Texture based recognition of topographic map symbols

Rudolf Szendrei, István Elek, István Fekete
Eötvös Loránd University, Budapest
swap@inf.elte.hu, elek@map.elte.hu, fekete@inf.elte.hu

***Abstract*—This paper introduces a method that is able to assign the symbols of vectorized maps into polygon attributes. It reduces the required storage space of the database by removing the corresponding polygons from the vectorized data model.**

This procedure is presented with optimized pattern matching on the raster image source of the map, where the symbols are handled as special textures. This method will be improved by using a raw vector model and the kernel symbols.

I. Introduction

This paper will describe a method recognizes symbols during the raster-vector conversion of maps. Maps that contain topographic symbols are made from vector data models, because photos and remote sensed images obviously do not contain map symbols.

If a map symbol is identified, then two transformation steps can be made automatically instead of the usual manual interaction [1], [3]. First, the vectorized polygon of the map symbol will be removed from the vectorized map. Next, the meaning of the removed symbol will be assigned as an attribute to the polygon of the corresponding object in the vector data model. For instance, after removing the symbol “vineyard”, this attribute will be added to the boundary polygon of the “real” vineyard (see Fig. 1). In practice, the attributes of the polygons are stored in a GIS database.

II. Main steps of raster-vector conversion

The raster-vector conversion of maps consists of three main steps.

A. Color classification

In the first step, the amount of the colors will be reduced in regard to the number of colors that the human eyes can logically separate during the interpretation of the map. This process can be set up as a series of image filters. These filters reduce the errors of the image, emphasize dominant color values or increase the distance between color classes. After these filters were applied, the intensity values of pixels are classified into color classes by clustering methods. Our goal is to determine appropriate reference colors in order to minimize the false pixel classifications.

Rudolf Szendrei is with the Department of Software Technology and Methodology, István Elek is with the Department of Cartography and Geoinformatics and István Fekete is with the Department of Algorithms and Their Applications, Eötvös Loránd University of Budapest (e-mail: swap@inf.elte.hu, elek@map.elte.hu, fekete@inf.elte.hu).

The research project was supported by the IKKK (Informatics Research and Education Cooperation Center) under project code GVOP-3.2.2-2004-07-0005/3.0.

B. Detecting vectors

This step will determine all edge vectors in the color reduced map. Edge filters and edge detectors, like Canny, Laplace or Prewitt methods are frequently used to solve this problem. Using these filters, we can obtain the direction vectors for each pixel that sits on a line. This will create the corresponding vector model. If a pixel does not belong to any edge, it can be dropperd or represented by a null vector.

C. Processing vectors

The last task is to process the vectors. This means, the extraction of as much structural information as only possible and storing them with the vectorized map in the database. This step will build polygons or polylines from the vectors determined for each pixel.

Fig. 1. Recognizing the symbol of vineyard

Experience shows the most difficult part of the raster-vector conversion is the third step (see Fig. 2). Let us examine the roads on a map for illustration. The width of their polylines can be different according to their types. In this case, most software interpret them as polygons which have edges on both sides of roads because of their width. This is not a correct representation of roads as the width property is only a symbolic attribute of the road and not a real measure. This kind of false classification is well known, and even the recent applications do not yield complete solution to this problem.

III. Symbol recognition

It is important to recognize those objects of the map which represent a symbol, even if they look like lines or polygons.

The use of texture based pattern matching algorithm developed by the authors will directly recognize these symbols. This algorithm also determines the positions of symbols on the map. The position is needed in order to query its respective polygon from the vector model. This polygon will be removed from the vector model and its attribute property (e.g. "vineyard") will be assigned to the polygon that contained the polygon of the removed symbol. A second query is required to determine the polygon that comprised the symbol [2].

Character recognition is a special case of symbol recognition [4]. It is assumed, that maps have a legend of symbols on the map sheet or the map interpreter identifies the map symbols (see Fig. 2).

A map can be represented as an $m \times n$ matrix, where each pixel described by a k number of color components. It is assumed, that a part of the map represents the symbol as a $u \times v$ matrix. It is possible that symbols are not rectangular. This difficulty can be handled by using an other $u \times v$ matrix that represents a bitmask. This matrix determines which pixels of the symbol will be used during pattern matching. Section IV. will show a simple, section V. an improved pattern matching method.

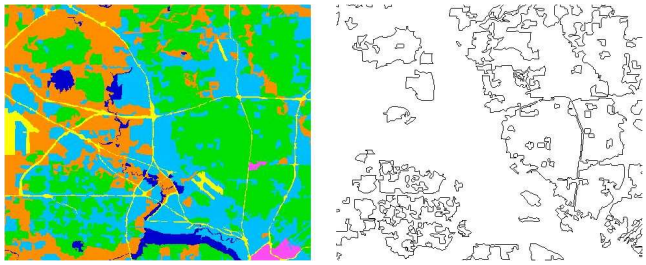

Fig. 2. The original map and its vectorized model using R2V software

IV. A simple pattern matching method

The basic method applies a brute force pattern matching as it tries to match the matrix of the symbol to each $u \times v$ matrix of the map. This is an inefficient solution, because it determines for each pixel of the map whether the pixel is a part of a symbol or not. Each map pixel can be covered by a $u \times v$ matrix in $u * v$ different ways. This leads to a number of $u * v$ pattern matching where each costs $u * v$ pixel comparisons. Thus, the runtime in pixel comparisons will be

$$T_{bf}(m, n, u, v, k) = \Theta((m * n) * (u * v)^2 * k).$$

In addition, this method works only if the symbols on the map have the same orientation as in the symbol matrix. Unfortunately, polylines mostly have transformed symbols in order to follow the curves of a polyline. Symbols on a map can be transformed in several ways that makes the matching more difficult. In the least difficult case an affin transformation was made to a symbol, e.g. it was rotated. However, it can be much more difficult to recognize the non-located symbols (e.g. railroads which continously follow the curves of the track). In this project only the problem of rotated symbols was treated. Without additional concrete or contextual information the rotated symbols can be identified if the matching symbol is rotated too. If there is no knowledge of the orientations of symbols, a number of directions has to be defined as possible positions for rotated pattern matching. Refining the rotations makes the recognition more accurate. A correct pattern matching algorithm without any knowledge has to test at least 20-30 directions. If the symbol is asymmetric, it may be necessary to do the pattern match with the mirrored symbol too (e.g. country borders)

As the maps are often distorted or defected, statistical methods should be applied instead of regular pattern matching methods. Several tests are known for statistical pattern matching depending on the problem class and they mainly use the mean and variance of a matrix. This paper uses a simple statistical comparison called similarity function. It takes two $u \times v$ matrices as parameters and calculates the variance of their difference matrix. The pattern matching algorithm uses the variance as a measure of similarity. In practice, the user defines or the software calculates a threshold value which will be used for pattern matching decisions. Each map pixel covered by the $u \times v$ matrix of the symbol is part of the symbol when the value of the similarity function is less than the threshold.

V. Efficient pattern matching

Some commercial software support the raster-vector conversion process. The embedded algorithms are well known, and most of them are filters (e.g. edge and corner detectors, edge filters). The efficient implementations of these filters are usually available in both pseudo and source codes on the Internet, therefore, the programming aspects are not discussed here.

Despite the large number of filters, the Gauss and Laplace filters are used most often in digital image processing as edge filters, while Canny and Prewitt (see Fig. 3) methods as edge detectors.

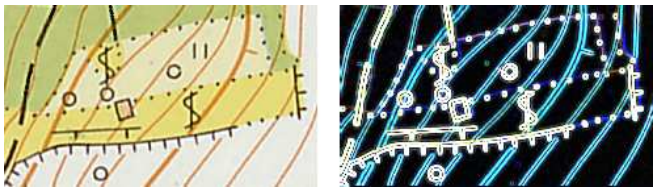

Fig. 3. An example for Prewitt filter

Our task is to enhance the efficiency of symbol recognition. As a starting point, the vector data model is needed in an uninterpreted raw format, which naturally contains redundant vectors. The goal is to create the model which is as similar to the raster image as only possible. From this model, those datas are required, which describe the presence of a vector and the directon of the vector (when it exists) at a given point. If a vector exists at a pixel of the map, then the pixel belongs to an edge, which is represented by a vector with direction d. If a vector does not exist at a point, no pattern matching is required there. In other words no symbol is recognized at this point. The pattern matching is much more efficient if only those map pixels and symbol pixels will be matched which sit on a line. Namely, these points have a vector in the vector data model.

It is assumed that total length of edges in the map is $l \leq m * n$, and the number of edge pixels in the symbol is $l_s \leq u * v$. The cost of pattern matching in a fixed position remain unchanged ($u * v$ pixel comparisons). The estimated runtime of the improved matching process is then

$$T_{\mathit{eff}}(m, n, u, v, k) = \Theta(l * (u * v) * l_s * k).$$

The total length of the edges may be $u * v$ at worst case. In this case the runtime can reach asymptotically the runtime of the brute force algorithm.

The effective runtime of this algorithm is certainly significantly less, because, in practice the total length of the symbol edges is a linear function of the diameter of symbols. As $\underline{u} = (u, 0)$ and $\underline{v} = (0, v)$ vectors are orthogonal, $|\underline{u} - \underline{v}|$ can be used to estimate l_s.

$$l_s < |\underline{u} - \underline{v}| * c < (u * v),$$

where c is a constant factor and

$$|\underline{u} - \underline{v}| = \sqrt{u^2 + v^2}.$$

These formulas lead to

$$min(u, v)\sqrt{2} < \sqrt{u^2 + v^2} < max(u, v)\sqrt{2}.$$

Now c can be estimated as

$$c < \frac{u * v}{min(u, v)\sqrt{2}}.$$

Because $min(u, v) = u$ or v,

$$c < \frac{u * v}{u\sqrt{2}} = \frac{v}{\sqrt{2}} \text{ or } c < \frac{u * v}{v\sqrt{2}} = \frac{u}{\sqrt{2}}.$$

The inequality is guaranteed, if

$$c < \frac{max(u, v)}{\sqrt{2}}.$$

To determine the efficiency of the improved pattern matching algorithm, the speed of the simple and the improved matching methods has to be estimated. The s_i symbol is a $u_{s,i} \times v_{s,i}$ matrix and u_{max} and v_{max} are the maximums of $u_{s,i}$ and $v_{s,i}$ values. The total length of edges in the ith symbol is $l_{s,i}$ and l_{sm} is the mean of the $l_{s,i}$ values. It can be assumed that $u = u_{max}$, $v = v_{max}$ and $l_s = l_{sm}$. If the map is totally covered by non-overlapping $u \times v$ matrices, the total length of the map edges l can be estimated as

$$l \approx l_s * \frac{m * n}{u * v}.$$

Because

$$l * u * v * l_s \approx l_s * \frac{m * n}{u * v} * (u * v) * l_s = m * n * l_s^2,$$

the "speed up factor" of the improved method is

$$\frac{T_{\mathit{eff}}}{T_{\mathit{bf}}} = O\Big(\frac{m * n * l_s^2 * k}{m * n * (u * v)^2 * k}\Big) = O\Big(\frac{l_s^2}{(u * v)^2}\Big).$$

VI. Finding the kernel of the pattern

Certain symbols are used as a tile in maps and this tile is called kernel. This often happens when the user selects a part of the map that is larger than the symbol. This part includes the symbol at least one occurence and may also contain the symbol partially. In this case the pattern matching is less efficient. The optimized algorithm uses the smallest tile (see Fig. 4). If a kernel K is a $u_K \times v_K$ matrix and S is a $u \times v$ symbol matrix, then

$$|S(i, j) - K(i \bmod u_K, j \bmod v_K)| < \frac{T}{u_K * v_K},$$

where $0 \leq i < u$, $0 \leq j < v$. Threshold T is used by the pattern matching algorithm applied on the original symbol.

The kernel can be determined, by for example, a brute force algorithm makes a self pattern matching with all the submatrices of the symbol matrix. Instead of using a brute force method of exponential runtime, the algorithm works with the vector data model of the symbol in the same way as it is used by the pattern matching algorithm.

Experience shows that the number of edge pixels in the vector data model is almost irrelevant in comparison with $u*v$. It is assumed that all tiles of the symbol matrix have the same direction in the selected area.

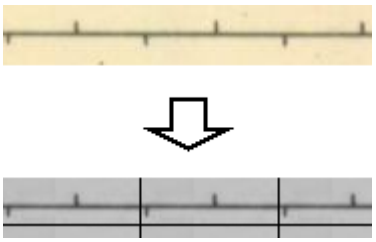

Fig. 4. Determining the kernel of the sample

Using vector data, the kernel of the sample can be determined by a motion vector searching algorithm. The details are not discussed here, because this algorithm is known in the image sequence processing to increase the compression ratio. (For example, the standard of MPEG and its variants use motion vector compensation and estimation to remove the redundant image information between image frames.)

VII. Linearizing the number of pattern matching

To apply the method of pattern matching, the previously determined kernel will be used. Let u denote the horizontal and v the vertical dimension of the kernel. A useful property of the kernel, which is the smallest symbol, is that it can be used as tiles to cover the selected symbol. The kernel never overlapped by itself. At this stage, the algorithm freely selects an edge pixel of the kernel. It is assumed, that the kernel can be matched in one orientation. The other pixels of the map region, which is covered by the kernel, do not need to be evaluated. In best case, the $u * v$ pixels of the map have to be used only once, that is all the pixels of the map are processed only once. Calculating with the number of rotations of the symbol, the runtime in optimal case is

$$T_{\mathit{eff}}(m, n, u, v, k, r) = \Theta(l * (u * v) * l_s * k * r),$$

where k is the number of color components and r is the number of tested rotations.

The vector which belongs to a pixel may have two direction. Therefore, in each selected part $r = 2$. The runtime that includes the cases of rotated symbols will be

$$T_{eff}(m,n,u,v,k) = \Theta(l*(u*v)*l_s*k*2) = \Theta(l*(u*v)*l_s*k).$$

When there is a symbol that is not represented in the pixel of the map, then two cases are possible

1) the pixel is not a part of an edge, or
2) the pixel is a part of an edge, but it is not identified as a part of the symbol in the given direction.

In the first case, no further pattern matching is needed. In the second case, an edge pixel of the symbol will be fixed, which is a part of an edge, and the pattern matching algorithm will start to work with rotating. The angle of rotation α can be calculated as

$$\alpha(d_m, d_s) = R(\frac{\underline{d}_m - \underline{d}_s}{|\underline{d}_m - \underline{d}_s|}).$$

where d_s is the vector that belongs to the fixed edge pixel of the symbol, d_m is the current starting map pixel of the pattern matching and the function R returns the angle of the given vector according to $\underline{i} = (1,0)$. The worst case gives the following runtime:

$$T_{eff,\ worst}(m,n,u,v,k) = \Theta(l*(u*v)*k).$$

Using the estimation

$$l \approx l_s * \frac{m*n}{u*v},$$

the runtime is

$$T_{eff,\ worst}(m,n,u,v,k) = \Theta(m*n*l_s*k) = \Theta(m*n).$$

In practice, k is a constant value (e.g. $k = 3$ for RGB images) and the value l_s has an upper boundary, which is not influenced by the size of the map. Therefore, the pattern matching algorithm works in linear runtime.

VIII. Map symbols as attributes

Finally, the represented attribute of the symbol has to be assigned to the corresponding object of the vectorized map. In order to do this, polylines and polygons should be handled differently. All segments of the polyline should inherit the attribute of the polyline symbol. The assignment to polygons is more sophisticated, because both the border and the interior of a polygon have to receive the attribute. The decision is user dependent, whether the attribute information is stored implicitly – assigned only to the polygon – or explicitly – assigned to all polyline segments of the polygon border.

IX. Conclusions

A texture based pattern matching algorithm was introduced that recognizes the symbols of a map. The algorithm needs both the raster and the raw vector data model of the map. This method makes it possible to assign the attribute of the symbol to the corresponding vectorized objects. The result is an interpreted vector data model of the map that does not have those vectors which were part of the vectorized symbol. The process begins on an apropriate part of the map representing a symbol, selected by the user or the software. After this step, the algorithm makes pattern matches and determines the positions of the symbol on the map automatically. The complete workflow can be seen on Fig. 5, with an optional component. The quality of the recognition is heavily influenced by the filter algorithms used before the pattern matching.

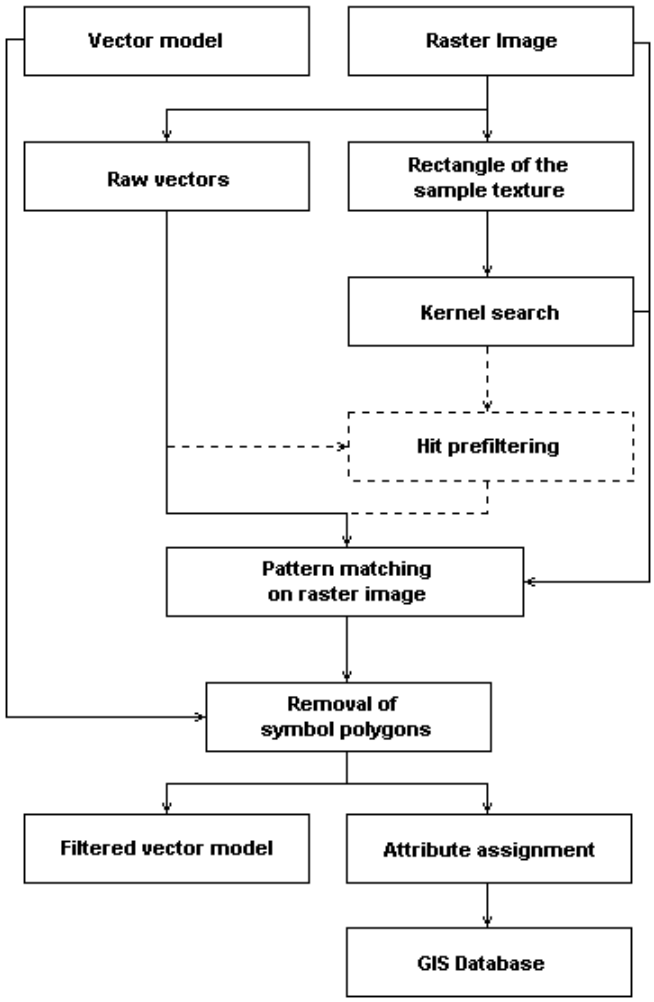

Fig. 5. The complete workflow.

References

[1] Ablameyko, S., et al. (2002), *Automatic/interactive interpretation of color map images.* Pattern Recognition, Vol. 3, pp. 69–72.

[2] Bhattacharjee, S. and Monagan, G. (1994), *Recognition of cartographic symbols.* MVA'94 IAPR Workshop on Machine Vision Applications, Kawasaki

[3] Levachkine, S. and Polchkov, E. (2000), *Integrated technique for automated digitization of raster maps.* Revista Digital Universitaria, Vol. 1., No. 1, http://www.revista.unam.mx/vol.1/art4/

[4] Trier, O. D., et al. (1996), *Feature extraction methods for character recognition - A survey.* Pattern Recognition, Vol. 29, No. 4, pp. 641–662.