AUTOMATIC INTERPRETATION OF FLOOR PLANS USING SPATIAL INDEXING*

HANAN SAMET
AYA SOFFER

Computer Science Department and
Center for Automation Research and
Institute for Advanced Computer Science
University of Maryland at College Park
College Park, Maryland 20742
E-mail: hjs@umiacs.umd.edu, aya@umiacs.umd.edu

ABSTRACT

A system is presented that classifies objects in raw images using statistical pattern recognition and spatial indexing. The system is given a training set containing samples of feature vectors of objects that may be found in the images. This method of classification was applied to automatic interpretation of floor plans. A number of data structures are suggested for storing the training set. Of these, the adaptive k-d tree was found to be a highly efficient storage mechanism, and provided recognition rates of 98% and higher. Storing the training set as feature ranges per class also gave very high recognition rates, at the cost of a relatively small number of comparisons.

1 Introduction

The successful use of image databases requires a means of indexing based on some of the objects found in the images. In the vast majority of systems it is up to the user to identify these objects to the system. It is clear that this mechanism must be automated. In many applications, the set of objects that may appear in the image are known a priori. Furthermore, the geometric shapes of these objects are relatively primitive thereby simplifying the task of object recognition in these images (e.g. OCR). Pattern recognition techniques are frequently used to identify the objects. They assign a physical object or an event to one of several prespecified classes. Patterns are recognized based on some features or measurements made on the pattern. In the statistical approach (reviewed in section 2), a pattern is represented in terms of n features or properties and viewed as a point in n-dimensional space. The features are selected in such a way that pattern vectors belonging to different classes will occupy different regions of this feature space. Sample patterns from each class are given (termed a *training set*) and stored in a sample library. This library is then used to classify an unknown pattern as a member of one of the prespecified classes. The nearest neighbor metric is the method most widely used for this classification.

The larger the training set, and the more features used to discriminate among the shapes, the better is the achievable recognition rate. However, such an improvement in the recognition rate will result in a degradation of the execution time. It is thus important to devise a method to prune the search space, and to have the classifier compare a minimal number of stored feature vectors to the one being classified. This paper presents such a method for storing and searching the library. The method is based on spatial indexing of the feature vectors. Using this method a relatively small number of comparisons will be made, while achieving an excellent recognition rate. We have applied this method to the automatic interpretation of floor plans. The results of our experiments are described in Section 4. Other methods that have been used to reduce the computation time include pruning the training set by eliminating patterns that do not help in the classification[1].

*The support of the National Science Foundation under Grant IRI-9017393, and the National Aeronautics and Space Administration under Grant NGT-30130 is gratefully acknowledged.

Fukunaga and Narendra[3] describe a branch and bound algorithm for computing the k-nearest neighbors. Further algorithmic improvement of this method is presented by Kamgar-Parsi and Kanal ?. Miclet and Dabouz[7] propose an algorithm to find an "approximate" nearest neighbor.

2 A Statistical Pattern Recognition System

There are two main phases in a statistical pattern recognition system. In the first phase, termed the *training phase*, the system is initialized with sample feature vectors for each class, termed the *training set*. These samples are stored in a library. In the second phase, termed the *classification phase*, the sample library is used to classify unknown patterns detected in an input image. Each phase is divided into four stages with the first three stages common to both phases.

Stage I consists of image acquisition. Stage II is a preprocessing stage where various image processing techniques are applied to enhance the image. The image is then segmented into its constituent elements. This may be achieved, for example, by a connected component labeling algorithm. Each connected component is considered a pattern, and is passed to stage III. Stage III performs feature extraction. The features that have been identified as those that will best discriminate between the classes are computed for each pattern detected in stage II. The feature vector pattern representation is composed and passed onto the next stage.

In stage IV of the training phase, a classification is attached to each pattern associating it with one of the possible classes, or assigning it to the undefined class. These sample feature vectors along with their classifications are built into the training set library. In stage IV of the classification phase, the class of the library pattern that best matches that of the unknown pattern is assigned to it. Using the nearest neighbor approach, the distance between the stored vectors and that of the input vector serves as the similarity measure. If no sample pattern is found within a prespecified maximum distance, then the pattern is classified as undefined. The most well-known similarity measure is the weighted Euclidean distance. The features that are known to be more reliable than others are given a greater weight when computing this similarity measure. It is given by

$$D_E = \sqrt{\sum_{i=1}^{N} w_i (F_i^L - F_i^I)^2}$$

where F_i^L is the i^{th} feature of the library vector, F_i^I is the i^{th} feature of the input vector, and w_i is the i^{th} weighting factor.

3 Building and Searching the Training Set Library

A crucial issue in the design of a statistical pattern recognition system is how to store and search the library of elements of the training set efficiently. To ensure a reasonable recognition rate, the system must be able to store a large number of elements.

Below we present three different storage and search strategies. The first is a simple list with no ordering of the feature vectors. It is searched using an exhaustive search. This is the method used by most classifiers both in research and in industry. It is presented here mainly for the sake of comparison with the other methods. The second is an adaptive k-d tree[2]. An algorithm that efficiently finds the nearest neighbor to a given point in such a tree is used to find the closest match. The third groups the elements of the training set into their respective classes, and stores a range of the possible values for each feature in the feature vector . An exhaustive search is used for the class range list, but there are much fewer elements in this list, so it is not as costly as the first method.

3.1 Library Stored as an Unordered List

In the training phase, the list of sample feature vectors along with their corresponding classifications can be represented as an unordered list, where the elements appear in the same order as they are

found in the input. In the classification phase, this list is searched exhaustively. That is, for each unknown pattern, the Euclidean distance between it and every object in the library is computed. The class associated with the object with the smallest distance is assigned to the unknown region.

This method guarantees the best possible recognition rate for a given training set and selected features. It has the worst execution time of the three methods presented here. Its execution time is always (i.e. the best, worst, and average) $O(J \cdot I)$, where J is the number of samples in the library, and I is the number of features in the feature vector.

3.2 Library Stored as points in an Adaptive k-d Tree

Realizing that a feature vector is a point in n-dimensional space, we can use methods borrowed from computational geometry and spatial data structures to spatially sort the training set. The data structure that we use is the adaptive k-d tree[2]. This is a variant of a binary search tree with two pieces of information stored at each node: a dimension number d indicating the discriminating dimension, and a discriminating value v which is usually the median value of dimension d of the set of points stored below this node. The discriminating dimension is chosen to be the dimension for which the range of values of is a maximum. The range is measured as the distance from the minimum to the maximum values normalized with respect to the median of the observed values. All observed points with values less than or equal to the discriminating value are inserted in the left subtree, and all points with values greater than the discriminating value are inserted in the right subtree. This process is continued recursively until only a few points are left in the set, at which time they are stored in what is termed a *bucket* (usually represented by a linked list). Figure 1 shows an example of a set of points in 2-space with the corresponding adaptive 2-d tree. The list of feature vectors of the training set is stored in the adaptive k-d tree using the algorithm described above. The leaf nodes will contain the feature vector values along with the associated classification.

In order to determine the leaf node to which a point belongs, the tree only needs to be descended. At each node, the value of the point's d^{th} component is compared to that of the discriminator v stored in that node. If it is less than or equal to v, then the procedure is recursively applied to the left child of the node. If it is greater than v, then the procedure is recursively applied to the right child of the node. The procedure stops when a leaf node is reached. The number of comparisons involved is equal to the depth of the tree, which is $\log_2 \frac{N}{L}$, where N is the number of points in the tree, and L is the maximum number of points stored in each leaf node.

In the classification phase, we use an algorithm based on this search strategy to find the nearest neighbor of a given feature vector. The simplest approach is to find the leaf node that would contain this point, say q. The nearest neighbor is then chosen from the points stored in this node. The computation time of this method is $O(\log_2 \frac{J}{L} + L \cdot I)$, where J is the number of samples in the library, L is the number of samples in each leaf node, and I is the number of features in the feature vector (hence we have an adaptive k-d tree over a I-dimensional space). However, following this procedure may not yield the nearest neighbor. For example in Figure 1, using this algorithm, the nearest neighbor of point (60,25) would be the object at (85,15), thus it would be classified as a range. Yet, clearly point (50,10) is closer to it, and it should be classified as a window.

In order to improve the recognition rate, we applied a more sophisticated algorithm. The algorithm **?** is based on the observation that once the nearest neighbor in the leaf node that was reached by descending the tree is found, the only other nodes that need to be searched for a possibly closer neighbor, are the nodes that correspond to neighboring boxes that are within the radius of the nearest point found so far. For example in Figure 1, when searching for the point (60,25), only the buckets adjacent to the one containing point (85,15) (i.e. the buckets containing (50,10), (50,45), and (25,35)) need to be searched. The number of additional leaf nodes that need to be searched using this algorithm is relatively small because the adaptive k-d tree acts as a sort on the space.

In order to implement this algorithm, a small addition to the search algorithm outlined above is

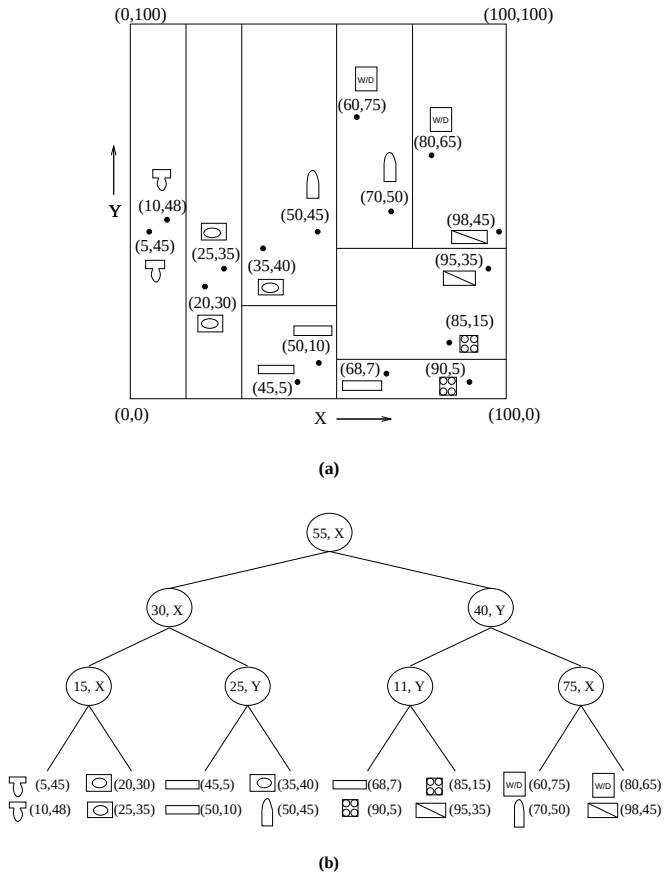

Figure 1: Adaptive k-d tree: (a) set of points in 2-space, (b) 2-d tree.

needed. When backing up during the tree traversal, the distance to the hyperrectangle represented by the nodes' brother is computed. If this distance is smaller than the minimum distance found so far, then recursively apply the search procedure to this node. Note, that since a node and its brother only differ by one coordinate, we only need to compute the difference between that one coordinate of the discriminating point and the search point in order to compute the distance between the search point and the hyperrectangle represented by the nodes' brother. Thus, backtracking only costs one operation that is independent of the dimension of space represented by the k-d tree.

The performance of this algorithm can be improved by noticing that many times the hyperrectangle represented by a node is much larger than the hyperrectangle defined by the points that are actually stored in the node. By storing the maximum and minimum of each coordinate among the points stored in the hyperrectangle, we can significantly improve the pruning of the search space. The only change in the algorithm is that when backing up, during the tree traversal, the distance to the hyperrectangle represented by the actual points stored in the nodes' brother is computed, rather than the distance to the hyperrectangle represented by the node itself. As a result of this change, the number of additional nodes that are searched is reduced significantly. Note however, that in this case we do need a full distance computation, i.e the cost of each computation is proportional to the dimension of space represented by the k-d tree. In addition, since we need to store the maximum and minimum of each coordinate among the points stored in the hyperrectangle, the space required to store the k-d tree increases significantly.

Observing these deficiencies in this version of the algorithm, a third variation is suggested. In this case we only store the minimum and maximum value of the discriminating coordinate, and

again compute the distance only using this coordinate. The decrease in the number of additional nodes searched is not as great as in the case of the full computation, but the additional storage requirements, and the cost of computing the distance between the point and the hyperrectangle is decreased significantly. Note that using this algorithm we are still guaranteed to find the nearest neighbor, since we under estimate the distance to the hyperrectangle, and thus we will search at least every node searched by the second algorithm. (see Section 4.2 for empirical results and comparison of the algorithms in the case of automatic interpretation of floor plans).

3.3 Library Stored as a Set of Ranges

In this storage method, the training set samples are grouped into their respective classes. For each class and for each feature, the lowest and the highest value are stored. This defines a set of n-dimensional hyperrectangles, where n is the number of features. Given a vector that needs to be classified, we find which hyperrectangle contains this point[†]. If no such hyperrectangle exists, then we find the nearest one using the following distance metric:

$$DH_E = \sqrt{\sum_{i=1}^{N} w_i \cdot Dist(F^L_{i_{low}}, F^L_{i_{high}}, F^I_i)^2}$$

where w_i is the i^{th} weighting factor, $F^L_{i_{low}}$ is the low value of the i^{th} feature of the library range set, $F^L_{i_{high}}$ is the high value of the i^{th} feature of the library range set, and F^I_i is the i^{th} feature of the input vector. *Dist* is defined as

$$Dist(low, high, val) = \begin{cases} 0 & \text{if } low \leq val \leq high \\ low - val & \text{if } val < low \\ val - high & \text{if } val > high \end{cases}$$

Using this method the size of the library is reduced to the number of classes rather than the number of elements in the training set, resulting in much smaller storage requirements. To find the nearest hyperrectangle to a given point q we use an exhaustive search. That is, DH_E is computed for each class in the library and q. The class with the minimum distance is assigned to q. The time to classify a point using this method is thus $O(K \cdot I)$ where K is the number of classes in the library, and I is the number of features in the feature vector. The number of classes is generally much smaller than the number of training samples, and thus this method results in improved time performance along with smaller storage requirements compared to those of the unordered list with an exhaustive search. It is also possible to use spatial indexing by representing the hyperrectangles as points in a higher dimensional space or by an MX-CIF tree[6] to further enhance this method. This is a subject for further research.

4 Experimental Study

The storage and classification techniques described in the previous section were applied to the automatic interpretation of floor plans. The input to the system was a binary image of a floor plan, created using a very primitive drawing package that provides line and curve drawing. Some example classes in the floor plan image are windows, rooms, bath tubs, toilets, and beds. See Figure 2 for an example of a floor plan that was used as a test case. Once the classification of the objects in the floor plans is achieved, the floor plans may be stored in an image database in a much more informative way than if they were stored as raw images. Along with the classification, the system also returns the coordinates of the locations of the objects. The objects may now be stored along with information about the spatial relationship between them. This information may then be used to index the floor plans and to answer meaningful queries on this database.

[†]We assume that the hyperrectangles do not overlap since the features discriminate between the classes and thus the ranges should be disjoint.

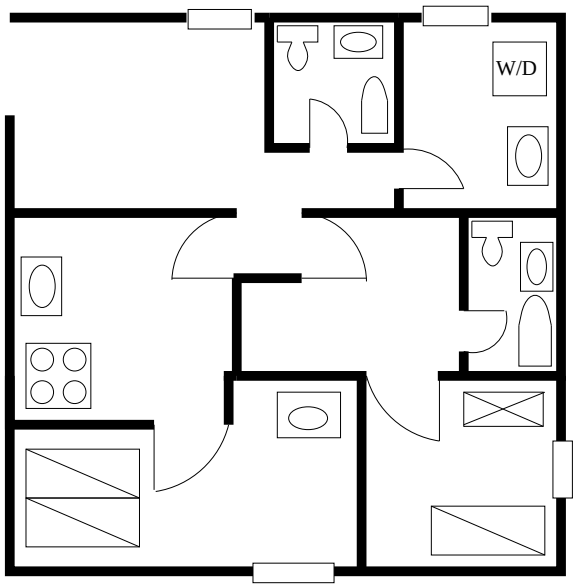

Figure 2: An example floor plan used as a test case for our system.

4.1 Feature Extraction

The image was segmented using a connected component labeling algorithm. Region numbers were assigned to each component. The output of this stage is an image in which every pixel has its region number as its value. Khoros[8], an integrated software development environment for information processing and visualization, was used for this stage and for the feature extraction stage.

Next the feature vector for each region found in the previous stage was computed. The output of this stage consists of a list of the feature vectors for each region, along with the region's centroid. For the training phase of the system, we manually tagged a classification for each region that was identified. This list was then passed to the next stage where the data structure was built. For our application, we used five features: first invariant moment[5], eccentricity[9], circularity[4], percent of bounding box, and percent of total image.

4.2 Results

We experimented with each of the three data structures described in the previous section. The unordered list was used as a basis for determining the best recognition rate that we could achieve using our training set, the discriminating features that we had selected, and the nearest neighbor classification method. There were 11 different classes in the floor plans. Our training set consisted of 130 sample points. We used this training set to classify 200 objects. Both the training and the test objects were extracted from floor plans. The objects in the floor plan (i.e. toilets, windows, etc.) were drawn with different scales and orientation (e.g. rotation) in each floor plan.

For the adaptive k-d tree we experimented with the three versions of the nearest neighbor search described in Section 3.2. We refer to them as algorithms A, B, and C respectively. We also varied the bucket size in our study. We used 1, 2, 4, and 8 points per bucket. Table 1 summarizes our results for the three algorithms. Two comparison percentages are presented in this table. The first only takes into account the comparisons performed while descending the tree. This gives us a good indication of how well the algorithm pruned the search space. The second comparison percentage takes into account the comparisons required to compute the distance to neighboring hyperrectangles when backtracking, thus giving us an indication of the true cost of the algorithm. The percentage of comparisons made are relative to the results when using unordered lists. Using the class range method (Section 3.3) we had a recognition rate of 98%. Using this method, one

	1 point per bucket		2 points per bucket		4 points per bucket		8 points per bucket	
Algorithm	descending percent comparisons	total percent comparisons	descending percent comparisons	total percent comparisons	descending percent comparisons	total percent comparisons	descending percent comparisons	total percent comparisons
A	13.43	17.72	14.28	17.33	16.2	18.24	18.74	20.17
B	5.82	17.6	7.24	16.82	9.78	17.19	14.22	19.73
C	12.84	16.91	13.4	16.32	15.33	17.33	18.17	19.59

A: When backing up during tree traversal, check the distance to hyperrectangle represented by brother (need only to compute for discriminating coordinate).

B: For each hyperrectangle, store the minimum and maximum values of each coordinate among the points stored in the hyperrectangle. When backing up during tree traversal, check the distance to the hyperrectangle defined by these coordinates (need full distance computation).

C: For each hyperrectangle, store the minimum and maximum values of the discriminating coordinate among the points stored in the hyperrectangle. When backing up during tree traversal, compute the distance only for the discriminating coordinate.

Table 1: Results of floor plan interpretation using adaptive k-d trees.

comparison is made for every class rather than for every training feature vector. Thus since there are 11 classes, the percent of comparisons made compared to the unordered list method was 8.5%.

As expected, our experimental results indicate that using spatial indexing to store and search the training set resulted in making a relatively low number of comparisons, while maintaining high recognition rates. For all three algorithms, we achieved the best results with only 2 points per bucket. Although a small number of points per bucket results in a deeper tree, the number of comparisons performed while descending the tree is less crucial, since usage of a k-d tree means that only one feature is compared at each node. When taking into account the total number of comparisons, the best method was found to be algorithm C with 2 points per bucket. It is interesting to note the significantly small number of tree descending comparisons used with algorithm B, indicating that a very good pruning of the search space was achieved, but at a relatively high cost which is incurred during backtracking, since a full distance calculation is needed rather than just along one dimension as in algorithm C. Thus there was overall only a small improvement over algorithm A (i.e comparable results). Further research is needed to determine whether a better compromise than algorithm C can be found. Another interesting result that we observed is that with eight points per bucket and a search strategy that finds the nearest neighbor within the bucket to which the query point belongs, the recognition rate was 85% with only 7.4% comparisons compared to an exhaustive search.

5 Conclusion

A number of issues remain to be addressed in order to use our method as a means of intelligently extracting meaningful data from raw images and storing this data in an image database. The images that were used as test cases were very clean. We need to experiment with images that have some noise in them. The features used to discriminate among the classes, and the weights assigned to these features were selected using rather ad-hoc methods. We need to devise methods to do this in a more systematic way. Finally, the use of contextual data to aid in the classification problem must be explored. Contextual data is heavily relied upon in OCR systems. Similar methods may be used in the interpretation of documents that are graphical in nature such as floor plans, engineering drawings, and maps. Knowledge about the domain of the images stored in the database such as possible spatial configurations among the objects, and proximity between the objects may be used to improve the classification process.

6 References

1. P.A. Devijver and J. Kittler. On the edited nearest neighbor rule. In *Proceedings of the 5th International Conference on Pattern Recognition*, pages 72–80, Miami Beach, FL, December 1980.

2. J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

3. K. Fukunaga and P.M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computing*, 24(7):750–753, July 1975.

4. S.B. Gray. Local properties of binary images in two dimensions. *IEEE Transactions on Computing*, C-20(5):551–561, May 1971.

5. M.K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, IT-8(2):179–187, February 1962.

6. G. Kedem. The quad-CIF tree: a data structure for hierarchical on-line algorithms. In *Proceedings of the 19th Design Automation Conference*, pages 352–357, Las Vegas, Nevada, June 1982.

7. L. Miclet and M. Dabouz. Approximate fast nearest-neighbor recognition. *Pattern Recognition Letters*, 1(5,6):277–285, July 1983.

8. J. Rasure and C. Williams. An integrated visual language and software development environment. *Journal of Visual Languages and Computing*, 2(3):217–246, September 1991.

9. A. Rosenfeld and J.L. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1(1):33–61, July 1968.