cost of the affine transformation which is applied to each pixel value for converting 2D to 3D.

We will assume that the basic vision modules provide the following arrays.

- the *needle map* which contains gradient space $(p, q)$ values at each pixel, and
- the *label map* which indicates the region to which each pixel belongs.

We will also use the following notations for computational costs.

$A$= the rectangular search area size of the target region

$B$= the rectangular search area size of the brother region

$a$= the area size of the target region

$b$= the area size of the brother region

$X$= unit computational cost of test

$Y$= unit computational cost of one add/subtract operation

$Z$= unit computational cost of one multiply/divide operation

$R$= unit computational cost of one array reference operation

$S$= unit computational cost of one square root operation

$T$= unit computational cost of one trigonometric operation

In what follows, the largest visible region is referred to as the target region, while surrounding regions are brother regions. When the singular term *brother region* is used, it refers to the largest of the brothers.

**F1:** *the area of the target region*

In order to calculate the area of the target region, we have to scan pixels over the possible existence area of the target region and examine whether or not a pixel belongs to the target region.

This operation requires:

- $AR$ - array access to the label map
- $AX$ - test as to whether the region label is the target region label

Once a pixel is determined to belong to the target region, we will add one to an accumulator for the area size. This operation occurs $a$ times. Thus, .

- $aY$ - add one to accumulator

As the result we obtain $AR + AX + aY$.

**F2:** *the dominant eigenvalue of the target region*

Similar to the case of region area, $AR+AX$ units are needed to find the pixels belonging to the target region.

The inertia matrix can be obtained using these formulae:

$$\begin{aligned} a &= \sum \\ I_{ii} &= \sum i^2 - \tfrac{1}{a}(\sum i)^2 \\ I_{ij} &= \sum ij - \tfrac{1}{a}(\sum i)(\sum j) \\ I_{jj} &= \sum j^2 - \tfrac{1}{a}(\sum j)^2 \end{aligned}$$

These requires the following operations over $a$ pixels of the target region.

$$\begin{aligned} \sum i^2 &-- a(Y+Z) \\ \sum j^2 &-- a(Y+Z) \\ \sum ij &-- a(Y+Z) \\ \sum i &-- aY \\ \sum j &-- aY \\ \sum &-- aY \end{aligned}$$

As the result we obtain $AR + AX + 6aY + 3aZ$.

**F3:** *the eigenvalue ratio of the target region*

This feature requires the same operations as those of F2. $AR + AX + 6aY + 3aZ$.

**F4:** *the number of brother regions*

In order to find the surrounding regions, we have to search for pixels over the possible existence area of the surrounding regions, $(BR)$, and examine whether or not these pixels belong to the new region, $(BX)$. $BR + BX$.

**F5:** *the distance between the largest brother region and the target region*

This distance is the distance between the centroids of the regions. The scanning and examining operations for the target and brother region cost $(BR + BX)$.

We can obtain the centroid of a region with area n as

$$\begin{aligned} n &= \sum &-- nY \\ i_0 &= \tfrac{1}{n}\sum i &-- nY \\ j_0 &= \tfrac{1}{n}\sum j &-- nY \end{aligned}$$

The cost to find the centroid of the target region is $3aY$; the cost for the brother region is $3bY$.

Thus, a total cost of $BX + BR + (3a + 3b)Y$ is necessary for this feature.

**F6:** *the angle between the principal axes of the largest brother region and the target region*

The scanning and examining operations for the target and brother regions cost $BR+BX$.

At each region, we will calculate the inertia moment direction. This yields $6aY + 3aZ$ for the target region and $6bY + 3bZ$ for the brother region.

The total cost is $BR + BX + (6a + 6b)Y + (3a + 3b)Z$ for this feature.

**F7:** *the surface orientation difference between the largest brother region and the target region*

The scanning and examining operations for the target and brother region cost $BR+BX$.

The average surface orientation of a region can be obtained as:

$$\begin{array}{rcccc} n & = & \sum & -- & nY \\ p_0 & = & \frac{1}{n}\sum p_i & -- & n(R+Y) \\ q_0 & = & \frac{1}{n}\sum q_i & -- & n(R+Y) \end{array}$$

Note that two array reference operations $p_i, q_i$ are required for each summing operation. $2aR + 3aY$ and $2bR + 3bY$ operations are necessary for the target and brother region, respectively.

$(B + 2a + 2b)R + BX + (3a + 3b)Y$ is the total cost for this feature.

**F8:** *the area of the largest brother region*

The scanning and examining operations for the brother region cost $BR + BX$.

The area of the brother region can be obtained using the same method as the area of the target region (F1), and thus it requires $bY$. This feature therefore requires $BR + BX + bY$.

**F9:** *the dominant eigenvalue of the largest brother region*

The inertia matrix of the brother region can be obtained using the same method as the inertia matrix of the target region (F2). Thus, $BR + BX + 6bY + 3bZ$ is the cost for this feature.

**F10:** *the eigenvalue ratio of the largest brother region*

Again, this can be obtained using the same method as the inertia ratio of the target region (F3). Thus, $BR + BX + 6bY + 3bZ$ is the cost.

**F11:** *the surface characteristics of the target region*

The surface characteristics of a region are the average Gaussian and mean curvatures. Therefore, we must compute the Gaussian and mean curvature at each pixel and average over the entire region.

The scanning and examining operations for the target region cost $AR + AX$.

The average Gaussian and mean curvature can be obtained from the needle map via the first and the second fundamental form of the surface [7]. Let us denote $E, F, G$ as

the first fundamental form, $e, f, g$ as the second fundamental form, and $K, H$ as the Gaussian and mean curvature. The following operations are executed at each pixel which belongs to the target region.

We will calculate the first fundamental forms.

$$\begin{array}{rcll} E & = & 1+p(i,j)^2 & -- \quad Y+Z \\ F & = & p(i,j)q(i,j) & -- \quad Z \\ G & = & 1+q(i,j)^2 & -- \quad Y+Z \end{array}$$

This requires $2Y + 3Z + 2R$, where we assume that we refer $p(i,j), q(i,j)$ once, store them at somewhere, and use that stored value in the later operations (2R).

We will also calculate the second fundamental forms.

$$\begin{array}{rcll} e & = & \frac{p_x(i,j)}{\sqrt{1+p(i,j)^2+q(i,j)^2}} \quad \frac{p(i+1,j)-p(i,j)}{\sqrt{1+p(i,j)^2+q(i,j)^2}} & -- \quad 3Y+3Z+S \\ f & = & \frac{p_y(i,j)}{\sqrt{1+p(i,j)^2+q(i,j)^2}} \quad \frac{p(i,j+1)-p(i,j)}{\sqrt{1+p(i,j)^2+q(i,j)^2}} & -- \quad 3Y+3Z+S \\ g & = & \frac{q_y(i,j)}{\sqrt{1+p(i,j)^2+q(i,j)^2}} \quad \frac{q(i,j+1)-q(i,j)}{\sqrt{1+p(i,j)^2+q(i,j)^2}} & -- \quad 3Y+3Z+S \end{array}$$

Since we have to refer $p(i-1,j), p(i,j-1), q(i,j-1)$, we need extra $3R$, and thus, $3R + 9Y + 9Z + 3S$.

Using the first and the second fundamental forms, we can obtain the Gaussian and mean curvature.

$$\begin{array}{rcll} K & = & \frac{eg-f^2}{EG-F^2} & -- \quad 2Y+5Z \\ H & = & \frac{1}{2}\frac{eG-2fF+gE}{EG-F^2} & -- \quad 3Y+8Z \end{array}$$

Thus, $5R + 16Y + 25Z + 3S$ is the computational cost for calculating the Gaussian and mean curvature at each pixel.

The average Gaussian and mean curvature for a region of $n$ pixels are obtained by:

$$\begin{array}{rcll} n & = & \sum & -- \quad nY \\ \bar{K} & = & \sum K & -- \quad nY \\ \bar{H} & = & \sum H & -- \quad nY \end{array}$$

Thus, $(A + 5a)R + AX + 19aY + 25aZ + 3aS$ is the computational cost of this feature.

**F12:** *the surface characteristics of the largest brother region*

The same operations as those of the target region are applied (F11) and incur a computational cost of $(B + 5b)R + BX + 19bY + 25bZ + 3bS$.

**F13:** *the surface characteristic distribution of the target region*

For this feature, we classify the surface type at every pixel in the target region, and compare the surface types of the image and model target regions.

The surface characteristic at a pixel is one of the following: hyperbolic, positive elliptic, negative elliptic, cylindrical, and planar. These features are obtained as the combination of the sign of the Gaussian and mean curvature at each pixel. For each

aspect region, a prototypical distribution, calculated at its representative attitude, is stored. The observed distribution will be rotated and translated so that it is nearly aligned with the model distribution. Then, the observed and model distributions are compared.

*generating the surface characteristics distribution*

Scanning and examining the target region requires $AR + AX$ total computational cost.

Calculating the Gaussian and mean curvature at each pixel for the target region requires $5R + 16Y + 25Z + 3S$ computational cost per pixel.

Calculating the mass center and the moment direction for the alignment requires $6Y + 3Z$ per pixel.

The assignment of the surface description can be determined using:

```
(if   ((> K 0.0)  (if    ((>= H 0.0) positive elliptic)
                         (t          negative elliptic)
      ((= K 0.0)  (if    ((=  H 0.0) planar)
                         (t          cylindrical))
      (t                             hypabolic))
```

and requires $2X$ per pixel.

Finally, storing this result on an array requires an extra $R$ per pixel. Taking all the costs over a region af area $a$ yields $(A + 2a)X + 22aY + 28aZ + 3aS + (A + 6a)R$ to obtain the surface characteristics distribution.

*Comparison*

Scanning and examining the target region requires computational cost of $AR + AX$.

The translation and rotation can be represented as:

$$\begin{aligned} X &= i\cos\theta - j\sin\theta - i_m \quad -- \quad 2Y + 2Z + 2T \\ Y &= i\sin\theta + j\cos\theta - j_m \quad -- \quad 2Y + 2Z + 2T \end{aligned}$$

The entire region is translated and rotated, so $a$ such operations are necessary.

Comparing the model description with the observed one requires two array reference operations (one each for the model array and the observed array) and one comparison operation, or $2R + X$ per pixel. If they are the same, the confidence measure will be increased, which requires $Y$. The resulting confidence measure will be divided by the area size. Thus, the area size is necessary, and it requires $Y$. As the result, this second part needs $(A + a)X + 6aY + 4aZ + 4aT + (A + 2a)R$.

The total required computational cost for this feature can be obtained to add the cost of the first part to those of the second part and is $(2A + 3a)X + 28aY + 32aZ + 3aS + 4aT + (2A + 8a)R$.

**F14:** *the surface characteristic distribution of the largest brother region*

The same operations as for the target region are applicable (F13), and the cost is $(2B + 3b)X + 28bY + 32bZ + 3bS + 4bT + (2B + 9b)R$.