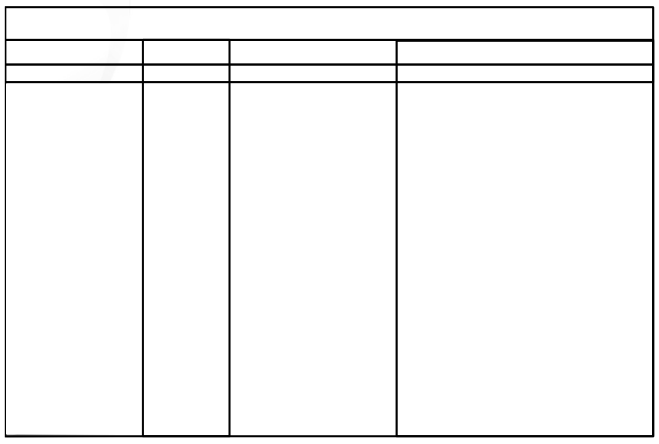


Figure 13: Optimal edge detection result for the table in Figure 12.

The edge detection process is divided into four distinct steps that are described in more detail in the following chapters:

1. Finding horizontal edges.
2. Finding vertical edges.
3. Finding crossing edges and creating "snapping points".
4. Finding cells (closed rectangular areas).

4.2.1 Finding horizontal edges

The horizontal edge detection algorithm starts by examining the pixels of a gray-scale image from the top left corner. The algorithm compares every top-bottom pair of adjacent pixels, looking for intensity value changes above a set threshold value. Once a pixel-pair with enough difference in their intensities is found, the algorithm proceeds to the right, comparing multiple pixels (in up-down direction) until the edge is no longer present or the right edge of the image is encountered. If the found edge is of sufficient length, it is accepted and registered as a horizontal edge in the image. The flow of the algorithm is visualized in Figure 14.

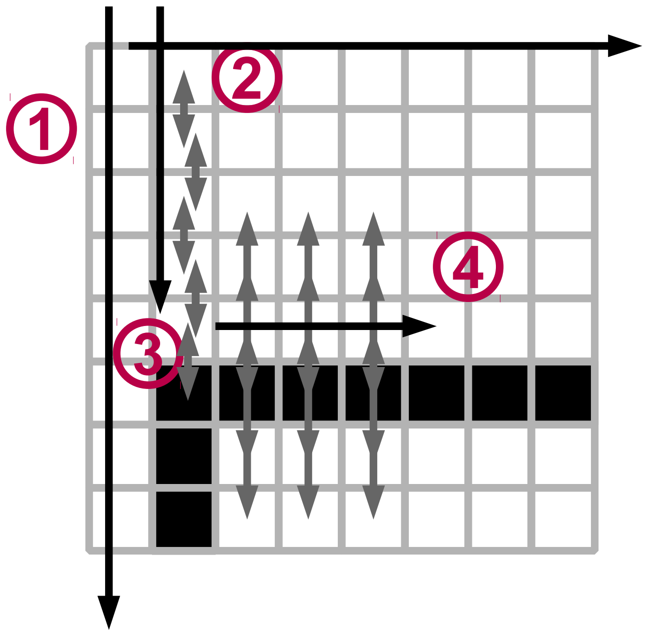
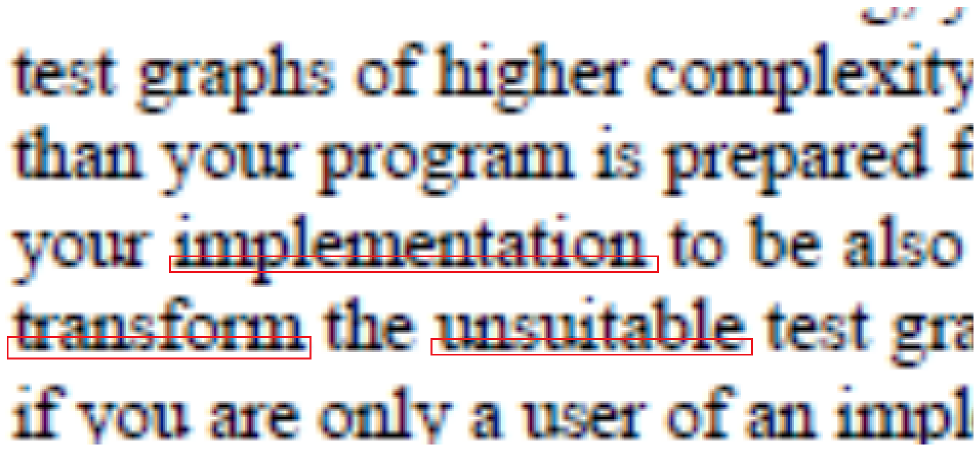


Figure 14: Graphical representation of the horizontal edge detection algorithm. The algorithm moves down the first column of pixels from the top-left corner of an image (1), comparing adjacent top-bottom pairs of pixels. Once it reaches the bottom of the image, it moves on to the next column to the right (2). When a horizontal edge is found (3), the algorithm proceeds along the edge to the right (4) comparing multiple pixels, and stopping when no edge is detected anymore (not shown). The search is then continued from the previously found edge starting-point (3) downward.

Using multiple pixels for detecting horizontal edge preservation, after the initial edge starting-point is found, helps the algorithm deal with edges that are not perfectly aligned, as well as with text elements that are on the edge, obscuring the underlying edge for short lengths at a time. Sufficiently long edges that have been found are saved into memory, so that when the algorithm encounters the same edge again in a subsequently examined column of pixels, it will not be examined again, but instead only skipped over.

If the *PDF* page is rendered as an image too small in pixel size, the bottoms of serifed fonts can blend or blur together, forming horizontal pseudo lines, as illustrated in Figure 15. To avoid this from happening the processed image has to be rendered in large enough size for the individual characters in a word to be separated adequately. Excluding the page text elements areas completely from the edge detection processed image areas, is not possible because in many tables the rows are packed together so tightly, that the text element areas overlap on real horizontal grid edges. The text elements areas often have some extra space under the actual text rendering, to accommodate characters that are partly drawn below the fonts baseline, such as characters “q”, “g” and “p” for example.



test graphs of higher complexity
than your program is prepared f
your implementation to be also
transform the unsuitable test gra
if you are only a user of an impl

Figure 15: Image quality is important for edge detection. Blurry text (rendered too small) can cause false detections. The three highlighted areas form continuous horizontal edge areas.

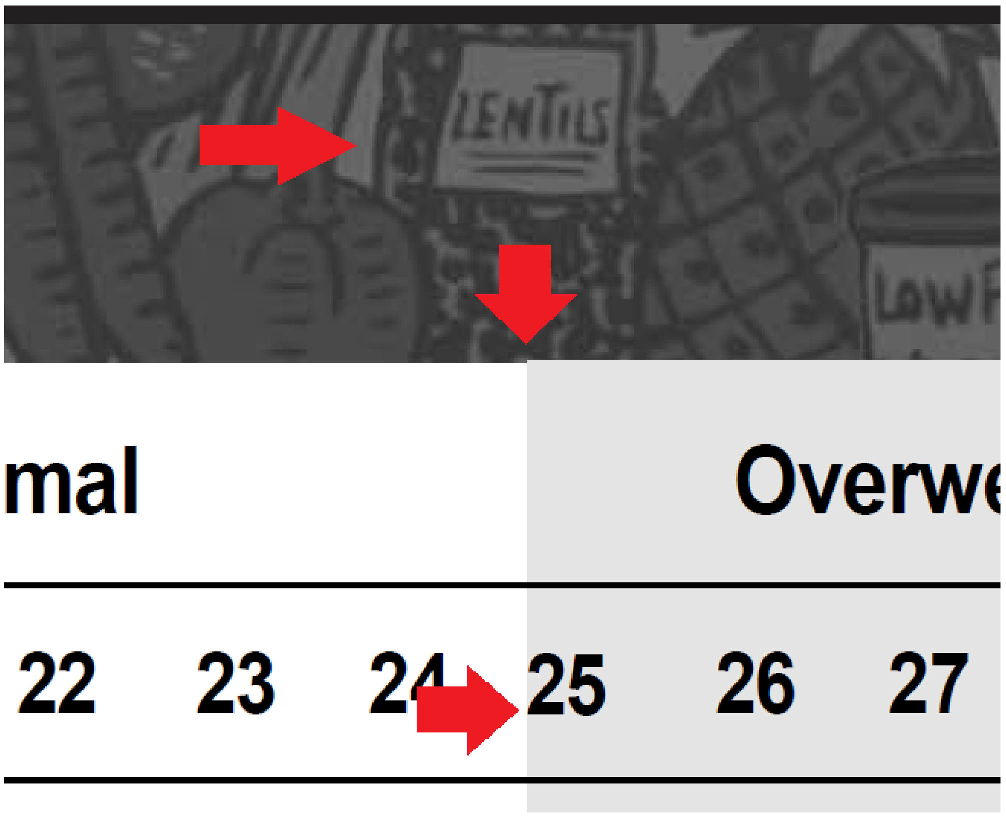


Figure 16: Common problems for edge detection are pointed out with red arrows. The illustrated problems include: graphic backgrounds, misaligned rectangles (or lines), edge-overlapping text, and low-threshold edges.

Using multiple pixels for finding continuing edges, avoids a few common problems as illustrated in Figure 16. Even though most *PDF* documents are created digitally (natively digital), not all edges can be assumed being in perfect alignment.

4.2.2 Finding vertical edges

The vertical edge finding process works much alike the horizontal edges finding process, with one major exception: the text element areas can be excluded from the processing. Unlike with horizontal edges, the real vertical edges very rarely overlap with the text element areas, because there are no issues with font baselines and empty areas within the text element in the left-right direction. Excluding the text element areas avoids the problem of setting the minimum edge length.

Without the exclusion of text elements, many characters within a text element cause false positive vertical edges. For example the leading edges of "I", "P", "L" and other long vertical lines containing characters are prime candidates for causing false positives. If the minimum vertical edge length is set too low (under row character height), all these characters are likely to show up as false positive edges. On the other hand, setting the minimum edge length too high the vertical separators will not show up at all in tightly gridded tables.

4.2.3 Finding and aligning crossing edges

Due to the nature of the edge finding algorithm, described in previous chapters, the crossing points of greater than 1 pixel thick vertical and horizontal edges represent discontinuities in the edge, as shown in Figure 17. The horizontal edge has a gap in it at the position of the original table separator line, and the same applies to the vertical edges. For the edges and table cells to become connected at these points, some further processing is required.

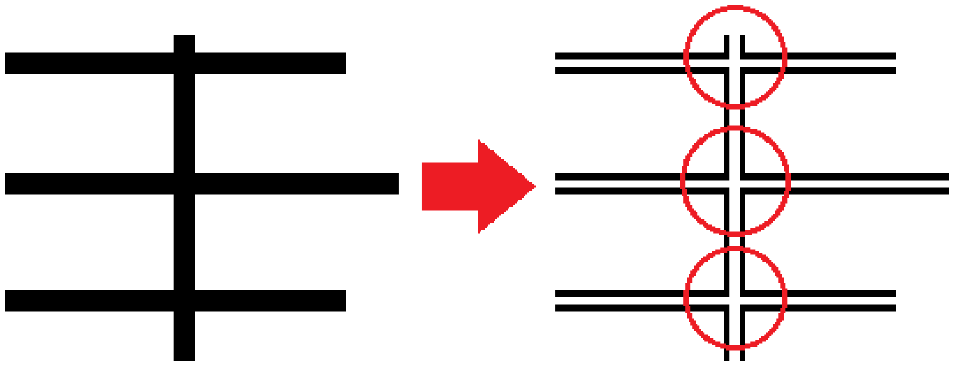


Figure 17: Thick table separator lines produce an edge detection pattern that has unconnected edges. "Snapping points" (red circles) are created to connect all the edge end-points, withing the area of a snapping point, to a single point.

A special snap-to-grid feature in employed. The algorithm searches for both vertical and horizontal positions in the image where many edges end. All edge end-points within a, for example, 10 pixel range from each other are averaged, and this average value will become a "snapping point" with a single pixel center point. All the edge end-points within the snapping point's area are reassigned to this single pixel center-point value. This procedure will connect all the edges in the vicinity of a snapping point.

There are some limitations to this method however. If the snapping point radius is set too big, some of the real grid crossing points become merged together, and if the radius is set too small, some of the edge end-points do not become connected and some rectangular areas are not found at all. Tables with thicker separator lines require longer snapping point radius lengths to become properly connected at their crossings.

4.2.4 Finding rectangular areas

The final step of the edge detection process is to identify closed rectangular spaces within the vertical and horizontal separator lines, and separating unconnected rectangular areas into different tables. This is done is several steps.

After the horizontal and vertical edge detection processing has finished, and the edges are aligned into appropriate snapping-points, points where a horizontal edge contacts a vertical edge are registered as crossing-points. The set of crossing-points is inclusive of points where both a vertical and a horizontal edge end. In other words, one edge does not have to continue through another, it is simply enough that the edges share a common pixel coordinate point.