Acta Cybernetica **00** (0000) 1–21.

A Knowledge-Based Approach to Raster-Vector Conversion of Large Scale Topographic Maps*

Rudolf Szendrei†, István Elek‡, Mátyás Márton§

Abstract

Paper-based raster maps are primarily for human consumption, and their interpretation always requires some level of human expertese. Todays computer services in geoinformatics usually require vectorized topographic maps. The usual method of the conversion has been an error-prone, manual process.

In this article, the possibilities, methods and difficulties of the conversion are discussed. The results described here are partially implemented in the IRIS project, but further work remains. This emphasizes the tools of digital image processing and knowledge-based approach.

The system in development separates the recognition of point-like, line-like and surface-like objects, and the most successful approach appears to be the recognition of these objects in a reversed order with respect to their printing. During the recongition of surfaces, homogeneous and textured surfaces must be distinguished. The most diverse and complicated group constitute the line-like objects.

The IRIS project realises a moderate, but significant step towards the automatization of map recognition process, bearing in mind that full automatization is unlikely. It is reasonable to assume that human experts will always be required for high quality interpretation, but it is an exciting challenge to decrease the burden of manual work.

Keywords: Geoinformatics, topographic maps, raster-vector conversion, artificial intelligence, knowledge representation

1 Introduction

Paper-based raster maps are primarily appropriate for human usage. They always require a certain level of intelligent interpretation. In GIS applications vectorized

*This research was supported by the project TÁMOP-4.2.1/B-09/1/KMR-2010-003 of Eötvös Loránd University.

†PhD student, E-mail: swap@inf.elte.hu

‡Department of Cartography and Geoinformatics, E-mail: elek@map.elte.hu

§Department of Cartography and Geoinformatics, E-mail: matyi@map.elte.hu

The authors are at the Faculty of Informatics, Eötvös Loránd University of Budapest, and they are members of the university research group in geoinformatics T.E.A.M. (http://team.elte.hu/)

maps are preferred. Especially, government, local authorities and service providers tend to use topographic maps in vectorized form. It is a serious challenge in every country to vectorize maps that are available in raster format. This task has been accomplished in most countries — often with the use of uncomfortable, "manual" tools, taking several years. However, it is worth dealing with the topic of raster-vector conversion. On one hand, some results of vectorization need improvement or modification. On the other hand, new maps are created that need vectorization. This is valid not only for topographic maps, but also for remote sensing images reflecting the status of agricultural areas.

The theoretical background of an intelligent raster-vector conversion system has been studied in the IRIS project [5]. Several components of a prototype system has been elaborated. It became clear very early that the computer support of conversion steps can be supported at quite different levels. For example, a map symbol can be identified by a human interpreter, but the recognition can be attempted with a software, using the tools of image processing. Therefore it is also valid that the level of intelligence of the raster-vector conversion system can be various. A computer system can be fairly valuable and usable even if every important decision of interpretation is made by the expert user. However, the system designed and developed by the authors is aimed at to automatize the raster-vector conversion as much as possible. This aim gives an emphasis to the knowledge-based approach.

Two types of expert knowledge in connection with maps are distinguished (see Fig. 1). First type consists of professional knowledge needed the interpretation of maps. This is required to derive an equivalent in vector format from a paper-based scanned topographic map. The basic level information is contained in a standard file. More sophisticated relationships are usually stored in a relational data base, connected to vectorized data. The other type of expertise consists of the knowledge needed to draw conclusions based on vector data model and the related data base contents.

There are several research results on the latter topic [1, 4]. Examples of questions to be answered are the area that will be inundated by water in case of a flood of a river, or how economic can the exploitation of an oil field. Beyond answers consisting of numerical data, expert systems often use visualization for better understanding [2].

It is important to realize that the adequacy of answers given by an expert system does not only depend on the inference rules applied [3], but also on the quality of stored data used as input of these rules [6, 10]. This also emphasizes the importance of map interpretation knowledge. In this paper knowledge-based approach means the first type of knowledge mentioned above, i.e. raster-vector conversion.

This paper deals with a part of raster-vector conversion applied in cartography, with knowledge-based approach. The types of map symbols used in topographical maps will be introduced, together with the algorithms used to recognize them. The organization of expertise into knowledge base will also be presented.

The following must be considered in connection with good quality and automated vectorization. Raster maps contain numbers, inscriptions and other kinds of data, but the majority of information is contained in a special cartographic

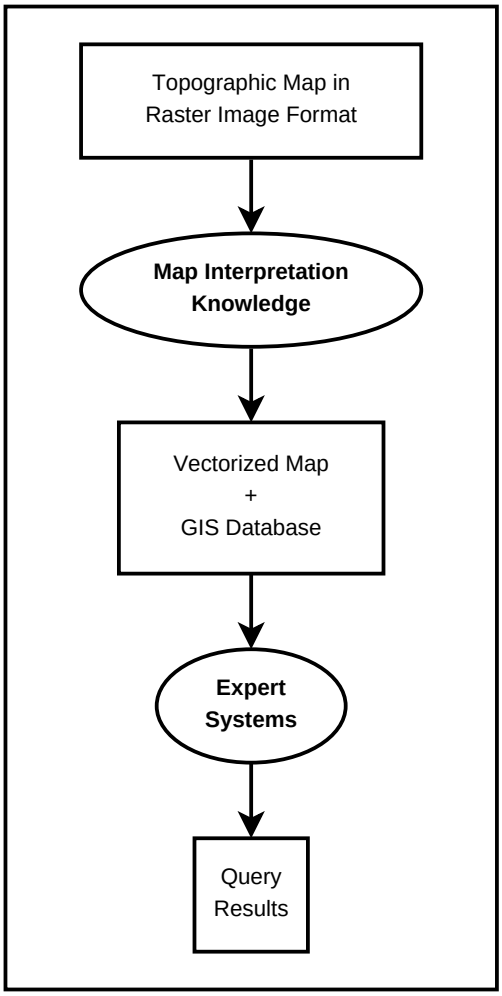

Figure 1: Knowledge representation in geoinformatics

context that can be adequately understood only by human expert. These relationships used for interpretation are no more contained in the vectorized map — it consists only of numerical and descriptive data. Vectorization necessarily involves some loss of information, this is why the depth of conversion must be carefully defined. Automatic interpretation of image contents requires sophisticated image processing tools, which are not comparable to human perception in the majority of cases. Therefore, the level of automatic recognition must also be appropriately determined.

2 Map symbols

The topic of this article is how to interpret printed variant of maps and how to represent them in computer systems. This process is considered basically as the result of interpretation and processing of map symbols. To accomplish this task it is very important to understand maps, and specifically, map symbols. To gain a comprehensive survey, refer to [8]. Although human cognition can not be completely understood, it is necessary to know to a certain extent how the human expert interprets graphical information. Regarding human perception, primarily points, lines and textured surfaces are sought and distinguished (see Fig. 6). It must be realized that human perception may reveal finer or hidden information, for example how roads crossing at different levels hide each other. Human mind is also capable of abstraction, for example when it disregards the actual texture of surface, and investigates only its shape. Human eye can make some corrections, for example in the determination of shades of color layers printed over each other.

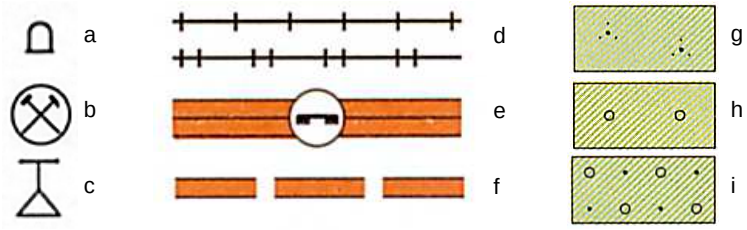

Figure 2: Point types: a) statue, b) mine, c) thunderhead. Line types: d) railway, e) highway with emergency phone, f) highway under construction. Area types: g) scrub, h) orchard, i) orchard with bushes.

Map interpretation process and the complexity of knowledge based object recognition can be visualized via any example of the four different object types — that is, point, line surface and inscription. IT tools seem to be capable of accomplishing map interpretation steps, but the extent that can be reached with human perception can not be approached by the IT technology of present days.

Point-like elements are usually graphical map symbols of small size, which often covers relatively larger area on map than the real object it represents. Nevertheless, its reference point can always be identified. Similarly, line-like elements can also cover larger area on map than their real size. For instance in the case of a highway, a zero-width center line can represent the theoretical position of the road in the database. Beyond the graphical properties of lines the database may contain real physical parameters, such as road width, carrying capacity, coating (concrete, asphalt) etc.

Hiding is a very inherent phenomenon in maps when line-like objects, landmarks (typically roads, railways and wires) located at different elevations intersect. This results in discontinuity of objects in map visualization. However, in map interpretation continuity must be assumed. Interpretation is not so straightforward in the

case of surface elements. The discontinuity of surface in map does not generally mean discontinuity of the real object it represents, e.g. in the case of a bridge over a river represented by water surface. However, a dam in the map may actually mean the separation of different levels of water.

The recognition and interpretation of inscriptions take place at two levels. Basically, the name represented by the inscription must be recognized, which may be accomplished with the support of a name data base. Interpretation means the establishment of appropriate connection between landmarks and their names. The recognition of inscriptions goes beyond the goals of this article.

3 Knowledge-based approach

One of the main tasks of geoinformatics is the conversion of available raster maps to vector format. Nowadays this is dominantly a manual task; it involves the tracing of polygon boundaries. This work is supported by the currently used software products, but the map interpretation is out of their scope. This is considered as a complex task in geoinformatics, which needs the intelligent application of various image processing algorithms. Our aim is to implement such a system that takes the majority of expert work, but it makes possible user interaction when it is necessary (see Fig. 3).

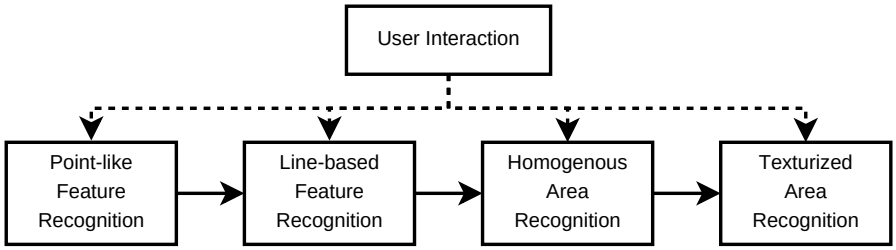

Figure 3: The flow-chart of the thematic map interpretation system.

In our approach, map symbols can be categorized into three groups according to their recognition algorithm. *Point-like* map symbols are used to mark objects that can be bound to a certain location (see Fig. 6 $a-c$) or land cover types, e.g. vineyard symbol. *Line-like* map symbols usually mean different types of roads and railroads, see Fig. 6 $d-f$. *Texturized surface* map symbols mean a texture covering a given type of land cover, see Fig. 6 $g-i$). Beyond these map symbols, extra care should be taken to recognize polygons delimiting areas.

The map representation of some objects involves several classes. For example, a lake is basically represented by a homogeneous blue surface, but its boundary is described by a line-type element. The case of buildings is similar, but their boundaries are more dominant than their (usually pink) surface.

One of the most difficult problems is the detection of rivers, as they can be represented either as line-like or surface element, depending on their width (e.g., in

case of a delta). In Table 3 below the categorization of some typical map objects can be seen.

Object	Point-like	Line-based	Texturized surface
Letter	X	–	–
Vineyard symbol	X	–	–
Road	–	X	–
Railway	–	X	–
House polygon	–	X*	–
River	–	X*	X
Lake	–	X*	X
Delta	–	X*	X
Field	–	X*	X
Forest	–	X*	X

Table 1: X - feature type can be recognized, * - boundary feature

In the following sections the knowledge-based algorithms used to recognize different map symbols will be presented.

In order to compile a map interpretation system from these algorithms, the order used during printing the map must be known, and the way of thinking of human interpreting the map must also be taken into account. As the first step an overview will be given on rules assembling maps, each resulting in an individual layer.

1. The boundaries of polygons are drawn.
2. Polygons are filled with a solid color or covered by a texture.
3. The road and railroad network is drawn, using their respective map symbols.
4. The map symbols of point-like objects are put onto the map.
5. Inscriptions belonging to point-like objects are printed.
6. Inscriptions belonging to line-like objects are printed following the arc of line — either next to the line or directly onto the line, omitting the section covered by the inscription.

During vectorization, these steps are executed in reverse order so as to collect map components. Map layers are apparently printed onto each other; top layers may hide elements belonging to layers beneath them. Within cartography, a more complex set of rules is used. Especially, conflicts may arise within one layer as well. For example, some characters may be omitted from inscriptions that intersect each other to avoid overlaps (map generalization).

4 Recognition of Point-Like Symbols

Point-like symbols are small objects (see Fig. 6 $a - c$), which usually also appear in the legend. They appear on the map undistorted, though they may be rotated.

In a previous article [11] the authors introduced an efficient, linear time algorithm for the recognition of point-like symbols (see Fig. 4), which is also capable of recognizing surface textures, since texture is nothing more than a point-like object repeated a number of times. We assume that for each map scale, an individual symbolset is used.

The following algorithm attempts to recognize all point-like objects.

1. Do edge detection (Canny or Laplace) on the whole map m.
2. For all possible symbols (appearing in the database or requested by the user) do the following:
   a) Let s_{edge} the result of the edge detection applied on the current symbol s of size $s_x \times s_y$.
   b) Let s_{otsu} the Otsu-thresholded image of s_{edge}.
   c) Find the first pixel (u, v) of s_{otsu}, searching top-down, left to right, and determine the corresponding direction angle $s_\Theta(u, v)$ by the gradients calculated on s at (u, v).
   d) For all edge pixel $m(x, y)$ of the transformed map image determine the corresponding direction angle $m_\Theta(x, y)$ by the gradients calculated on m at (x, y) and perform the following steps:
      i. Rotate the original raster image of the point-like object around point $s(u, v)$, by the angle $m_\Theta(x, y) - s_\Theta(u, v)$ to get S_{mat}, that is the matrix representation of the symbol rotated.
      ii. From the values of S_{mat} substract the values of the underlying map pixels considering $m(x, y)$ as the origin, to get the difference matrix D_{mat}.
      iii. Calculate the standard deviation σ of D_{mat}.
      iv. If σ is smaller than a given threshold, then the point-like symbol is recognized with coordinates $(x - u + s_x/2, y - v + s_y/2)$. The coordinates and the type of symbol are exported into a file or database.

	Laplace-I	Laplace-II	Prewitt	Sobel
Map	23.05%	29.07%	20.72%	27.89%
Symbol	15.63%	21.88%	32.81%	35.94%

Table 2: Ratio of edge pixels after an Otsu-thresholding is applied.

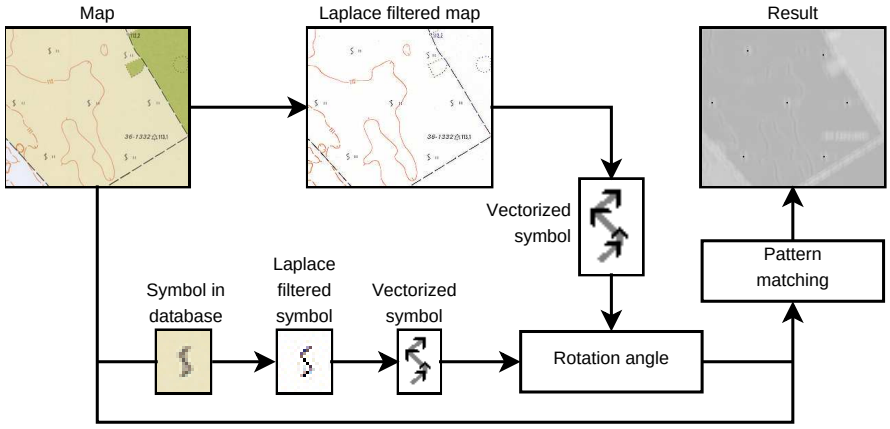

Figure 4: The flowchart of the recognition algorithm for a given symbol. The result image and the filtered images are in negative for better visibility. Each black point on the result image represents a recognized symbol.

The algorithm does edge detection to omit those map points, where pattern matching is unnecessary. The edge detection is also needed to determine which pixel of a symbol should be matched to an edge pixel of the map. We made Otsu-thresholding on the results of four different edge detectors and counted the remaining edge pixels (see Table 2) to decide which edge detector is the most useful. We chose Laplace-I, because it gives less points where pattern matching has to be made. It is also important to choose the interpolation method to symbol rotation. Simple point sampling interpolation produces unneeded distortion, since it only chooses the closest neighbour to a pixel. Bilinear and bicubic interpolation give better results (see Fig. 5). These interpolation methods are also considering the surrounding pixels. We found that bicubic interpolation keeps more image details, like edge features.

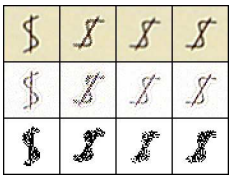

Figure 5: Top row: Original image, and its *Point sampling, Bilinear* and *Bicubic* interpolated rotations. Middle row: Results of Laplace-I. Bottom row: Results of Otsu-thresholding.

The method is compared to a rotation-invariant pattern matching method [12] based upon color ring-projection. That gives a high recognition rate (about 95%).

The authors wrote that "*Computation time of the proposed color ring-projection matching is 6 s on a Pentium 300 MHz personal computer for an arbitrarily rotated image of size 256 x 256 pixels and a circular window of radius 25 pixels.*" Nowadays, the algorithm may run on a Core i7 920 processor (without specific instruction sets, like SSE, etc.) approx. 35 times faster, but we have 100 megapixels resolution topographic maps. This gives us a runtime of 261 seconds in the case of a multi-threaded implementation. Because of the topographic symbols are simple graphics (versus natural images), we can perform the recognition with our parallelized algorithm in approx. 2-3 s on the same machine with a recognition rate $> 99\%$. In practice, the point-like symbols of a high detail topographic map can be vectorized manually in approx. 1 hour. We note that a great similarity of a point-like symbol and a map region can lead to a false positive match. These false positives can be easily removed manually after vectorization, as we did. In the future, we try to automatically remove the false positives based on line filtering methods, e.g. Hough transformation, Canny etc.

5 Recognition of Line-Like Symbols

Line-like symbols are usually the trace of a road like object, or edge of a polygon with a given texture/attribute.

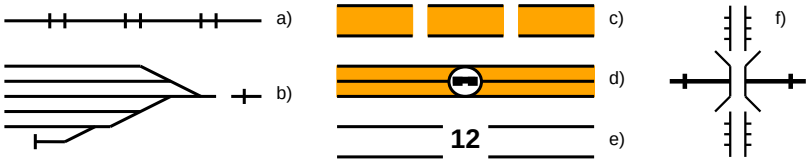

Figure 6: Examples for line-like symbols: a) railway b) railway network at a railway station, c) highway under construction d) highway with emergency phone. e) road with a specified width f) bridge above a canal

Recognition of line-like symbols is one of the most difficult tasks of raster-vector conversion. These symbols are often complex, and it is permitted for two symbols to differ only in their size, to cross each other or to join to form a single object (see Fig. 6a, b, respectively). Difficulties are posed by parallel lines belonging to the same object (see Fig. 6d, e) versus lines running in parallel which belong to separate objects. Further difficulties are the discontinuous symbols (see Fig. 6c, e, f).

It is beyond the aim of the current article to solve all the difficulties mentioned above, so for the purpose of this paper we assume that line-like symbols

1. do not cross each other, and
2. do not join to form a single object, and
3. are continuous.

A classic way of line-like symbol vectorization is introduced in [7], where cadastral maps in binary raster image format are vectorized. The additional features of color topographic maps, like road width, capacity, coating etc. can not be recognized the classical way [9]. Each of these features are represented by a corresponding graphics, color and structure. The following method is able to recognize the trace of the line-like symbols of a topographic map.

1. Do image segmentation and classify each pixel.
2. Create a binary map, where each black pixel belong to eg. road.
3. Apply thinning and morphological thinning on the binary map.
4. Vectorize the one pixel thin skeletons.

The first step is the segmentation, which works as follows. Define an object color set O and a surface color set S. The amount of colors in each color set is approx. 5-7 in the case of topographic maps. We assume that on a printed map each pixel color can be defined as a linear combination of a surface and an object color. In optimal case, this can be written as a $c = \alpha * c_o + (1 - \alpha) * c_s$ equation, where c is the value of the current pixel, and c_o, c_s are the respective object and surface colors, so the segmentation can be done by solving the minimalization task $\min_{o \in O, s \in S} |c - \alpha * c_o + (1 - \alpha) * c_s|$ for each pixel.

As the second step is a simple selection on the segmented pixels, it can be done easily. The third step consists of two different thinning methods. A general thinning method is used first to avoid creating unneeded short lines by morphological thinning. The general thinning can be described as it iteratively deletes pixels inside the shape to shrink it without shortening it or breaking it apart.

P_9	P_2	P_3
P_8	P_1	P_3
P_7	P_6	P_5

Table 3: 3×3 binary matrix, and the indices of its elements.

To decide whether an edge pixel P1 should be deleted, sider its 8 neighbors in the 3 by 3 neighborhood (see Table 3), $P_2, P_3, \ldots, P_8$ and P_9 and define:

- $N(P_1)$: number of non-zero neighbors ($N(P_1) = P_2 + P_3 + \cdots + P_9$).
- $S(P_1)$: number of 0 to 1 (or 1 to 0) transitions in the sequence $P_2, P_3, \ldots, P_9$.

The meaning of the values:

- $N(P_1) = 0$ (an isolated point)
- $N(P_1) = 1$ (tip of a line)

- $N(P_1) = 7$ (located in concavity)
- $N(P_1) = 8$ (not a boundary point)
- $S(P_1) \geq 2$ (on a bridge connecting two or more edge pieces)

Repeat the following steps, until no more change can be made

1. Mark all pixels satisfying **all** of the following: $(P_1 = 1)$ and $(2 \leq N(P_1) \leq 6)$ and $(S(P_1) = 1)$ and $(P_2 * P_4 * P_6 = 0)$ and $(P_4 * P_6 * P_8 = 0)$ and $(P_7 \neq 0)$.
2. Delete all marked pixels.
3. Mark all pixels satisfying **all** of the following: $(P_1 = 1)$ and $(2 \leq N(P_1) \leq 6)$ and $(S(P_1) = 1)$ and $(P_2 * P_4 * P_8 = 0)$ and $(P_2 * P_6 * P_8 = 0)$ and $(P_3 \neq 0)$.
4. Delete all marked pixels.

Because the result of the above algorithm may contain small pixel groups, a morphological thinning should be performed. This morphological thinning can be done by using the structuring elements shown in Table 4. At each iteration, the image is first thinned by the left hand structuring element (see 1st element of Table 4), and then by the right hand one (see 2nd element of Table 4), and then with the remaining six 90° rotations of the two elements. The process is repeated in cyclic fashion until none of the thinnings produces any further change. As usual, the origin of the structuring element is at the center.

0	0	0
	1	
1	1	1

	0	0
1	1	0
	1	

	1	
1	1	1

	1	
	1	1

Table 4: The first and second structuring elements are used to morphological thinning based skeletonization, while the third and fourth structuring elements are used to morphological fork detection on binary images. Values of the elements are: 0 - background, 1 - foreground. Empty places can be either 0 or 1.

The skeletonized binary image can be vectorized in the following way. Mark all object pixels *black* and surface pixels *white*. Mark those *black* pixels *red*, where N($P1$)$>$ 2, and then mark the remaining *black* fork points *blue* by using the 3rd and 4th structuring elements of Table 4 in the same way as structuring elements are used in morphological thinning. The *red* fork points are connecting lines, while *blue* fork points are connecting forks. Mark *green* each *black* pixel, if at most one neighbour of it is *black* (tip of a *black* line). It can be seen that a priority is defined over the colors as *white* $<$ *black* $<$ *green* $<$ *red* $<$ *blue*. The following steps vectorize the object pixels

1. Select a *green* point, mark *white* and create a new line segment list, which contains that point.
2. Select a *black* neighbour if it exists and if the current point is also *black*. Otherwise select a higher priority point. Mark *white* the point and add to the end of the list.
3. Go to Step 2, while a corresponding neighbour exists.
4. Go back to the place of the first element of the list and go to Step 2. Be careful that new points should be added now to the front of the list. (This step processes points in the opposite direction.)
5. Go to Step 1, while a *green* point exists.
6. Select a *black* point, mark *white*, and create a new line segment list, which contains that point.
7. Select a *black* neighbour of the current point, mark *white*, and put it at the end of the list.
8. Go to Step 7, while a *black* neighbour exists.
9. Select a *red* point p, mark *white* and create a new line segment list, which contains that point. Let $NeighbourSelect = RedSelect = Counter = 0$, $BlueFirst =$ false, $where =$ back, $q = p$.
10. Let $PrevPoint = q$.
11. If the $NeighbourSelect$th neighbour r of q exists, let $q = r$, let $BlueFirst = (Steps = 0$ and $where$=back), let $n = q$, and increment $NeighbourSelect$ by 1. Put q into the list at $where$ and go to Step 13.
12. If the $RedSelect$th neighbour r of q exists,
    a) If q and n are neighbours and $where$ =front, then let $q = PrevPoint$ and increment $RedSelect$ by 1. Go to Step 10.
    b) Put q into the list at $where$, mark q *white*, let $NeighbourSelect = 0$ and increment $Counter$ by 1. Go to Step 10.
13. If $where$=back, then let $where$=front, $q = p$ and go to Step 10.
14. Go to Step 9, while a *red* point exists.

Although, the algorithm above vectorizes all the objects, it merges the several object types and colors. Hence, pixels of a given object color are copied onto a separate binary image before they are vectorized.

We introduce an approach, which is able to recognize the features of line-like objects, so the corresponding attributes can be assigned to them. This assumes

that the path of each object exists in the corresponding vector layer. In order to recognize a specific feature, its properties should be defined for identification.

Two properties of vectorised symbols are recognized: forks ($F \sim Fork$), and end-points ($E \sim End$). Both are well known in fingerprint recognition where they are called *minutiae*. In the case of fingerprints, a fork means an end-point in the complement-pattern, so only one of them is used for identification. In our case, we can not define a complement-pattern, so both forks and end-points are used.

Representation of line-like symbols is based on weighted, undirected graphs. An EF-graph is an undirected graph with the following properties:

- Nodes are either of type E or F. The color of a node is determined by the corresponding vector layer.
- Two nodes are connected if the line-segment sequence connecting the nodes in the corresponding vector layer does not contain additional nodes. Edges running between nodes of different colors can be defined by the user (in case of multicolor objects). The weight of the edge is equal to the length of the road connecting the two nodes, and it has the color of the corresponding symbol part.
- There are special nodes, denoted by an index P, which occur on the trace of a line object. These will be used to produce the final vector model.

An EF-graph can also be assigned to the vectorised map, not only to the vectorised symbols, where line-like symbols are not separated to their kernels. For recognition we use the smallest units of the symbol, called the kernel. The smallest unit is defined as the one which can be used to produce the entire symbol by iteration. In the EF-graph there is usually only two nodes participating in the iteration; these are type F with only a single edge, so become the entry and exit points to the graph. In the very few cases, where the entry and exit points of the smallest unit can not be identified, the kernel of the line-like object is itself. Smallest unit can not be defined for the whole vectorised map.

Figure 7 shows how a symbol is built up from its smallest units by iteration. Weights represent proportions and depend on the scale of the map. Beside weights, we can assign another attribute to edges, their color. In the figure almost all edges are coloured black.

The recognition of line-like objects is reduced to an extended subgraph isomorphism problem: we try to identify all the occurrences of the EF graph of the symbol (subgraph) in the EF graph of the entire map. The weights of the EF graphs are normalized with respect to the scale of the map, and the collection is sorted in decreasing order of node degrees. Call this collection of sorted EF graphs S. Since the EF graphs created to maps do not contain the edges those connecting nodes with different colors, this case should be handled. In this article, the potential edges are identified by searching the corresponding neighbour on its own layer in the given distance of the node. The validity of a found potential edge is verified by

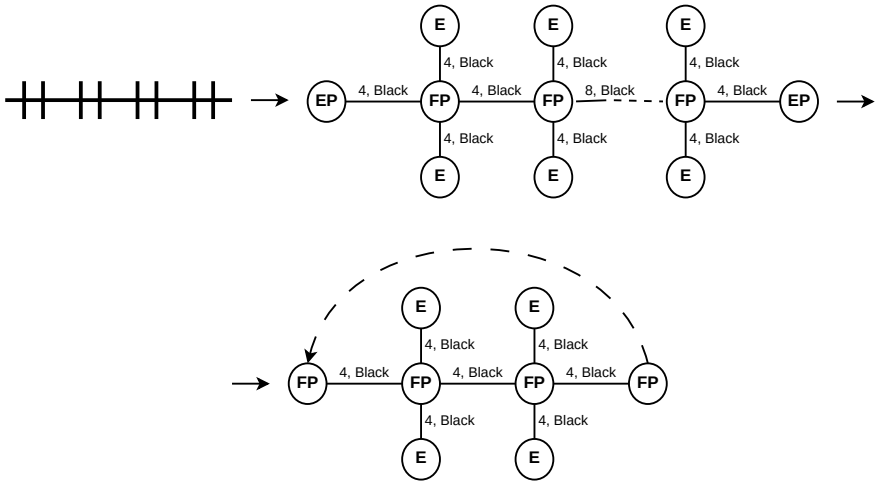

Figure 7: The EF graph and the elementary EF graph of a double railway line. Distances are relative values and refer to the scale of the map. The dashed line in the elementary EF graph represents its cyclical property.

comparing the color of the edge and the color of the segmented image pixels lying *under* the edge.

Subject to the conditions above it is possible to design an algorithm for the recognition of subgraphs. While processing the map, recognized objects are removed, by recoloring the corresponding subgraph. Two colors, say blue and red, can be used to keep track of the progress of the algorithm and to ensure termination.

The following algorithm stops when there are no more red nodes left in the graph.

1. Choose an arbitrary red node U with the highest degree from the EF graph of the map.

2. Attempt to match the subgraph at node U against S, that is the sorted collection of EF graphs, until the first successful match in the following way:

   a) Perform a "parallel" breadth-first search on the EF graph of the map and the EF graph of kernel of the current symbol with origin U. This is called successful if both the degree of all nodes match, and weights are the same approximtely.

   b) In the case of success, all matching nodes become blue, otherwise they remain red.

Upon successful matching the EF-graph of the symbol is deleted from the EF-graph of the map. Entry and exit points must not be deleted unless they are marked as E, and the degree of remaining F nodes must be decreased accordingly. The

equality of edge weights can only be approximate, due to the curvature of symbols. The algorithm above can be implemented, as it keeps track the given object by using the line segments as paths in vector data. The other difficulty is that edges with differently colored nodes are not directly defined by the vector layers. In practice, we have created a spatial database, which has contained the vectorized line segments and their color attribute. The potential edges was determined by a query, looking for existence of a neighbour with a corresping color in a given distance from the corresponding node.

6 Recognition of Homogeneous Surfaces

Although, some surface vectorization methods [5, 6] are working well on homogeneous surfaces, these methods are not able to deal with visually separated surface regions. During the recognition of surfaces, the most frequent tasks are the unification of areas which logically belong together, but are visually separated, and correction of map errors. For this purpose we use a mask, which identifies the points in the map which can be considered part of a surface. The mask can be defined as the area which remains after the removal of point and line-like symbols. Another possibility is to consider a mask to be the pixels of the approriate color that remain after color segmentation on the original map. Pixels belonging to the mask get the color of the surface, others are set to be UNDEFINED. For best results, we use both approaches combined. For recognition and high-quality polygonization we defined a heuristic rule system, which we discuss next.

6.1 Removal of the Pixels of False Surfaces

According to the rule, we remove all pixels, and groups of pixels of a given surface type if they occur less frequently then a given treshold. Many misclassified false surfaces are removed by this rule.

Surface type	**Searching region size**	**Minimal pixel occurence**
e.g. Grass	n px $\times$ n px	m pcs

Table 5: An example to remove false surface pixels.

In the example (see Table 5), a given pixel is classified as e.g. grass, if and only if, in the surrounding $n \times n$ square there are at least m pixels that can be classified as grass. At the boundaries of the image, where less pixels exist, the treshold is decreased appropriately. The rule is applied top-down, left to right. The process is not adaptive in the sense that mask is developed as a new image, while the original map remains unchanged.

6.2 Classification of Delimited Surfaces

On topographical maps it is usual practice that certain areas are delimited with a line, which forms a polygon. This helps recognition if we know that the polygon exclusively surrounds a homogeneous area of given type. In such cases, as soon as we determined a dominant surface we can classify the whole surface within the polygon. For example, in the case of a house, the color of the delimiting polygon is always black, and the surface within the polygon is pink. To determine if the polygon surrounds the given ares we use the usual "flooding" technique from any inner point. In practice, this needs a treshold (see Table 6) to limit "flooding".

Border	**Blob type**	**Flood limit**
e.g. Black	e.g. House	n px

Table 6: An example of the classification rule.

If flooding does not stop under the specified treshold n, we terminate and classify the area as UNDEFINED.

6.3 Joining Surfaces

The removal of point and line-like objects result in discontinuity of surfaces. We need a separate rule to join these (see Table 7), which describes what to do if there is an unexpected pixel during the top-down, left-right processing of the mask. From the given point, that is the occurrence of the first unexpected pixel, it searches within a predetermined limit n, in opposing directions for surfaces which have been classified as of the same type. If search succeed in finding such surfaces, then the color of the original pixel is changed to that of the surrounding surface. This process can be repeated m times or until no further change has made.

Blob type	**Searching radius**	**Iterations**
e.g. Grass	n px	m

Table 7: Rule to fill gaps between surfaces.

6.4 Removal of Small Damaged Areas and Holes

Small erros can be often found on maps. In a clearly defined area, there might be spots which apparently do not belong there. Forests, for example, often contain empty areas yet the whole should be classified as one. Another source of errors, is when the type of a small area is clearly identified, yet, due to its small size or its context this classification can not be accepted. In these cases, the anomalous pixels are removed. Table 8 gives further details.

The rules given above are mostly complete, in the sense that most pixels and areas are classified into one of the known types. Yet, unclassified areas may remain.

Blob type	**Bounding type**	**Blob size limit**	**Decision**
Non-forest	Forest	n px	$Type := Forest$
Forest	Non-forest	m px	Eliminate

Table 8: Eliminate blobs and holes on a forest layer.

Most of the time, those unclassified areas are surrounded by many areas of *different* types, so the techniques above fail. Table 9 shows a rule set which helps under such circumstances, but interestingly it also helps to reclassify previously classified areas. The table specifies for which pixels or areas should the rules apply, which surrounding areas are dominant, which areas are sub-dominant, and what other types must occur nearby. In other words, Table 9 is a higher level rule set, providing a sort of context sensitivity to the classification process. The distance limit can be interpreted as the maximum of the smallest diameter of the discontinuity, and the size limit determines the largest discontinuity which can be filled. If at least one of the conditions of the rule is satisfied the area can be reclassified to the dominant type.

Type	**Dominant type**	**Sub-dominant types**	**Dist. limit**	**Size limit**
Unknown	Grass	Set(Forest, Vineyard)	n px	m px

Table 9: Rule to fill multi-bounded blobs, pixels and holes.

The rules specified in 9 specify for which areas a rule should apply. Often what is needed is to exclude a rule to be applied to a given surface. Such a case is when we are attempting to eliminate discontinuities between two areas, but we want to avoid considering a river as a discontinuity. To handle such situations, we need to provide a mask for the areas we wish to retain. Only the areas of interest get an appropriate type, others get UNDEFINED. Then we unite the results of the original and the temporary mask by copying the defined types from the later onto the former. Table 10 provides the details.

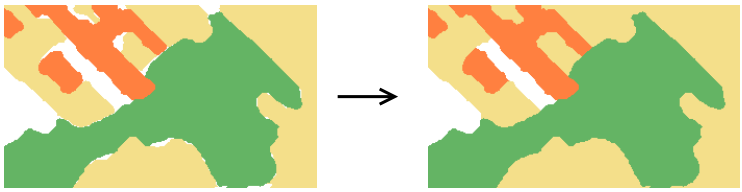

Figure 8: Result of using rules for filling multi-bounded blobs, pixels and holes.

While a human expert is able to manually vectorize map anomalies, like discontinuity and additional hidden information, the raw segmentation is unable to recognize them.

Surface type	**Type of dominant surface**	**Type of sub-dominant surface**
House	House	Grass

Table 10: A. Surface type selection. B. Combine grass and house surfaces, preferring houses to "draw" onto the grass layer.

	Surface	**Correct**	**False$^+$**	**False$^-$**	**Time**
Segmentation results	Field	88.75%	0.93%	11.25%	0ms
	Forest	84.59%	3.53%	15.44%	2460ms
	Vineyard	51.23%	443.01%	48.76%	2460ms
	Meadow	85.67%	1.87%	14.34%	3100ms
Reconstruction results	Field	98.17%	3.77%	1.83%	57ms
	Forest	96.96%	0.78%	3.04%	4927ms
	Vineyard	63.01%	25.93%	36.99%	5044ms
	Meadow	97.31%	1.19%	2.69%	8964ms

Table 11: Segmentation is done by the method described in section Recognition of Line-Like Symbols. Runtime and quality were measured on a topographic map of scale 1:10000. Results were compared to the result of the manual vectorization.

As it can be seen on Table 11, the reconstruction algorithms decreased the ratio of false positives, improved the vectorization accuracy and "recognized" some hidden information, which could be recognized earlier only by a human expert.

7 Recognition of Texturized Surfaces

Texturized surfaces are made up of repetition of smaller images, which are never rotated. The algorithm we described for the recognition of point-like symbols can be generalized (and at the same time specialized by removing the rotation steps) to simultaneously recognize multiple point-like symbols.

Figure 9: Texture, representing a region covered by scrub: a) texture, b) a texture tiled with subtextures showing the kernel.

It is assumed that, besides the point-like symbols, raster data for all textured surfaces, with the appropriate scale, are available and can be uniquely identified by their index. The algorithm of Section 4 will now be applied to recognize texture kernels instead of point-like objects. The modifications required are as follows:

1. Prepare a binary mask with the same size as the original map, and initialize all elements of it to FALSE.
2. Prepare a surface mask with the same size as the original map and initialize all elements of it to UNDEFINED. Each element of the surface mask is an index, which uniquely identifies a surface type or remains UNDEFINED.
3. Perform thinning on the map (for example Laplace).
4. Order the texture kernels according to their standard deviation, and call it TKA.
5. Process those pixels of the map which are currently unidentified (their index is UNDEFINED) in a top-down left to right fashion, which means performing the following steps:
   a) Assign TRUE to all elements of TKA (meaning no matching has been attempted).
   b) Calculate the standard deviation σ_c of the current map segment, that is the image starting at the pixel identified in step 5.
   c) In TKA find the closest match to σ_c, for which its boolean entry is TRUE.
   d) Match the two pieces of images within the given standard deviation treshold.
   e) If they match, assign them matching texture type to each element of the surface mask for the area under consideration, and set the corresponding elements of the binary mask to TRUE. Then continue from step 5.
   f) If there is no match, mark the texture kernel with FALSE and continue from step 5.c.

On the edges of textured surfaces, the kernels are usually not complete. In such cases, the algorithm as it stands is not capable of successful recognition. Those fractions can usually be identified by surrounding texture rectangles.

8 Conclusion

In this article, the results of the IRIS projects are discussed. The project aims to automate and support the recognition of raster images of topographic maps, with the combination of digital image processing and a knowledge-based approach. The developed system contains the basic knowledge of the classification of symbols to point-like, line-like and surface-like. From the point of recognition, there is an algorithmic difference between the recognition of homogeneous surfaces and textured surfaces.

The robustness of IRIS is based on the insight that layers of symbols are recognized (and "removed") in reverse order of their printing. The interpretation of line-like symbols is the most difficult, and it can not be surprising that the task requires the heaviest mathematical machinery. EF graphs are used for the recognition of curved, line-like objects with a systematic pattern.

Rules of knowledge-based systems also appear in IRIS, and numerous rules are used for the recognition of homogeneous surfaces.

It is widely accepted, that knowledge-based systems do not match the quality of human experts, yet, there are areas where automation is desirable and produces better results. Within its own limits a knowledge-based system is reliable, it is neither superficial nor does it makes mistakes. Both of these qualities have exceptional value in the case of maps with rich structure. While human interpretation and manual vectorization are error-prone, an automated system can process vast amount of details.

Experiences with IRIS clearly demonstrates that human experts will always be required, yet it is an exciting challenge to decrease the amount of repetitive, and error-prone tasks.

While most point-like objects are recognized ($> 99\%$), some works remain to eliminate the false positives. We experienced the same difficulties when we tried to recognize the texturized surfaces. We have shown also a feature recognition method to line-like symbols. We found that these features could be recognized well ($> 80\%$), which speeds up the later manual postprocessing. It removes the recognized features and assigns the attributes of the feature to the corresponding object. Furthermore, we developed a method to deal with the hidden surface objects. Formerly, these objects could be identified only by a human expert. As it can be seen on Table 11, our knowledge-based method is able to make deductions by predefined rules to refine the segmentation results and recognize surface discontinuities and hidden surface regions.

We tested our methods on hungarian topographic maps at scale 1:10000. We have scanned the maps at 300dpi resolution, which produced ca. 100 megapixels bitmaps. The recognition time for each point-like symbol was 3-4 seconds, while the recognition of the line-like object features on the whole map took 15-18 seconds. The surface reconstruction took 5-6 seconds for each surface layer. The total vectorization time was less than 2 minutes, compared to a vectorization made by a human expert, which takes many hours.

References

[1] Baik, S., et al. A naive geography analyst system with cognitive support of imagery exploitation. In Monroy, R., et al., editor, *Lecture Notes in Artificial Intelligence*, number 2974, pages 40–48, 2004.

[2] Buttenfield, B. P. and McMaster, R. B., editors. *Map generalization: Making rules for knowledge representation.* Longman Scientific & Technical, 1991.

[3] Chen, H., et al. A geographic knowledge representation system for multimedia geospatial retrieval and analysis. *International Journal on Digital Libraries*, 1(2):132–152, September 1997.

[4] Corner, R. J. *Knowledge Representation in Geographic Information Systems.* PhD thesis, Curtin University of Technology, December 1999.

[5] Dezső, B., Elek, I., and Máriás, Zs. IRIS, Development of Automatized Raster-Vector Conversion System. Technical report, Eötvös Loránd University and IKKK, November 2007. in Hungarian.

[6] Dezső, B., Elek, I., and Máriás, Zs. Image processing methods in raster-vector conversion of topographic maps. In Karras, A. D., et al., editor, *Proceedings of the 2009 International Conference on Artificial Intelligence and Pattern Recognition*, pages 83–86, July 2009.

[7] Janssen, R. D. T. and Vossepoel, A. M. Adaptive vectorization of line drawing images. *Computer Vision and Image Understanding*, 65(1):38–56, January 1997.

[8] Klinghammer, I. and Papp-Váry, Á. *Földünk tükre a térkép (Map, mirror of the Earth).* Gondolat, 1983.

[9] Liang, SC. and Chen, WJ. Extraction of line feature in binary images. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E91A:1890–1897, August 2008. Issue 8.

[10] Samet, H. *The Design and Analysis of Spatial Data Structures.* Addison - Wesley, 1990.

[11] Szendrei, R., Fekete, I., and Elek, I. Texture based recognition of topographic map symbols. In Karras, A. D., et al., editor, *Proceedings of the 2009 International Conference on Artificial Intelligence and Pattern Recognition*, pages 7–10, July 2009.

[12] Tsai, D. and Tsai, Y. Rotation-invariant pattern matching with color ring-projection. *Pattern Recognition*, 35(1):131–141, January 2002.