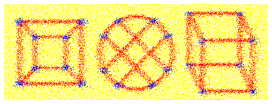

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 17, no. 4, pp. 541–565 (2013)
DOI: 10.7155/jgaa.00303

Optical Graph Recognition

Christopher Auer[1] *Christian Bachmaier*[1] *Franz J. Brandenburg*[1]
Andreas Gleißner[1] *Josef Reislhuber*[1]

[1]University of Passau, 94030 Passau, Germany

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. It has been tested both on hand-made graph drawings and on drawings generated by graph drawing algorithms.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
December 2012	February 2013	March 2013	April 2013	April 2013
		Published: July 2013		
	Article type: Regular paper	Communicated by: W. Didimo and M. Patrignani		

A preliminary version [2] appeared in the 20^{th} Symposium on Graph Drawing GD 2012.

E-mail addresses: auerc@fim.uni-passau.de (Christopher Auer) bachmaier@fim.uni-passau.de (Christian Bachmaier) brandenb@fim.uni-passau.de (Franz J. Brandenburg) gleissner@fim.uni-passau.de (Andreas Gleißner) reislhuber@fim.uni-passau.de (Josef Reislhuber)

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 attaches graphic features such as icons, line styles and colors. The goal is a "nice" drawing, which shall convey the underlying structural relations and make them easily understandable to a human user. "Nice" can be evaluated empirically and approximated by formal terms, such as bends, crossings, angular resolution, and uniform distribution [14]. This is what the field of graph drawing is all about.

The reverse process has been neglected 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. For instance, Fig. 11 shows a hand-made drawing of a graph that contains three multi-edges. Without the multi-edges, the graph is isomorphic to the Harries-Wong (3,10)-cage. However, for a human this is hard to verify. Here optical graph recognition comes into play. One needs a tool to convert the image of a drawn graph into the topological structure. At the 20^{th} Symposium on Graph Drawing (GD 2012) in Redmond, WA we have asked ten participants to draw a graph by hand. On the spot, our prototype for graph recognition $\mathrm{OGR}^{\mathrm{up}}$ correctly recognized five graphs. Two correctly recognized examples are shown in Figs. 9 and 10. Both graphs support our motivation for optical graph recognition as they have graph theoretic properties which are difficult to check for a human, but can be easily validated by an analysis tool. Figure 9 (by David Eppstein) shows a 5-dimensional hypercube which is missing four edges in the upper-leftmost part of the drawing and Fig. 10 (by Till Bruckdorfer) has multi-edges. One of the five drawings that were not correctly recognized, is the "Double Circulant" graph (by Steven Chaplick) in Fig. 12. $\mathrm{OGR}^{\mathrm{up}}$ recognized some edges as incident to a vertex, although they pass the vertex. A human will correctly recognize the graph using the context information "Double Circulant" and will infer that two neighboring vertices on the circle are not adjacent.

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) [7], 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, see Fig. 2. OGR proceeds in four phases: preprocessing, segmentation, topology recognition, and postprocessing. The topology recognition is the core of OGR. This phase takes a digital 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 part of Fig. 6. The problem of crossing edges does not occur if the drawing is plane. In this case there are approaches to recognize a planar graph [5] and to extract the circuit from its plane drawing [8, p. 476]. However, crossings are unavoidable. Our topology recognition resolves crossings similarly to the human eye. The eye follows the edge curve to the crossing and then proceeds in the "most likely" direction, which is the 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 [20,22]. Recently, Pach [24] 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 an 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 [7]. However, the emphasis behind OCR is to recognize the shape of a certain character rather than its topological structure as in OGR. In [11,23], 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. During the past decade an increasing number of research emerged at the intersection of pattern recognition and image analysis on one hand and graph theory on the other hand. In fact, the IAPR Workshop on Graph-based Representations in Pattern Recognition is devoted to this topic.

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 and we discuss "good" and "bad" features of a drawing for $\mathrm{OGR}^{\mathrm{up}}$'s recognition rate.

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 which meets the vertex. Each edge has exactly two ports. Note that this is only true if no edge crosses a vertex. In that case it is also hard for a human to recognize the adjacency correctly. Hence, we assume no edge-vertex-crossings in the following. An *(edge-edge-)crossing* is a point where two or more edges cross.

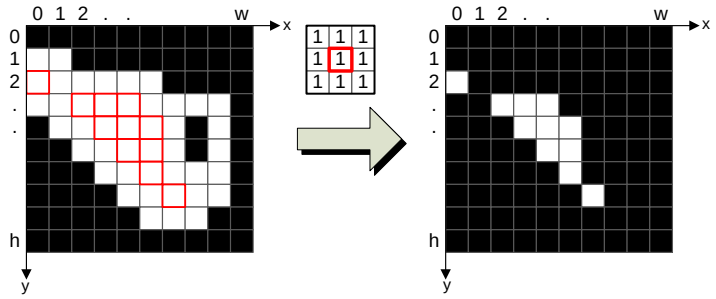


Figure 1: Erosion: The reference pixel (center) of the pattern (above the arrow) is put on every pixel of the image and if the pattern and the pixels underneath do not match, the pixel underneath the center is turned into a background pixel.

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-1}, y_{i-1}) and (x_i, y_i) are 4-adjacent for $1 \leq i \leq k$. A subset of pixels R is called a 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 $N_4(x, y)$ and additionally the pixels $(x \pm 1, y \pm 1)$. 8-adjacency, 8-path, and 8-region are defined analogously.

We use *morphological image processing* to alter or analyze binary images. Its basis is set theory. Each morphological operation relies on the same basic concept, which is to fit a predefined structuring element on every pixel of an image and to compare the set of pixels from the structuring element with the set of pixels that lies underneath the structuring element. A structuring element is a pattern of object and background pixels and is most commonly a square of size 3×3. 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.

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, the information contained in the digital image is reduced, until only the sets of vertices and edges remain. Thereby 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 implementation OGR$^{\text{up}}$. All phases but the topology recognition use standard image processing techniques for which we only demonstrate their effects. The reader is referred to the literature on image processing, e. g., [19]. The characteristic of our approach 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. The image is binarized such that every object pixel is part of the drawing and every background pixel is not. Information is removed from the image if it is unimportant to OGR, such as the color. This can be achieved with any binarization algorithm like *global*, *adaptive* or *hysteresis thresholding* [10, 11, 13, 17, 19]. The extent of information that is filtered depends both on the drawing of the graph and the tasks of the subsequent phases of OGR.

In OGR$^{\text{up}}$ we use *histogram based global thresholding* for the binarization [19, p. 599]. 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 set manually, or it is automatically estimated by using the gray-level histogram. Fig. 2 shows the effect of binarization.

After binarization, we additionally apply the noise reduction method from [19, 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 i times and then dilatates i 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 [19, p. 528]. Closing also smoothens the border of object pixel regions and 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 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. But the number of closings i must be chosen with care since too many closing operations may attach a passing edge to a non-incident vertex or may introduce touching edges, which are then classified as crossing edges. Experiments with values of $i > 1$ often lead to these undesired results, therefore we recommend using only a single closing operation. By a similar reasoning more than one opening operation is not advisable.

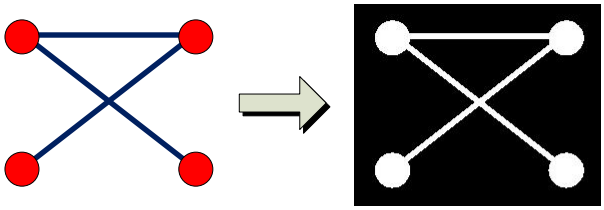


Figure 2: A drawing of a graph and the result of the preprocessing phase.

Figure 3: Vertex recognition in the segmentation phase: input, after erosion, and after dilatation.

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 background, vertex, and edge pixels. Note that, depending on the shape of the vertices, different methods have to be applied.

In OGR$^{\text{up}}$, we have implemented a generic approach inspired by [8, 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 by the erosions, applying dilation k times inflates vertices to their prior size. The desired ternary image is obtained by comparing the object pixels after these operations with the binary image from the input of this phase.

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 [6] as implemented in OGR$^{\text{up}}$. For each object pixel the distance image gives the minimum distance to the next background pixel. Large local maxima in the distance image can be found in the vertices' centers (cf. [8, p. 477]); we denote by $k_{\max}$ the smallest such maximum. In contrast, the local

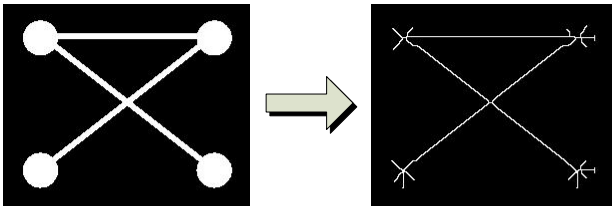


Figure 4: A binary image after morphological thinning.

maxima belonging to edges are small in comparison to the maxima found in vertices; we denote by $k_{\min}$ the largest such local maximum. By assumption $k_{\min} \ll k_{\max}$, 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 k erosions remove all edge curves but not the vertices.

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*.

The basic idea of skeletonization is to discard redundant information while retaining only the topological information of the graph. To this effect 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 [13, p. 151]. Skeletonization results in the *skeleton* of a binary image, as shown in 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 [8, p. 474]. A useful interpretation is to imagine that the region of object pixels of Fig. 5 consists of an inflammable material. This material is set on fire simultaneously on all points of its border. From each side the fire burns at the same speed and, eventually, the fires from opposing sides meet and extinguish [8, p.474]. The points where two fires meet are the points of the skeleton or to put it in another way, these points are the remains of the inflammable material that has not burned down. 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 preserves the connectivity of the original binary image. For more information on skeletons consider [19, p. 543–545, 650–653] or [13, p. 151–163]. There are several skeletonization algorithms from which we use the morphological thinning operation [19, p. 541]. It 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. We used skeletonization by morphological thinning to obtain the result as shown in Fig. 4. Note that the skeleton of each connected component of a graph must be a 4-region for our approach.

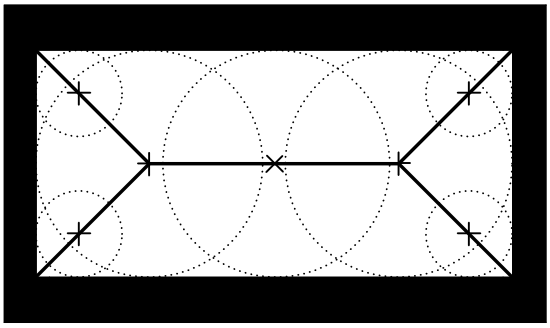


Figure 5: Skeleton of a rectangular vertex.

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. 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 relevant for the approach. *Edge pixels* P_E are pixels with $n_o = 2$ that lie on the skeleton of an edge. During the skeletonization phase each edge has been reduced to a line of single pixel thickness. Hence, every (inner) pixel on the skeleton of an edge has exactly two 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, then there is at least one pixel in the intersection that has more than two object pixels in its 4-neighborhood. *Vertex pixels* P_V are part of the skeleton of a vertex which have already 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 edge may enter a vertex at the same port section (see Fig. 8). Finally, in *crossing sections* both endpoints are crossing pixels in P_C, e. g., "4" in Fig. 6.

While experimenting with our edge classification approach we observed that the result was often a short intermediate crossing section if two edges cross, like "4" in Fig. 6. As we will see later, this is problematic for the subsequent edge traversal subphase. 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

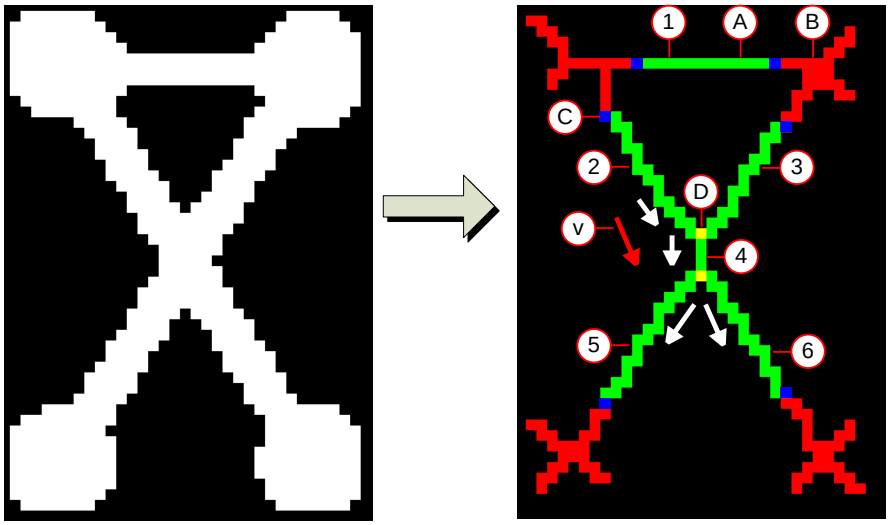


Figure 6: A binary image of a graph and the result of edge classification. An edge pixel is marked with "A", a vertex pixel with "B", a port pixel with "C", and a crossing pixel with "D". Edge sections are marked with numbers, v and the small arrows are direction vectors.

of such small crossing sections proved to be difficult. A crossing section may either result from a single crossing as "4" in Fig. 6 or may be caused by two entirely independent crossings as in Fig. 7. In this figure the crossing sections "a", "b", "c", and "d" are interpreted as four adjacent crossing sections and, hence, as a single crossing which is obviously not the case, as seen on the left side of the figure. The subsequent edge traversal subphase may recognize the edge consisting of sections "1" and "2", which is not existent in the original drawing. To avoid this problem, our approach is to interpret each crossing section of a size smaller than a predefined parameter as part of a single crossing. Depending on how thick the edges of a graph are drawn, values for the parameter between 5 and 15 pixels lead to suitable results in OGR$^{\text{up}}$ [1].

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. For example 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 pixel that is common to two or more edge sections. 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 $i \neq j$, $1 \leq i, j \leq l$, which defines the direction of e. p_i is the tail and p_j the head of the vector.

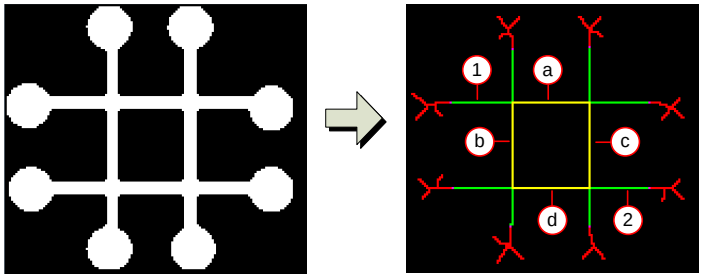


Figure 7: A binary image of a graph and the result of the edge classification subphase where every crossing section is interpreted as part of a crossing.

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, since its direction is most similar to the direction of "2". It is important that the direction vector of an edge section does not take the whole section into account. That is, the direction vector of "2" is not $\overrightarrow{CD}$ but computed only from the immediate area around the crossing. This is primarily necessary for graphs with non-straight edges, where the edges are arbitrary Jordan curves. Hence, when approaching an edge crossing, only the local direction must be used to determine the most likely subsequent edge section. In our implementation we use direction vectors with the same 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 "2" is not pixel "D".

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 follow-up 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, i. e., $v = \overrightarrow{4} + \alpha \cdot \overrightarrow{2}$. Let e_i be the edge section for which the subsequent section must be determined, $\overrightarrow{pq}$ the direction vector of e_i, and e the current edge of which e_i is part of. If e consists of more than one edge section, we take the predecessor e_{i-1} of e_i in e, determine the direction vector of e_{i-1} denoted by $\overrightarrow{op}$, and compute the direction vector of e_i as $\overrightarrow{pq}' = \overrightarrow{pq} + \alpha \cdot \overrightarrow{op}$ with $0 \leq \alpha \leq 1$. The reason for the α weighting is to reduce the influence of $\overrightarrow{op}$ on the final direction vector $\overrightarrow{pq}'$. The relaxation is to reduce the influence of prior edge sections to

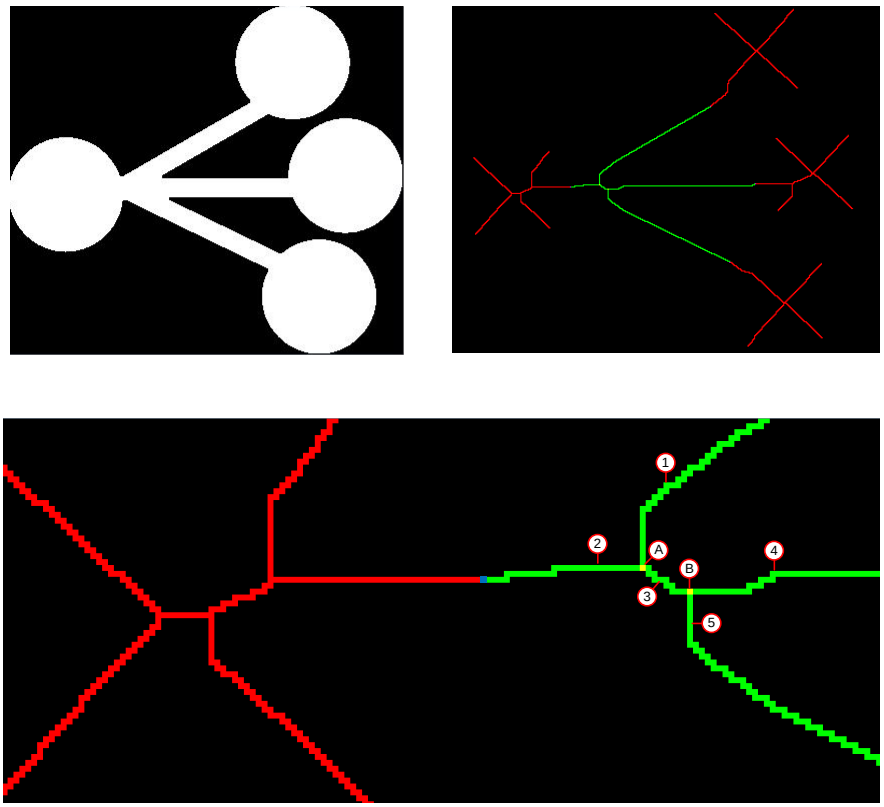


Figure 8: Binary image of a graph and the result of the edge classification subphase. In the lower image (zoomed to the left part of the skeleton) two recognized crossings "A" and "B" have an odd number of adjacent edge sections.

avoid unsolicited 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.

Note that the idea to use matching in a straight-forward manner to resolve crossings yields inferior results to the approach shown in this section. For example, in Fig. 8 there are crossings with an odd number of adjacent sections.

3.4 Postprocessing

The postprocessing phase concludes OGR and includes procedures that use the topological structure as input. In OGR$^{\text{up}}$, the postprocessing phase assigns

coordinates to the vertices obtained in the segmentation phase. Further, it assigns bends to every edge such that they resemble their counterparts from the original image. To obtain the coordinates of the bends, each edge is sampled every i pixels, where i is a user parameter, e. g., $i = 10$ leads to sufficiently smooth curves.

4 Experimental Results

Our prototype OGR$^{\text{up}}$ assumes the following preconditions. The vertices are drawn as filled shapes, e. g., circles or rectangles and all vertices have approximately the same size. The edges are represented by curves of a width significantly smaller than the diameter of the vertices, and they should be contiguous and should exactly end at the vertices.

As a rule of thumb, the easier a graph is be recognized by a human, the better the graph is be recognized by OGR$^{\text{up}}$. Our first experiments with OGR$^{\text{up}}$ show that graph recognition is error-prone if many edges cross in a local area and if edges cross in a small angle. This parallels recent empirical evaluations [20, 22], which report that right angle crossings are as legible as no crossings and have led to the introduction of RAC drawings [15].

Plane drawings of graphs have no crossings and are recognized very well by OGR$^{\text{up}}$. Graphs with few edge crossings, preferably in a large angle, are also recognized well and the same holds for RAC drawings. As already stated in Sect. 1, the correct topology can only be recognized if the input is unambiguous [24]. Otherwise, even a human must guess the intention of the graph drawer, e. g., Fig. 17. As for most digital image processing approaches, the results of OGR$^{\text{up}}$ heavily depend on the careful adjustment of the necessary parameters.

Experiments with OGR$^{\text{up}}$ so far include graphs drawn with pencil and paper as in Figs. 9–12, graphs drawn with an image editing tool as in Figs. 13–15, and graphs drawn with a graph drawing algorithm, for example with Gravisto [3], as in Fig. 16. The hand-made drawings in Figs. 9 and 10 show that OGR$^{\text{up}}$ can recognize freehand curves and achieves good results if our preconditions are met. For the drawing in Fig. 9 a proper adjustment of the parameters in the preprocessing phase was necessary to ensure that all edges are contiguous and end directly at the vertices. The graphs in Fig. 14 illustrate features of drawings that lead to a false recognition by OGR$^{\text{up}}$. In Fig. 14(a) the crossing region has approximately the size of a vertex. Hence, OGR$^{\text{up}}$ misinterprets the crossing as a vertex. In Figs 14(b-c) the small crossing angles lead to a false recognition. However, all three drawings are also hard to grasp for a human.

As a first benchmark for OGR$^{\text{up}}$, we used the Rome graphs[1] (undirected graphs with 10 to 100 nodes). Each of the 11534 Rome graphs was drawn 10 times in Gravisto [3], with a spring embedder applying the force model from Fruchterman and Reingold [18]. We used a slightly modified Fruchterman and Reingold force model, which also models repulsive forces between vertices and their non-incident edges to avoid crossings between them. 107543 drawings

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

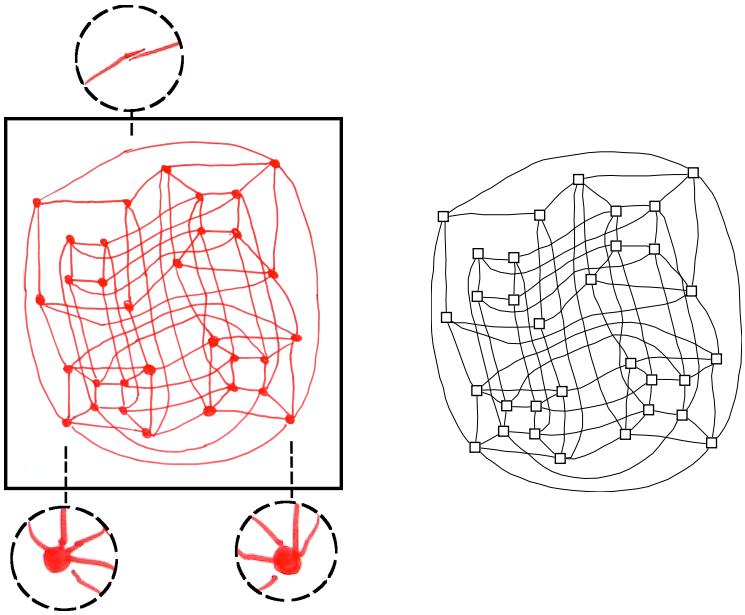


Figure 9: A drawing of a 5-dimensional hypercube "$Q5$" by David Eppstein is correctly recognized by OGR$^{\text{up}}$. In the drawing the topmost edge is not contiguous and the bottommost edge does not end at a vertex (shown magnified). This problem was resolved by applying opening and closing in the preprocessing phase.

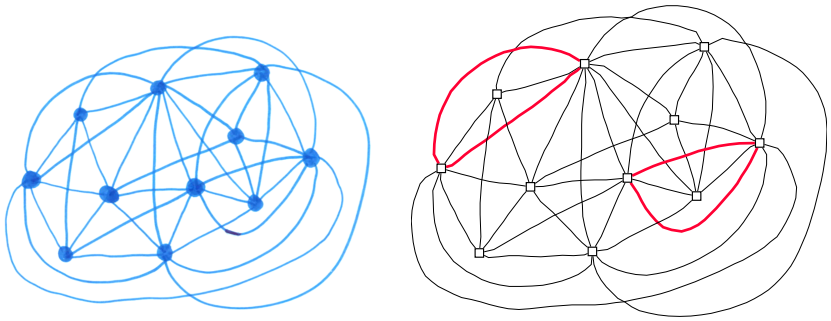


Figure 10: The "T_{11}^{*}" graph by Till Bruckdorfer is correctly recognized by OGR$^{\text{up}}$. Multi-edges are shown bold and red.

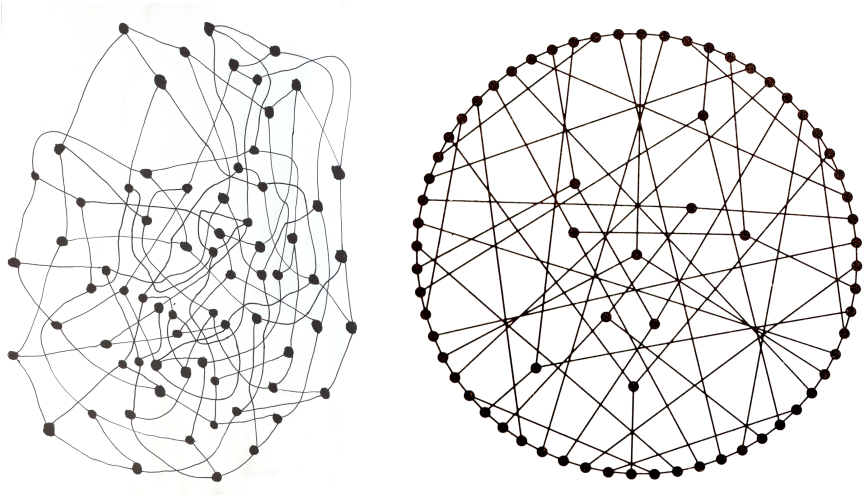


Figure 11: On the right side a drawing of the Harries-Wong (3,10)-cage from [25, p.272]. The hand drawn graph on the left side has three multi-edges. Without the multi-edges, the two graphs are isomorphic.

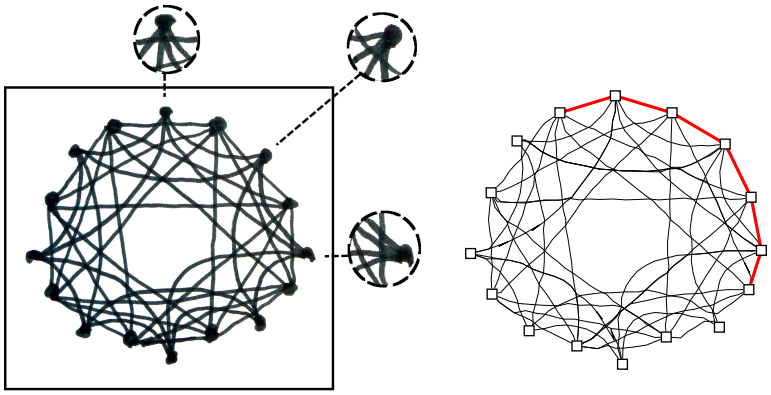
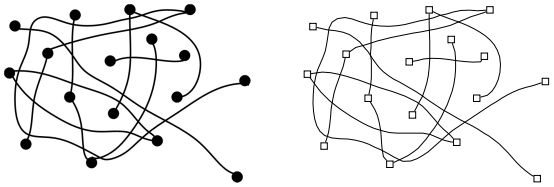
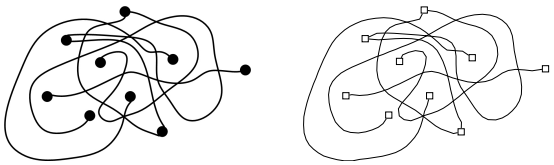


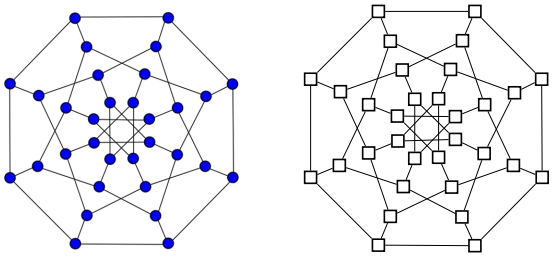
Figure 12: The "Double Circulant" graph by Steven Chaplick and the graph recognized by OGR^{up}. Some falsely recognized edges are shown bold and red. The parts of the drawing that led to the falsely recognized edges are shown magnified.



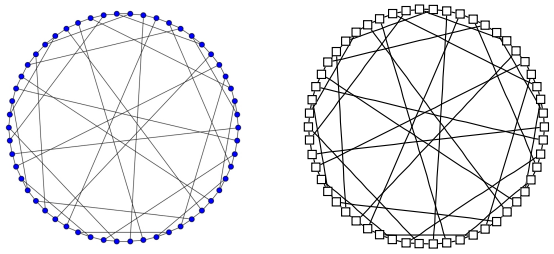
(a) Arbitrary Jordan curves as edges



(b) More engulfed Jordan curves as edges

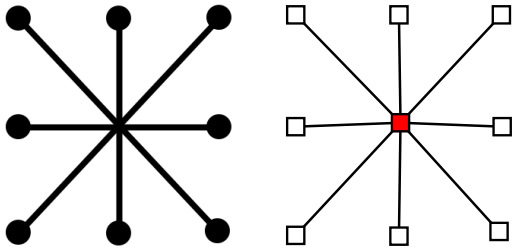


(c) The Dyck graph. Layout from [26]



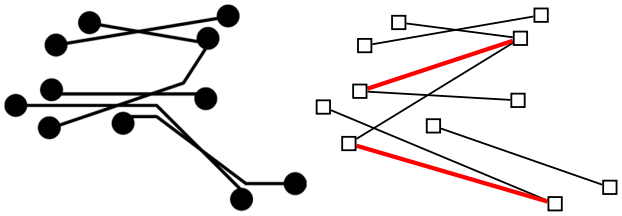
(d) The Gray Graph. Layout from [27]

Figure 13: Different drawings of graphs and the graph recognized by $\mathrm{OGR}^{\mathrm{up}}$. Each graph was correctly recognized.

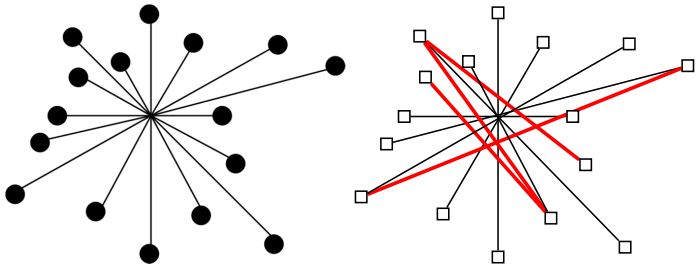


(a) Large area of crossing edges falsely recognized as a vertex

(b) Small crossing angles at a single point



(c) Small crossing angles



(d) Many crossings at the same point

Figure 14: Different drawings of graphs and the graph recognized by OGR^{up}. Falsely recognized edges and vertices are shown bold and red.

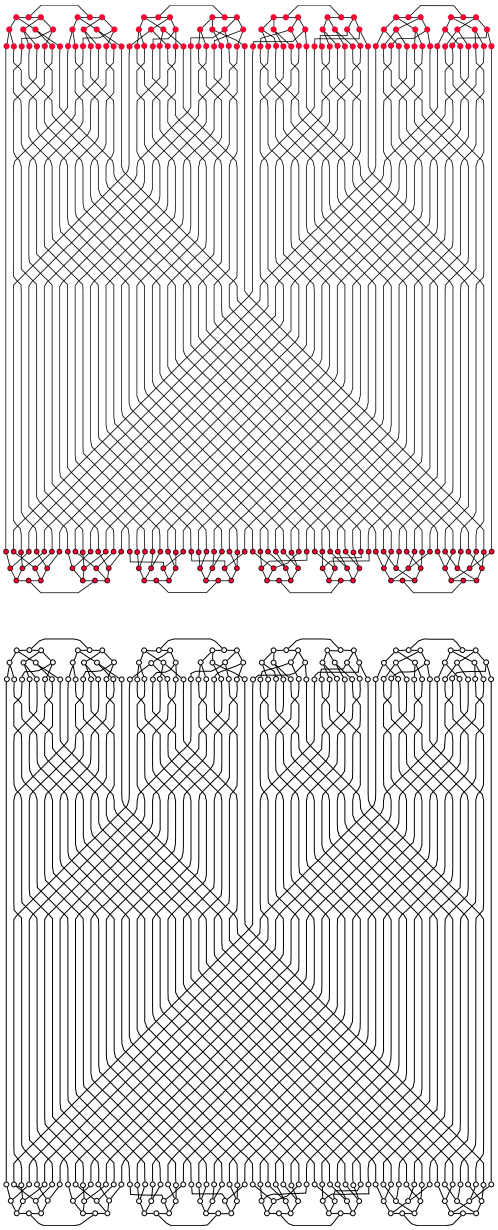


Figure 15: A drawing of the Balaban 11-cage with the layout from [4] which was correctly recognized by $\mathrm{OGR}^{\mathrm{up}}$.

(93.24%) were correctly recognized. In 50 drawings, large regions of many crossing edges were erroneously recognized as vertices.

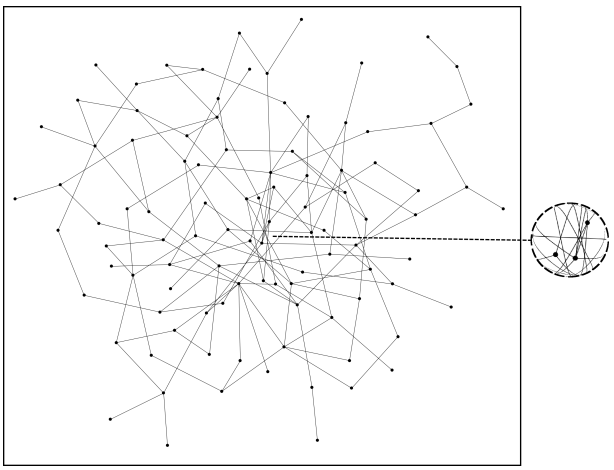
In the remaining 7747 drawings, the average number of false positives was 1.24 (variance 0.51), i. e., non-existent edges of the original graphs were erroneously recognized by OGR^{up} (cf. Fig. 14(d)). The average number of false negatives was 0.14 (variance 0.97), i. e., edges not recognized by OGR^{up}. The maximum number of false positives and false negatives in a graph was 10 and 5, respectively. 77.25% (75.96%) of all false positives (negatives) occurred in drawings with more than 75 vertices. This is caused by many edge crossings in small areas and small crossing angles, which frequently occur for larger graphs. Altogether, OGR^{up} resulted in 1069 false negatives, i. e., 99.99% of all 7965120 edges were recognized and the number of false positives was 9681.

In order to test a different graph drawing algorithm as a benchmark, the Rome graphs were drawn with the Lombardi spring embedder from [9]. In a Lombardi drawing [16] edges are drawn as circular arcs. It achieves the maximum angular resolution possible at each vertex. We ran separate benchmarks with the Tangent-Based and the Dummy-Vertex Lombardi spring embedders. The Tangent-Based approach is able to achieve near-perfect angular resolution at all nodes. However, a problem for OGR is that the algorithm often draws vertices on top of non-incident edges [9], which are then recognized as being incident, see Fig. 17. With the additional context information that the graph is a Lombardi-drawing, these falsely recognized edges can possibly be detected and corrected in the postprocessing phase. For instance, in this case the angular resolution is not (close to) maximal at the vertex. While the Dummy-Vertex approach is less successful in producing near-perfect angular resolution, it increases the distances between vertices and non-incident edges [9] and thus reduces the probability of a vertex being drawn on a non-incident edge.

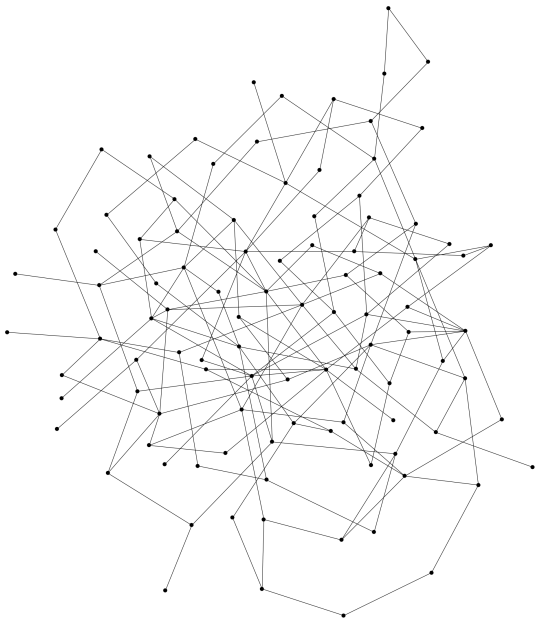
Every Rome graph was drawn three times with both Lombardi spring embedders. With the Tangent-Based approach, OGR^{up} recognized 9502 drawings (37.79%) correctly. In three drawings OGR^{up} failed to recognize the correct number of vertices. In the remaining 25097 drawings, the average number of false positives was 8.60 (variance 47.97) and the average number of false negatives was 3.27 (variance 6.62). The maximum number of false positives and false negatives in a graph was 58 and 22, respectively. The high number of false positives (negatives) results from drawings where vertices are drawn on edges. Here, OGR^{up} resulted in 82066 false negatives, i. e., 96.57% of all 2389536 edges were recognized and the number of false positives was 215840.

With the Dummy-Vertex approach, OGR^{up} correctly recognized 13884 drawings (40.12%). In 42 drawings OGR^{up} failed to recognize the correct number of vertices. In the remaining 20676 drawings, the average number of false positives was 6.16 (variance 27.50) and the average number of false negatives was 2.18 (variance 3.68). The maximum number of false positives and false negatives in a graph was 46 and 18, respectively. In total, OGR^{up} resulted in 45206 false negatives, i. e., 98.11% of all 2389536 edges were recognized and the number of false positives was 127810.

For the Lombardi spring embedders the recognition rate of OGR^{up} is inferior



(a) A sample Rome graph, which was not correctly recognized by OGR$^{\text{up}}$. The part of the drawing that led to a false recognition is shown magnified.



(b) This Rome graph was correctly recognized by OGR$^{\text{up}}$.

Figure 16: Two drawings of Rome graphs.

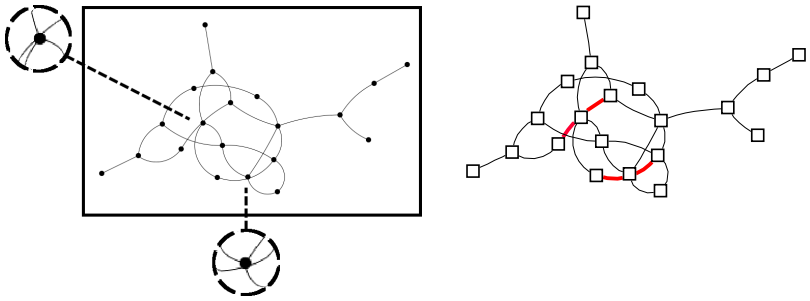


Figure 17: A Rome graph drawn with the Tangent-Based Lombardi spring embedder and the graph recognized by OGR$^{\text{up}}$. The magnified parts show where a vertex was drawn on a non-incident edge. The falsely recognized edges are shown bold and red.

as for the Fruchterman and Reingold spring embedder. However, one might use the context information that a drawing is in Lombardi-style. For instance, the information that the edges are drawn as circle segments, and the maximal angular resolution can be utilized to implement a different approach in the topology recognition phase of OGR. This likely increases the recognition rate for Lombardi-style drawings. Figure 18 gives an overview of the results of our benchmarks.

In all examples, the running time of OGR$^{\text{up}}$ was less than 10 seconds, however, there is room for improvements. The maximum resolution of the drawings was approximately 2600×2600 pixels (or 360kB png images).

5 Summary and Perspectives

OGR is a framework to reverse the process of graph drawing, i. e., to extract the topological structure of a graph from its drawing. To our best knowledge the approach is the first which permits edge crossings. Our prototype OGR$^{\text{up}}$ shows the usefulness of the approach and addresses problematic and error-prone tasks. Its modular architecture allows extensions and exchanges of specific implementations of all phases.

Currently, OGR$^{\text{up}}$ can recognize undirected graphs and presumes that the vertices are drawn as filled shapes. A necessary extension is the recognition of directed edges in the topology recognition phase. Therefore, we make use of miscellaneous pixels detected during edge classification to identify arrow heads and their directions. Further, we will tackle unfilled vertex shapes, e. g., unfilled circles or squares which is of particular interest for hand-drawn graphs as it is tedious to fill vertices. This can directly be achieved in the segmentation phase by using the *Hough Transformation* [12] or any of the rich set of algorithms for

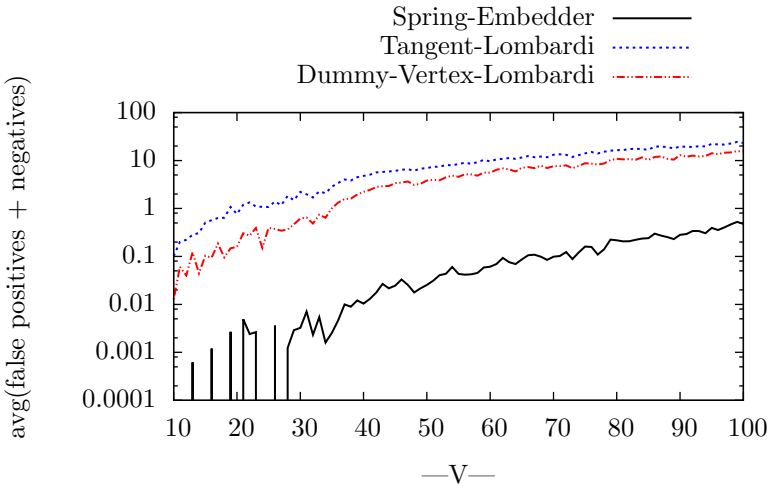


Figure 18: Graphical visualization of OGR$^{\text{up}}$'s recognition rate during our benchmarks with the Rome graphs. The $y-$axis is the sum of false positives and false negatives averaged over all graphs with the same number of vertices.

object detection [8, 13, 19].

An ongoing challenge is to improve the recognition rate of edges in the topology recognition phase. Particularly, we want to reduce the undesired results originating from many edge crossings within a small local area and from small crossing angles. For instance, we will test alternative skeletonization algorithms, since the current method often led to unwanted side effects. An even more promising idea is to use the gray-value image instead of the binary skeleton in the edge traversal subphase. Edge sections can then be merged based on the gradient [19]. Another common problem in hand-made drawings are gaps between edges and their end vertices.

In order to recognize vertex and edge labels we will use optical character recognition [7] in the preprocessing phase. Then, we remove the labels from the drawing such that they are not confused with parts of the graph. Finally, the labels are assigned to the vertices and edges in the postprocessing phase.

A promising future research direction is to incorporate context information into the recognition algorithm. For instance the knowledge about the used drawing algorithm or style, or type and properties of the graph may improve the recognition. OGR should also enrich the output by its confidence of the correctness of the result. For example, the variance of the vertex sizes and the sizes of the angle deviations while merging edge sections are useful indicators of the error-proneness of the algorithm. This information can be used to incorporate graphical user interaction into the algorithm.

The readability of graph visualizations is often evaluated by subjective weight-

ings of some aesthetic criteria. Hence, researchers had to resort to empirical studies measuring how fast and accurate a human subject group can recognize graphs. However, the approach is biased due to individual performance and preferences [21]. Here, OGR may serve as an objective evaluator. The recognition rate of OGR^{up} can be interpreted as a quality indicator. However, there are some features of drawings which pose no difficulties for OGR but do not meet common aesthetic criteria. Examples are hard to follow long edges like intertwined spirals or numerous crossings despite of large crossing angles, e. g., see Fig. 15.

Last but not least, we are working on a version of OGR^{up} that is compatible with smart phones and tablet computers. Then, the user can directly take a picture of a graph with the built-in camera and use OGR^{up} to recognize the graph for further processing and interaction.

Acknowledgments

We would like to thank David Eppstein, Till Bruckdorfer, Steven Chaplick and the other participants of the 20^{th} International Symposium on Graph Drawing for providing hand-drawn graphs. We are grateful to Stephen Kobourov, Roman Chernobelskiy and Kathryn Cunningham for providing their Tangent-Based and the Dummy-Vertex Lombardi spring embedder implementations. Also we would like to thank Peter Barth for discussions and useful hints on image processing.

References

[1] C. Auer, C. Bachmaier, F. J. Brandenburg, A. Gleißner, and J. Reislhuber. Calibration in optical graph recognition. *Imagen-A*, 3(5), 2013.

[2] C. Auer, C. Bachmaier, F. J. Brandenburg, A. Gleißner, and J. Reislhuber. Optical graph recognition. In W. Didimo and M. Patriganai, editors, *Graph Drawing*, LNCS, pages 529–540. Springer, 2013. doi:10.1007/978-3-642-36763-2_47.

[3] C. Bachmaier, F. J. Brandenburg, M. Forster, P. Holleis, and M. Raitner. Gravisto: Graph visualization toolkit. In J. Pach, editor, *Graph Drawing, GD 2004*, volume 3383 of *LNCS*, pages 502–503. Springer, 2004. doi:10.1007/978-3-540-31843-9_52.

[4] Balaban 11-cage alternative drawing, Nov 2012. http://commons.wikimedia.org/w/index.php?title=File:Balaban_11-cage_alternative_drawing.svg&page=1.

[5] S. Birk. Graph recognition from image. Master's thesis, University of Ljubljana, 2010. http://eprints.fri.uni-lj.si/1250/ (in Slovene).

[6] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34(3):344–371, 1986. doi:10.1016/S0734-189X(86)80047-0.

[7] H. Bunke and P. S. Wang. *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997. doi:10.1142/2757.

[8] K. R. Castleman. *Digital Image Processing*. Prentice-Hall, 1996.

[9] R. Chernobelskiy, K. I. Cunningham, M. T. Goodrich, S. G. Kobourov, and L. Trott. Force-directed Lombardi-style graph drawing. In M. van Kreveld and B. Speckmann, editors, *Graph Drawing, GD 2011*, volume 7034 of *LNCS*, pages 320–331. Springer, 2012. doi:10.1007/978-3-642-25878-7_31.

[10] C. K. Chow and T. Kaneko. Automatic boundary detection of the left ventricle from cineangiograms. *Comput. Biomed. Res.*, 5(4):388–410, 1972. doi:10.1016/0010-4809(72)90070-5.

[11] A.-P. Condurache and T. Aach. Vessel segmentation in angiograms using hysteresis thresholding. In *IAPR Conference on Machine Vision Applications 2005*, pages 269–272, 2005.

[12] E. R. Davies. A modified Hough scheme for general circle location. *Pattern Recogn. Lett.*, 7(1):37–43, 1988. doi:10.1016/0167-8655(88)90042-6.

[13] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, 2nd edition, 1997.

[14] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[15] W. Didimo, P. Eades, and G. Liotta. Drawing graphs with right angle crossings. *Theor. Comput. Sci.*, 412(39):5156–5166, 2011. doi:10.1016/j.tcs.2011.05.025.

[16] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, and M. Nöllenburg. Lombardi drawings of graphs. *J. Graph Alg. App.*, 16(2):85–108, 2010. doi:10.1007/978-3-642-18469-7_18.

[17] R. Estrada and C. Tomasi. Manuscript bleed-through removal via hysteresis thresholding. In *Proc. International Conference on Document Analysis and Recognition, ICDAR 2009*, pages 753–757. IEEE, 2009. doi:10.1109/ICDAR.2009.88.

[18] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. doi:10.1002/spe.4380211102.

[19] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice-Hall, 2nd edition, 2002.

[20] W. Huang, P. Eades, and S.-H. Hong. 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*, pages 3:1–3:8. ACM, 2008. doi:10.1145/1377966.1377970.

[21] W. Huang, P. Eades, and S.-H. Hong. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Inform. Visual.*, 8(3):139–152, 2009. doi:10.1057/ivs.2009.10.

[22] W. Huang, S.-H. Hong, and P. Eades. Effects of crossing angles. In I. Fujishiro, H. Li, and K.-L. Ma, editors, *Proc. IEEE Pacific Visualization Symposium, PacificVis 2008*, pages 41–46. IEEE, 2008. doi:10.1109/PACIFICVIS.2008.4475457.

[23] V. Lauren and G. Pisinger. Automated analysis of vessel diameters in MR images. *Visualization, Imaging, and Image Processing*, pages 931–936, 2004.

[24] J. Pach. Every graph admits an unambiguous bold drawing. In M. van Krefeld and B. Speckmann, editors, *Graph Drawing, GD 2011*, volume 7034 of *LNCS*, pages 332–342. Springer, 2012. doi:10.1007/978-3-642-25878-7_32.

[25] R. C. Read and R. J. Wilson. *An Atlas of Graphs*. Oxford University Press, 1998.

[26] E. W. Weissstein. Dyck graph, Nov 2012. http://mathworld.wolfram.com/DyckGraph.html.

[27] E. W. Weissstein. Gray graph, Nov 2012. http://mathworld.wolfram.com/GrayGraph.html.