Optical Graph Recognition

Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, and Josef Reislhuber

University of Passau, 94030 Passau, Germany
{auerc,bachmaier,brandenb,gleissner,reislhuber}@fim.uni-passau.de

Abstract. Optical graph recognition (OGR) reverses graph drawing. A drawing transforms the topological structure of a graph into a graphical representation. Primarily, it maps vertices to points and displays them by icons and it maps edges to Jordan curves connecting the endpoints. OGR transforms the digital image of a drawn graph into its topological structure. It consists of four phases, preprocessing, segmentation, topology recognition, and postprocessing. OGR is based on established digital image processing techniques. Its novelty is the topology recognition where the edges are recognized with emphasis on the attachment to their vertices and on edge crossings.

Our prototypical implementation OGR$^{\text{up}}$ shows the effectiveness of the approach and produces a GraphML file which can be used for further algorithmic studies and graph drawing tools.

1 Introduction

Graph drawing addresses the problem of constructing visualizations of graphs, networks and related structures. It adds geometric and graphic information, assigning coordinates to the vertices and routing the edges, and adding graphic features such as icons, line styles and colors. The goal is a "nice" drawing, which shall convey the underlying structural relations and make it easily understandable to a human user. "Nice" may be evaluated empirically and approximated by formal terms, such as bends, crossings, angular resolution, and uniform distributions, e. g., see [10]. This is what the field of graph drawing is all about.

The reverse process has been disregarded so far. There is a need for it. We often make drafts of a diagram using pencil and paper and then would like to use a graph drawing tool for improvements of the drawing or a graph algorithm for an analysis. Here optical graph recognition comes into play. One needs a tool to convert the image of a drawn graph into the topological structure.

In this paper, we propose *optical graph recognition* (*OGR*) as a method to automatically extract the topological structure of a graph from its drawing. OGR is an adaption of optical character recognition (OCR) [3], which extracts plain text from images for automatic processing. Since there is a large variety, we restrict ourselves to the most common drawing types. The vertices are represented by geometric objects such as circles or rectangles, which are connected by (images of) Jordan curves representing the edges. The drawing is given as a (digital) image. An example of such a drawing if given in Fig. 2. OGR proceeds in four phases: preprocessing, segmentation, topology recognition, and postprocessing. The core of OGR is the topology recognition. This phase takes an image as input, where all pixels are classified as either background, vertex, or edge pixels. In the image, the regions of vertex pixels of two vertices are connected by a contiguous region of edge pixels if the two vertices are connected by an edge. However, the converse is not true. A contiguous region of edge pixels corresponds to several edges if the edges cross, see Fig. 2 and the left side of Fig. 6. The problem of crossing edges does not occur if the drawing is plane. In fact, an approach that extracts the circuit from its plane drawing is given in [4, p. 476], which is similar to our approach in the first two phases. However, crossings are unavoidable. The topology recognition resolves crossings similar to the human eye. The eye follows the edge curve to the crossing and then proceeds in the "most likely" direction, which is the same direction in which the edge curve enters the crossing.

Due to this similarity, OGR's recognition rate can be used as a measure of the legibility of a drawing. OGR is error-prone if many edges cross at the same point or if edges cross in small angles. Such drawings are hardly legible for humans as well [15,17]. Recently, Pach [19] has defined *unambiguous bold drawings*, which leave no room for different interpretations of the topology of the graph. For example, in unambiguous drawings areas of overlapping edges do not hide vertices. In fact, OGR presumes a unambiguous bold drawing as input.

The problem of automatically recognizing objects has been studied extensively in the field of digital image processing. Most prominently, optical character recognition has significantly advanced in the last decades [3]. However, the emphasis behind OCR is on recognizing the shape of a certain character and not its topological structure as with OGR. In [7,18], the authors have proposed methods to trace blood vessels and measure their size in X-ray images. Again, these approaches are designed to evaluate the shape of the blood vessels and ignore their topological structure.

Our paper is organized as follows. In Sect. 2, we give some preliminaries. The four phases approach of OGR is presented in Sect. 3 with an emphasis on the topology recognition phase in Sect. 3.3. An experimental evaluation is given in Sect. 4, where we discuss features of graph drawings which improve OGR's probability of a correct recognition and features which reduce the probability.

2 Preliminaries

In this paper, we deal with undirected graphs $G = (V, E)$. Directions of edges can be recognized in a postprocessing phase of OGR. In the following, a *drawing* of G maps vertices to graphical objects like discs, rectangles, or other shapes in the plane. The edges are mapped to Jordan curves connecting its endpoints. For convenience, we speak of vertices and edges when they are elements of a graph, of its drawing, and in a digital image of the drawing. A *port* is the point of an edge that meets the vertex. Every edge has exactly two ports. Note that this is only true if no edge crosses a vertex, where it is also hard for a human to recognize the correct adjacency. Hence, we assume no edge-vertex-crossings in the following. An *(edge-edge-)crossing* is a point where two or more edges cross.

A *digital image* is a set of pixels. Each pixel p has coordinates (x, y) in a two-dimensional grid of a certain width and height, and a *color* which is taken from a finite set $\mathbf{C}$. In a *binary image* only two colors are allowed, i. e., $\mathbf{C} = \{0, 1\}$, where a pixel with color 0 is a *background pixel* (black) and a pixel with color 1 is an *object pixel* (white). When a (color) image is converted into a binary image, the result is called *binarized image*. The *4-neighborhood* $N_4(x, y)$ of a pixel (x, y) consists of $(x - 1, y)$, $(x + 1, y)$, $(x, y - 1)$, and $(x, y + 1)$. Two pixels p and q of the same color are *4-adjacent* if they are 4-neighbors. A *4-path* from pixel (v, w) to pixel (x, y) is a sequence of distinct pixels $(x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k)$, where $(x_0, y_0) = (v, w)$, $(x_k, y_k) = (x, y)$ and pixels (x_i, y_i) and (x_{i-1}, y_{i-1}) are 4-adjacent for $1 \leq i \leq k$. A subset of pixels R is called *4-region* if there is a 4-path between every pixel $p \in R$ and $q \in R$ such that R is maximal. The *8-neighborhood* $N_8(x, y)$ of (x, y) consists of its neighbors $(x \pm 1, y \pm 1)$. 8-adjacency, 8-path, and 8-region are defined analogously.

We use *morphological image processing* to alter or analyze binary images. For example, *erosion* converts each object pixel with at least one background pixel in its 8-neighborhood into a background pixel, see Fig. 1. Similarly, *dilatation* converts each background pixel with at least one object pixel in its 8-neighborhood into an object pixel.

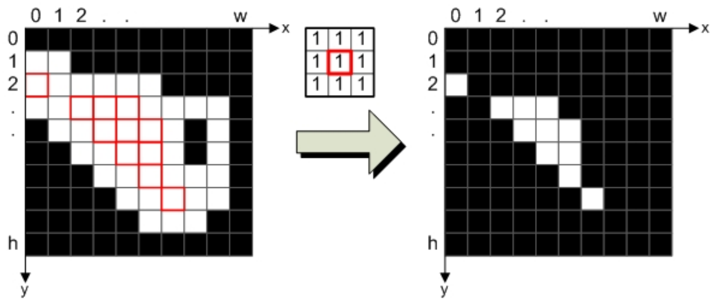


Fig. 1. Erosion: The reference pixel (center) of the pattern (above the arrow) is put on every pixel of the image and if the pattern and the pixels underneath do not match, the pixel underneath the center is turned into a background pixel.

3 Optical Graph Recognition

OGR is divided into the four phases: preprocessing, segmentation, topology recognition, and postprocessing. The input of the first phase is a drawing of a graph G as a digital image. From an information theoretic point of view, every phase reduces the information contained in the digital image, until only the sets of vertices and edges remain. A digital image with a size of several MB is reduced to a GraphML file of only a few KB. Each of the following sections is devoted to a phase, its purpose, suggestions for possible algorithms, and a description of our prototypical OGR implementation OGR$^{\text{up}}$. All phases but the topology recognition use standard image processing techniques for which we only show their effects. The reader is referred to standard literature on image processing for details, e. g., [14]. A detailed description of each phase can be found in [20]. The particularity is the topology recognition phase, which is therefore described in more detail as it involves non-standard techniques developed for the purpose of OGR.

3.1 Preprocessing

The purpose of the preprocessing phase is to separate the pixels belonging to the background from the pixels of the drawn graph, i. e., the image is binarized such that every object pixel is part of the drawing and every background pixel is not. Information that is unimportant to OGR, like the color of the vertices (edges) and the background, is removed from the image. This can be achieved with any binarization algorithm like *global*, *adaptive* or *hysteresis thresholding* [6, 7, 9, 13, 14]. The extent of information that is filtered depends both on the drawing of the graph and the tasks of the subsequent phases of OGR. For example, if the image contains vertex or edge labels, OCR [3] can be applied to identify them. Then, the labels are removed from the image such that they later on are not erroneously identified as part of the graph, and are passed to the postprocessing phase to assign them to vertices and edges.

In OGR$^{\text{up}}$, we use *histogram based global thresholding* for the binarization. With this method, each pixel with a color (gray value) greater than a predefined threshold is an object pixel and it is a background pixel otherwise. The threshold color t can either be manually defined, or it is automatically estimated by using the gray-level histogram [14, p. 599]. Fig. 2 shows the effect of binarization.

After binarization, we additionally apply the noise reduction method from [14, p. 531] depending on the quality of the image. There are two types of noise. Isolated object pixels (white) called *salt* and isolated background pixels (black) called *pepper*. Both types of noise can be reduced by the *opening* and *closing* operators. The opening operator first erodes k times and then dilates k times; closing does the same in inverse order. Opening generally smoothens the border of a region of object pixels and eliminates thin protrusions and, thus, salt [14, p. 528]. Closing also smoothens the border of object pixel regions but, in contrast to opening, removes pepper. Closing also fuses narrow breaks, eliminates small holes, and fills gaps in the border of object pixel regions. This is important in the context of OGR. For instance, if an edge curve is not contiguous or there is a gap between the edge curve and one of its attached vertices due to bad image quality, the edge cannot be recognized. Closing makes edges contiguous and fills small gaps between edges and vertices.

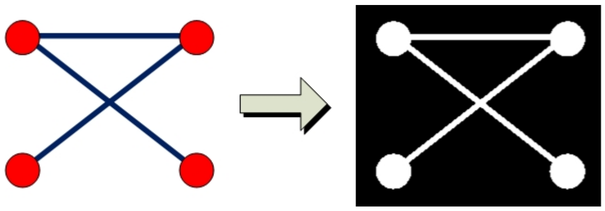


Fig. 2. A drawing of a graph with the result of the preprocessing phase

3.2 Segmentation

The input of the segmentation phase is a binarized image resulting from the preprocessing. In a nutshell, segmentation identifies the vertices in the binary image. More precisely, for each object pixel it determines whether it belongs to a vertex or to an edge. The output is a ternary image with three colors, for the background, the vertex, and edge pixels. Note that, depending on the shape of the vertices, different methods have to be applied.

In OGR$^{\mathrm{up}}$, we have implemented a generic approach inspired from [4, p. 476] which assumes the following preconditions. The vertices are represented by filled shapes, e. g., circles or rectangles, and the edges are represented by curves of a width significantly smaller than the diameter of the vertices, see Fig. 3. Using this assumption, we can use the opening operator which first erodes k times and then dilatates k times. Erosion shrinks regions of objects pixels by turning pixels at the border to background pixels. We choose k large enough such that all edge curves vanish, see Fig. 3. By assumption, the remaining object pixel regions belong to vertices. Since the regions occupied by the vertices have shrunk due to the erosion, applying dilation k times inflates vertices to their prior size. By comparing the object pixels after these operations with the binary image from the input of this phase, the desired ternary image is obtained.

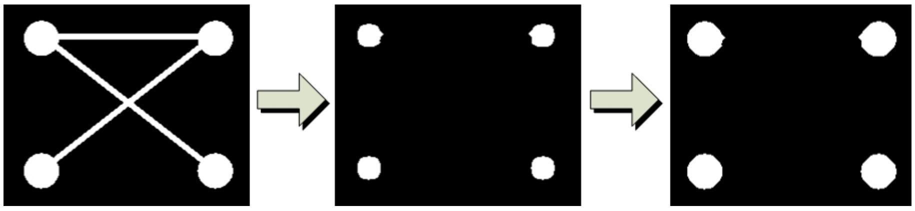


Fig. 3. A drawing of a graph with the result of the segmentation phase, input, after erosion, and after dilatation

The number k of erosions and dilatations can either be chosen manually or automatically with the help of the *distance image* obtained by the Chamfer algorithm [2] as implemented in OGR$^{\mathrm{up}}$. The distance image gives the minimum distance to the next background pixel for each object pixel. Large local maxima in the distance image can be found in the vertices' centers (cf. [4, p. 477]); we denote by $k_{\max}$ the smallest such maximum. In contrast, the local maxima belonging to edges, are small in comparison to the maxima found in vertices; we denote by $k_{\min}$ the largest such local maximum. Note that $k_{\min} \ll k_{\max}$ by assumption, which makes it possible to accurately determine $k_{\min}$ and $k_{\max}$ automatically. If k is chosen such that $k_{\min} < k < k_{\max}$, then the k erosions removes all edge curves but not the vertices.

If vertices are drawn as unfilled circles, the centers of the vertices can be detected by the *Hough transformation* [5,8,12]. Afterwards, the vertices can be filled and the approach described before can be used to detect the vertices. Other vertex shapes, filled or unfilled, need different approaches. Here, we can give no approach that works in any case but instead refer the reader to the rich set of algorithms for object detection [4,9,14].

3.3 Topology Recognition

The input for topology recognition is the binarized image and the results from the segmentation phase. Our approach can be divided into three subphases: *skeletonization*, *edge classification* and *edge traversal*. A detailed description in pseudocode can be found in [20].

The basic idea of skeletonization is to discard redundant information while retaining only the topological information of the graph, i.e., the regions of object pixels that represent the graph are reduced in size until a further removal of a single object pixel would destroy the connectivity of the regions [9, p. 151]. Skeletonization results in the *skeleton* of a binary image, e. g., see Fig. 4. The skeleton, also known as the *medial axis*, is the locus of the centers of all circles that are tangent to the border of an object region at two or more disjoint points [4, p. 474]. An example for the skeleton of a region of object pixels is shown in Fig. 5. The most important property of a skeleton for OGR is that it conserves the connectivity of the original binary image, i. e., the skeleton itself is connected. For more information on skeletons consider [14, p. 543–545, 650–653] or [9, p. 151–163]. Skeletonization can be achieved by the morphological thinning operation from [14, p. 541], which basically turns object pixels at the borders of object pixel regions into background pixels until a further thinning would destroy the connectivity of the regions. A result of skeletonization by morphological thinning is shown in Fig. 4. Note that for our approach the skeleton needs to be a 4-region.

The edge classification subphase classifies the pixels of the skeleton of the image based on the 4-neighborhood of every pixel and the pixels recognized as vertex pixels in the segmentation phase. Let $n_o(p)$ be the number of object pixels in the 4-neighborhood of p. Then, there are the following four classes of object pixels. *Miscellaneous pixels* P_M with $n_o \leq 1$ generally result from lines in the skeleton that do not end at a vertex, e. g., due to noise in the image. Miscellaneous pixels are ignored, as they are not necessary for the described approach. *Edge pixels* P_E are pixels with $n_o = 2$ that lie on the skeleton of an

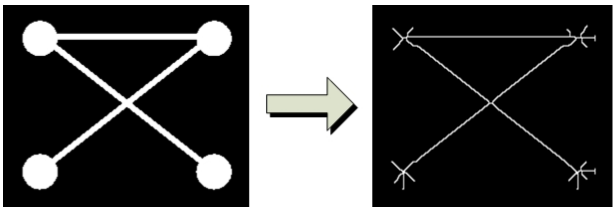


Fig. 4. A binary image after morphological thinning

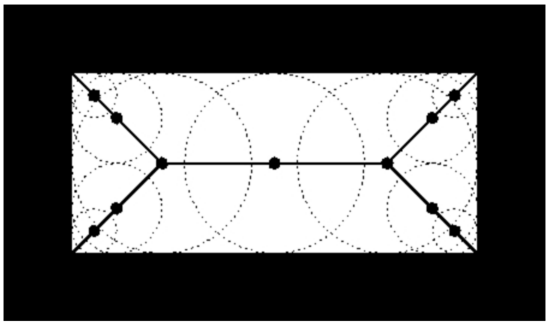


Fig. 5. Skeleton of a region of object pixels

edge. During the skeletonization phase each edge has been reduced to a line of single pixel thickness. Hence, every (inner) pixel that lies on the skeleton of an edge has exactly 2 object pixels in its 4-neighborhood. *Crossing pixels* P_C with $n_o > 2$ lie on the crossing of two or more skeletons of edges. If two lines in the skeleton cross, there is at least one pixel in the intersection that has more than 2 object pixels in its 4-neighborhood. *Vertex pixels* P_V are part of the skeleton of a vertex which already have been recognized in the segmentation phase. *Port pixels* P_P have a vertex pixel and an edge pixel in their 4-neighborhood, and thus, are the points where an edge meets a vertex.

The classification of pixels allows us to identify *edge sections* in the image. An edge section entirely consists of edge pixels up to both endpoints, which are in P_C or in P_P. Every edge section is a 4-region. Based on the two endpoints, we classify the sections in three categories as follows. In *trivial sections* both endpoints are port pixels in P_P, e. g., edge section “1” in Fig. 6. In *port sections* one endpoint is a port pixel in P_P and the other is a crossing pixel in P_C, e. g., edge sections “2”, “3”, “5”, and “6” in Fig. 6. Note that the simple strategy of counting ports to determine the number of edges usually fails as more than one edges may enter a vertex at the same port section. Finally, in *crossing sections* both endpoints are crossing pixels in P_C, e. g., section “4” in Fig. 6.

During our experiments with the edge classification it we observed that if two edges cross, the result was often a short intermediate crossing section like “4” in Fig. 6. This is problematic for the subsequent edge traversal subphase.

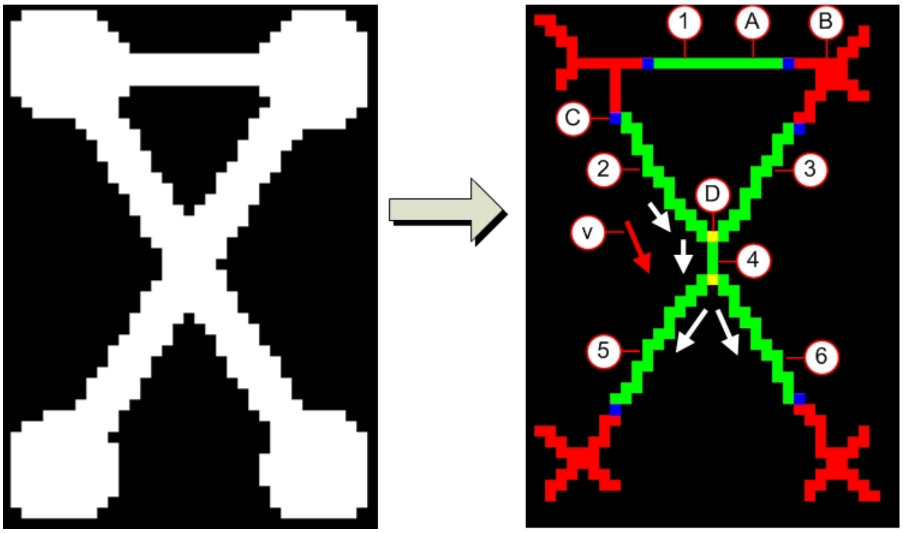


Fig. 6. A binary image of a graph and the result of edge classification. An edge pixel is marked with "A", a vertex pixel with "B", a port pixel with "C", and a crossing pixel with "D". The edge sections are marked with numbers and v is a direction vector.

Our solution is to first detect crossing sections like "4" in Fig. 6 and then to interpret them as part of a crossing. However, the recognition of such small crossing sections proved to be problematic and we have found no reliable way to distinguish crossing sections resulting from one crossing and crossing sections lying between two different crossings. Our simple approach is to interpret every crossing section of a size smaller than a predefined parameter as part of a crossing. Note that for Fig. 6 we did not apply this method for illustration purposes.

Trivial sections directly connect two vertices without interfering with any other edges. For every trivial section we directly obtain an edge, e. g., section "1" in Fig. 6. In contrast, port and crossing sections need a more elaborate treatment as these sections are caused by crossings. In the edge traversal phase, we merge port and crossing sections "adequately" to edges, e. g., in Fig. 6, sections "2", "4" and "6" are merged to one edge as well as "3", "4" and "5". We start the traversal always at a port section and traverse it until we find a crossing with another edge section. At this point we determine the adjacent edge section which most probably belongs to the current section using *direction vectors*. The direction vector of an edge section $e = (p_1, p_2, \ldots, p_l)$ is a two dimensional vector $\overrightarrow{p_i p_j}$, with $i \neq j$, $1 \leq i, j \leq l$, that describes the direction of e. p_i is the tail and p_j the head of the vector. Let (x_i, y_i) be the coordinates of p_i and (x_j, y_j) the coordinates of p_j. Then, $\overrightarrow{p_i p_j} = (x_j - x_i, y_j - y_i)$. $|i - j| + 1$ is the *magnitude* of $\overrightarrow{p_i p_j}$. An example is illustrated in Fig. 6, where we start at port section "2", reach crossing "D" and then compare the directions of port section "3" and crossing section "4" with the direction of "2". In this case "4" is chosen as its direction is most similar to the direction of “2”. It is important that the direction vectors used are not the direction vectors of the whole edge sections, i. e., direction vector “v” is not $\overrightarrow{CD}$, but the direction vector of the immediate area around the crossing. This is primarily necessary for graphs with non-straight edges. Here, an edge can be an arbitrary Jordan curve before it crosses another edge, and if the direction vector of the whole edge section to the crossing is used, then a false edge section may be chosen for the subsequent section of an edge. In our implementation we use direction vectors with an identical magnitude for all edge sections.

During skeletonization it may happen that the directions of edge sections in the vicinity of a crossing are distorted which can lead to false results. To avoid this problem, we do not choose the head of the direction vector directly at the crossing, but a few pixels away from the crossing when determining direction vectors. For example in Fig. 6, the head of the direction vector of “4” is not pixel “D”. In this example a distortion due to skeletonization does not occur.

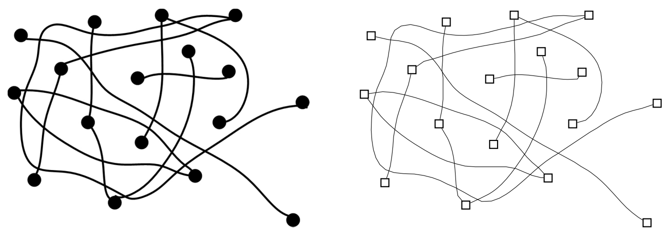
Continuing with the example from Fig. 6, we may determine an edge consisting of “2” followed by “4”. Then both sections “5” and “6” are suitable following sections when only considering the direction vector of “4”. To resolve this, we additionally take the direction vector of the preceding edge section (if existent) into account as indicated by direction vector “v” in Fig. 6. Let e_i be the edge section for which the subsequent section must determined, $\overrightarrow{pq}$ the direction vector of e_i, and e the current edge e_i is part of. Then, if e has more than one element, we take the predecessor e_{i-1} of e_i in e, determine the direction vector of e_{i-1} denoted by $\overrightarrow{rs}$ and compute the direction vector of e_i as $\overrightarrow{pq}' = \overrightarrow{pq} + \alpha * \overrightarrow{rs}$ with $0 \leq \alpha \leq 1$. The reason for the α weighting is to reduce the influence of $\overrightarrow{rs}$ on the final direction vector $\overrightarrow{pq}'$. Without the relaxation the effect is simply so large that it led to unwanted results. Due to this modification, section “6” is chosen as the succeeding section of “4” and as “6” is a port section, we have recognized an edge of the graph consisting of “2”, “4” and “6”. In the same way, the edge consisting of “3”, “4”, and “5” is recognized. With the edge consisting of “1” we now have recognized the topology of the input with four vertices and three edges.

3.4 Postprocessing

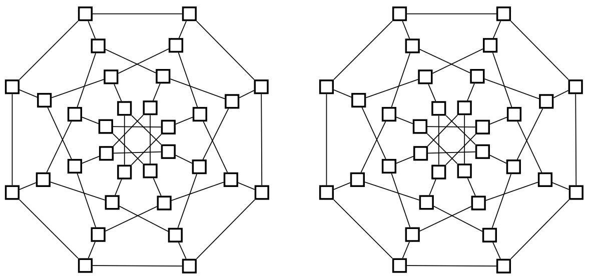
The postprocessing phase concludes OGR and includes procedures that use the topological structure as input. Possible tasks are the assignment of coordinates to the vertices obtained in the segmentation phase, the attachment of labels recognized in the preprocessing phase, the recognition of edge directions, the assignment of colors, and the transformation to file formats. In OGR$^{\text{up}}$, the postprocessing phase assigns coordinates to the vertices obtained in the segmentation phase and traverses every recognized edge and assigns bends to every edge so that they are drawn similar to the edge representations of the input image.

4 Experimental Results

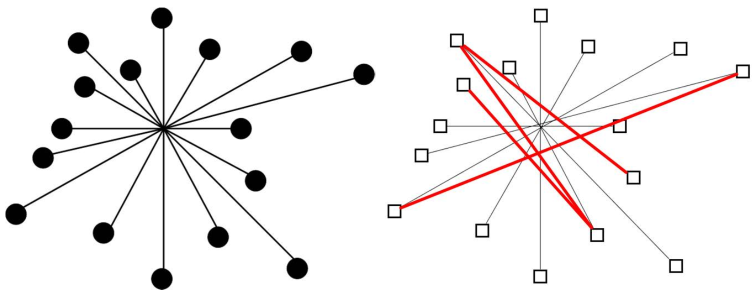
As a rule of thumb, the easier a graph can be recognized by a human, the better the graph can be recognized by OGR. Experiments with OGR$^{\text{up}}$ show



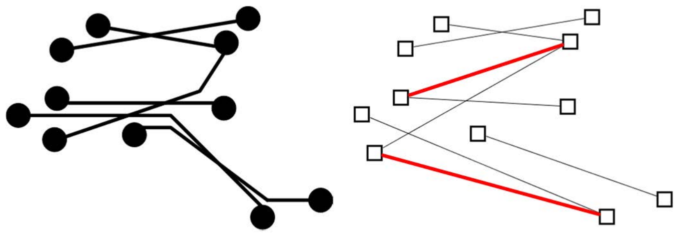
(a) Arbitrary Jordan curves as edges



(b) The Dyck graph



(c) Many crossings at the same point



(d) Small crossing angles

Fig. 7. Different drawings of graphs with the graph recognized by OGR^{up} viewed in Gravisto. The graphs in (a) and (b) are correctly recognized by OGR^{up}, whereas the graphs in (c) and (d) are not correctly recognized. Falsely recognized edges are shown bold and red.

that graph recognition is error-prone if many edges cross in a local area and if edges cross with small angles. This parallels recent empirical evaluations [15,17], which report that right angle crossings are as good as no crossings and have led to the introduction of RAC drawings [11].

As plane graphs have no crossings, they can be recognized very well by OGR. Drawings of 1-planar graphs [21], where each edge can have at most one crossing, preferably in a large angle, are also recognized well and the same holds for RAC-drawings. As already stated in Sect. 1, a unique topology can only be recognized if the input image supports it [19]. Otherwise, even a human has low chances to guess the intention of the creator. Fig. 7 gives exemplary graphs that were recognized by $\mathrm{OGR}^{\mathrm{up}}$. The drawing in Fig. 7(c) was not correctly recognized since too many edges cross at the same point. In Fig. 7(d) the small crossing angles of the edges lead to false results. However, both drawings are also hard to recognize for a human. As it is common to most digital image processing approaches, the results of $\mathrm{OGR}^{\mathrm{up}}$ heavily depend on the careful adjustment of the necessary parameters.

As a benchmark for $\mathrm{OGR}^{\mathrm{up}}$, we used the Rome graphs[1] (undirected graphs with 10 to 100 nodes). Each of the 11,534 Rome graphs was drawn 10 times with a spring embedder from Gravisto [1]. 107,543 drawings (93.24%) were recognized correctly. In 50 drawings, large regions of many crossing edges were erroneously recognized as vertices. In the remaining 7,747 drawings, the average number of false positives was 1.24 (variance 0.51), i. e., non-existent edges of the original graphs erroneously recognized by $\mathrm{OGR}^{\mathrm{up}}$ (cf. Fig. 7(d)). The average number of false negatives was 0.14 (variance 0.13), i. e., edges not recognized by $\mathrm{OGR}^{\mathrm{up}}$. The maximum number of false positives and false negatives was 10 and 5, respectively. 77.25% (75.96%) of all false positives (negatives) occurred in drawings with more than 75 vertices. The reason for this are many edge crossings in small areas and small crossing angles which frequently occur for larger graphs.

5 Summary and Perspectives

In Graph Drawing one is concerned with the construction of nice drawings. The drawings shall be well readable and there are many aesthetic criteria for the specification of "nice". Aesthetic criteria and graph visualizations are typically evaluated by comparing their differences in effectiveness, measured by task performance such as response time and accuracy. However, the approach has limitations by the individual performance and preferences [16]. Here, OGR may serve as an objective evaluator.

OGR is a framework to reverse the process of graph drawing. We have implemented a prototype of the framework which shows the usefulness of the approach and addresses problematic and error-prone tasks. Our tool $\mathrm{OGR}^{\mathrm{up}}$ is flexible enough for extensions and exchanges of concrete implementations of the phases, e. g., to introduce human interaction or to transfer it to a smartphone.

[1] `http://www.graphdrawing.org/data.html`

References

1. Bachmaier, C., Brandenburg, F.J., Forster, M., Holleis, P., Raitner, M.: *Gravisto*: Graph Visualization Toolkit. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 502–503. Springer, Heidelberg (2005), `http://gravisto.fim.uni-passau.de/`
2. Borgefors, G.: Distance transformations in digital images. Computer Vision, Graphics and Image Processing 34(3), 344–371 (1986)
3. Bunke, H., Wang, P.S.: Handbook of Character Recognition and Document Image Analysis. World Scientific (1997)
4. Castleman, K.R.: Digital Image Processing. Prentice-Hall (1996)
5. Cauchie, J., Fioletb, V., Villers, D.: Optimization of an Hough transform algorithm for the search of a center. Pattern Recogn. 41, 567–574 (2008)
6. Chow, C.K., Kaneko, T.: Automatic boundary detection of the left ventricle from cineangiograms. Comput. Biomed. Res. 5(4), 388–410 (1972)
7. Condurache, A.P., Aach, T.: Vessel segmentation in angiograms using hysteresis thresholding. In: IAPR Conference on Machine Vision Applications 2005, pp. 269–272 (2005)
8. Davies, E.R.: A modified Hough scheme for general circle location. Pattern Recogn. Lett. 7(1), 37–43 (1988)
9. Davies, E.R.: Machine Vision: Theory, Algorithms, Practicalities, 2nd edn. Academic Press (1997)
10. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall (1999)
11. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. Theor. Comput. Sci. 412(39), 5156–5166 (2011)
12. Duda, R.O., Hart, P.E.: Use of the Hough transformation to detect lines and curves in pictures. Commun. ACM 15(1), 11–15 (1972)
13. Estrada, R., Tomasi, C.: Manuscript bleed-through removal via hysteresis thresholding. In: Proc. International Conference on Document Analysis and Recognition, ICDAR 2009. IEEE (2009)
14. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 2nd edn. Prentice-Hall (2002)
15. Huang, W., Eades, P., Hong, S.H.: Beyond time and error: A cognitive approach to the evaluation of graph drawings. In: Proc. Beyond Time and Errors: Novel Evaluation Methods for Information Visualization, BELIV 2008, pp. 3:1–3:8. ACM (2008)
16. Huang, W., Eades, P., Hong, S.H.: Measuring effectiveness of graph visualizations: A cognitive load perspective. Inform. Visual. 8(3), 139–152 (2009)
17. Huang, W., Hong, S.H., Eades, P.: Effects of crossing angles. In: Fujishiro, I., Li, H., Ma, K.L. (eds.) Proc. IEEE Pacific Visualization Symposium, PacificVis 2008, pp. 41–46. IEEE (2008)
18. Lauren, V., Pisinger, G.: Automated analysis of vessel diameters in MR images. Visualization, Imaging, and Image Processing, 931–936 (2004)
19. Pach, J.: Every Graph Admits an Unambiguous Bold Drawing. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 332–342. Springer, Heidelberg (2012)
20. Reislhuber, J.: Graph Recognition. Master's thesis, University of Passau (2011), `http://www.infosun.fim.uni-passau.de/br/publications/mt-reislhuber-2011.pdf`
21. Ringel, G.: Ein Sechsfarbenproblem auf der Kugel. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 292, 107–117 (1965)