Vol.2013-AL-143 No.4
2013/3/1

A New Compact Encoding of Rectangular Drawings

Yuto Saikawa[1,a)] Shin-ichi Nakano[1,b)]

Abstract: A rectangular drawing is a plane drawing of a graph in which every face is a rectangle. Rectangular drawings have an application for floorplans, which may have a huge number of faces, so a compact code to store the drawings is desired. The most compact code for rectangular drawings needs $4f-4$ bits, where f is the number of inner faces of the drawing.

In this paper we design a new simple and compact code for rectangular drawings. The code needs $4f-3-B$ bits for each rectangular drawing, where B is the number of inner faces having bottom line segments on the bottommost horizontal line segment. Since $B \geq 1$ holds, the length of our new code is at most $4f-4$ bits. Our encoding and decoding algorithms run in $O(f)$ time.

Keywords: graph,algorithm,encoding,rectangular drawing

1. Introduction

In this paper we study on a compact representation of a class of drawings. A *rectangular drawing* is a plane drawing of a graph in which every face, including the outer face, is a rectangle. See an example in Fig. 1. Rectangular drawings have an application for floorplans [2], which may have a huge number of faces, so a compact code to store the drawing is desired. For simplicity we assume that rectangular drawings have no vertex shared by four rectangles, as assumed in [2]-[5].

There are papers for compact encodings of rectangular drawings [2]-[5]. The most compact code needs at most $4f-4$ bits [3], where f is the number of inner faces of the drawing.

In this paper we design a new simple and compact code for rectangular drawings. The code needs $4f-3-B$ bits for each rectangular drawing, where B is the number of inner faces having bottom line segments on the bottommost horizontal line segment. For example the rectangular drawing in Fig. 1 has $B = 4$. If $B \geq 2$, our new code is shorter than $4f-4$. Our encoding and decoding algorithms run in $O(f)$ time.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 explains our new code. Finally Section 4 is a conclusion.

2. Preliminaries

In this section we give some definitions.

Let G be a connected graph. A *tree* is a connected graph with no cycle.

A drawing of a graph is *plane* if it has no two edges intersecting geometrically except at a vertex to which they are both incident. A plane drawing divides the plane into connected regions called *faces*. The unbounded face is called *the outer face*, and other faces are called *inner faces*. A *rectangular drawing* is a plane drawing in which every face (including the outer face) is a rectangle. See an example in Fig. 1.

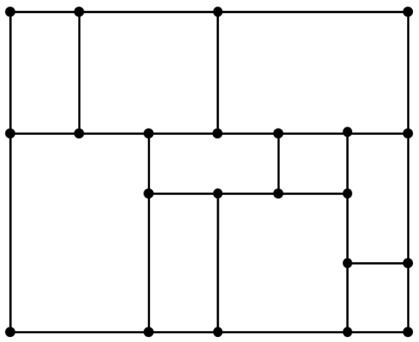

Fig. 1 An example of a rectangular drawing.

Let n be the number of vertices of a rectangular drawing, m the number of edges, and f' the number of faces (including the outer face). In this paper we only consider rectangular drawings with no vertex shared by four (or more) rectangles. This is a typical assumption to simplify discussions [2]-[5]. Thus a rectangular drawing has $n-4$ vertices with degree three, and four vertices with degree two (at the four corner of the outer face), and we have $2m = 3(n-4) + 2 \cdot 4$. The equation and the Euler's formula $n-m+f' = 2$ gives $n = 2f'$ and $m = 3f'-2$. Let $f = f'-1$ be the number of inner faces of a rectangular drawing. Then $\frac{n}{2} - 1 = f$ and $m = 3f+1$ hold.

A vertex with degree three is *W-missing* (West missing) if it has upward, rightward, downward edges, but no leftward edge. We denote the number of W-missing vertices as n_W. Similarly we define *E-missing* (East missing), *N-missing* (North missing), *S-missing* (South missing), n_E, n_N, and n_S. Note that, since each *W-missing* vertex is the left end of some maximal horizontal line segments, and each *E-missing* vertex is the right end of some maximal horizontal line segments, $n_W = n_E$ holds in any rectangular drawing. Similarly $n_N = n_S$ holds. Thus $n_E + n_N = \frac{n-4}{2}$.

[1] Department of Computer Science, Gunma Univercity
[a)] t10307022@gunma-u.ac.jp
[b)] nakano@cs.gunma-u.ac.jp

Note that any rectangular drawing has exactly four vertices of degree two.

3. The New Coding

A code of rectangular drawings with $6f + O(1)$ bits is known [4]. The code is based on the depth first traversal of a tree in the rectangular drawing. In this section we give a new code for rectangular drawings by improving the code in [4]. Our new code needs only $4f - O(1)$ bits for each rectangular drawing. We first review the code in [4].

Given a rectangular drawing R, cut the lower right corner of each inner face by replacing the lower right corner vertex by two vertices as depicted in Fig. 2 (b) and (d). Also we need some tricky replacement at the two lower corners of the outer face as depicted in Figs. 2 (g) and (h). See an example of those replacement in Fig. 3. Since we only break each cycle corresponding to an inner face at the lower right corner, the resulting graph has only one face and is still connected. So the resulting graph R' is a tree.

Starting at the upper left corner, we traverse the tree R' with depth first manner with right priority. (We regard R' as a wall and we go around the wall. For each vertical edge the left side is traced first to down, then later the right side is traced to top. For each horizontal edge the bottom side is traced first to right, then later the top side is traced to left.) When we arrive at a vertex from one of four directions, we always have only two choices for the next direction to trace, as shown in Fig. 4, even though there is four possible directions [4]. For instance see Case 2 in Fig. 4. When we trace an edge to right then the next trace is always down or right. If the next trace is up then it contradicts to the fact that we have cut the lower right corner of each inner face, and if the next trace is left then it contradicts to the fact that there is no vertex of degree one having a leftward edge in the tree R'. (Each degree one vertex has only upward edge because of the way of cut.)

While tracing each "side" of each edge of the tree R' we encode the next directions to trace into a bitstring. Since we trace each edge exactly twice (once in each direction), we need two bits for each edge. Thus we can encode the tree R' into $2m + 3 = 6f + 5$ bits. Note that we have appended two edges at the two lower corners of the outer face, and the last trace to left has no "next" trace, so we have "+4" and "−1". Given the $2m + 3$ bits code we can easily reconstruct the tree R' then the original drawing R using a simple algorithm with a stack [4].

Now we give our new code. Our idea is two fold.

(1) Even if we remove the bottommost horizontal line segment we can still encode the resulting drawing with the same manner.

(2) We can save the most of bits in Case 3 and Case 4 by checking some simple conditions.

Given a rectangular drawing R, we first remove the bottommost horizontal line segment. Now R has B less inner faces. Then we cut the lower right corner of each "remaining" inner face as depicted in Fig. 2 (b) and (d). See an example of a graph in Fig. 5. The resulting graph R'' is also a tree.

Then we traverse the tree R'' with depth first manner. We have four cases.

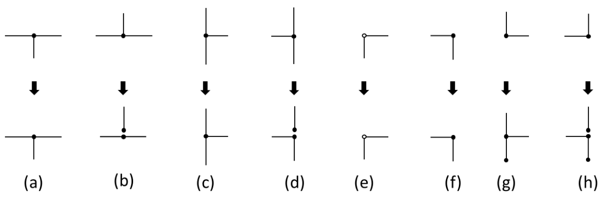

Fig. 2 The replacement of vertices.

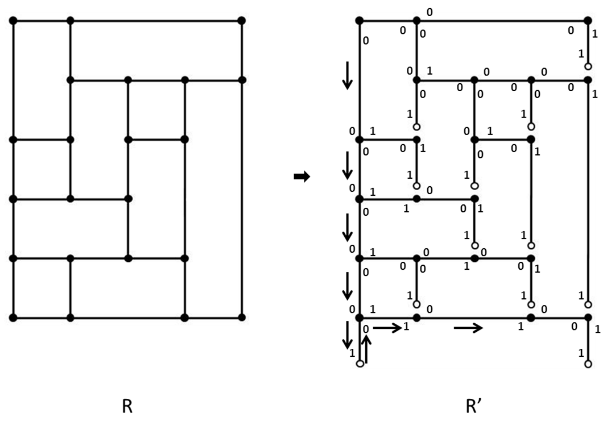

Fig. 3 The transformation into the tree.

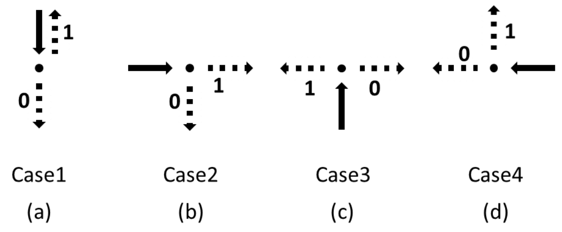

Fig. 4 The two choices of the next trace and their codes.

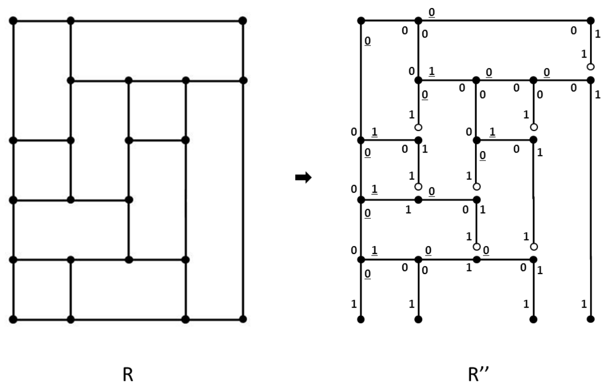

Fig. 5 The new transformation into the tree.

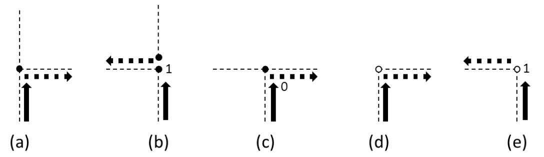

Fig. 6 Illustrations for Case 3. (a)*W-missing*, (b)*E-missing*, (c)*N-missing*, (d)*the upper left corner vertex*, and (e)*the upper right corner vertex*.

Case 1: When trace an edge downward.

The next trace is either down or up. Thus we need one bit to store the direction of the next trace. See Fig. 4 (a).

Case 2: When trace an edge rightward.

The next trace is either down or right. Thus we need one bit to store the direction of the next trace. See Fig. 4 (b).

Case 3: When trace an edge upward.

We have traced an edge, say e, upward and arrived at a vertex, say v. Clearly v has an edge to down since we have just traced the edge. Thus if v has degree three in the original drawing R then v is either W-missing, E-missing or N-missing. Otherwise v has degree two and v is either the upper left corner vertex or the upper right corner vertex. See Fig. 6. Since we have formerly traced (the other side of) the edge e downward we know whether v has an edge to left or not. (However we still do not know whether v has an edge to right .)

If v has no edge to left then v is either (a) has degree three and is W-missing, or (d) has degree two and is the upper left corner vertex. Then the next trace is always right. See Fig. 6 (a) and (d). Thus we need no bit to store the direction of the next trace.

Otherwise v has an edge to left, and the next trace is either left (Fig. 6 (b) and (e)) or right (Fig. 6 (c)). Thus two choices remain, so we need one bit to store the direction of the next trace. Note that in Fig. 6 (b) we have replaced the lower right corner of an inner face by two vertices.

Thus for Case 3 we need $n_E + n_N + 1 = \frac{n-4}{2} + 1 = \frac{n}{2} - 1$ bits in total.

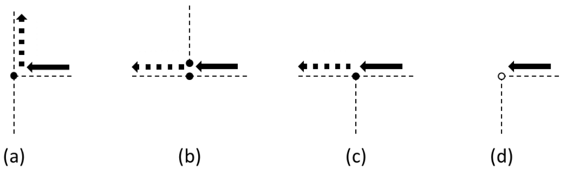

Fig. 7 Illustrations for Case 4 (a) W-missing, (b) S-missing, (c) N-missing, and (d) the upper left corner vertex.

Case 4: When trace an edge leftward.

We have traced an edge, say e, leftward and arrive at a vertex, say v. Clearly v has an edge to right since we have just traced the edge. See Fig. 7. Since we have formerly traced (the other side of) the edge e rightward we know (1) whether v has an edge to left or not, and (2) v has an edge to bottom or not. (However we still do not know whether v has an edge to top.)

If v is the upper left corner of the outer face (Fig. 7 (d)) then the leftward trace of e is the last trace in the traversal, so there is no “the next” trace.

Otherwise If v has no edge to left then v has degree three and is W-missing, and the next trace is always upward (Fig. 7(a)). Thus we need no bit to store the direction of the next trace.

Otherwise v has degree three and either (b) S-missing or (c) N-missing, and the next trace is always left. Thus we need no bit to store.

Thus for Case 4 we need no bit.

Theorem 1 One can encode a rectangular drawing into a bit string of length at most $4f + 1 - B$.

Proof. We trace each edge exactly twice (once in each direction). We estimate the length of the bitstring for each case. For Case 1 and Case 2 we need one bit to store the next direction after tracing each edge. Thus we need $m - B$ bits in total. For Case 3 we need $n_E + n_N + 1$ bits in total. For Case 4 we need no bit. Thus we need $(m - B) + (\frac{n}{2} - 1) = 3f + 1 - B + f = 4f + 1 - B$ bits in total. □

With a suitable data structure with a stack, similar to [4], one can reconstruct the original rectangular drawing from the bitstring in $O(f)$ time. Whenever we find a degree two vertex (in leftward trace) we push the vertex into the stack and whenever we find a degree one vertex we merge it with the vertex at the top of the stack to become an original degree three vertex.

Additionally one can save more bits as follows. Let T be the number of inner faces having the top horizontal line segment on the topmost horizontal line segment. For example the rectangular drawing in Fig. 1 has $T = 3$. If $T = 1$ then remove the topmost inner face and let R be the remaining rectangular drawing. Repeat this while $T = 1$ holds. Assume that this occurs k times. Now the number of inner face is $f - k$. We store this by appending the prefix 0^k1 into the code. If $f = k$ then we have $f + 1 \ll 4f - 4$ bit code for the drawing. Assume otherwise. Now $T \geq 2$ holds. Let F be the inner face having the upper right corner of the remaining drawing. We have two cases.

If in the original drawing F has (1) two or more neighbour faces to left (Fig. 8(a)), or (2) exactly one face to left and the lower left corner of F is S-missing(Fig. 8(b)), then the last part of the traversal is always up, $\underline{\text{right}}$, $\underline{\text{down}}$, $\underline{\text{up}}$, $\underline{\text{right}}$, $\underline{\text{down}}$, $\underline{\text{up}}$, then left $T \geq 2$ times.

Otherwise F has exactly one face to left and the lower left corner of F has a downward edge(Fig. 8(c)). Then the last part of the traversal is always up, $\underline{\text{right}}$, $\underline{\text{down}}$, $\underline{\text{up}}$, some lefts, up, $\underline{\text{right}}$, $\underline{\text{down}}$, $\underline{\text{up}}$, then left $T \geq 2$ times. Thus in each case we can save 6 bits, indicated by underlines. We append one bit as the prefix to distinguish these two cases.

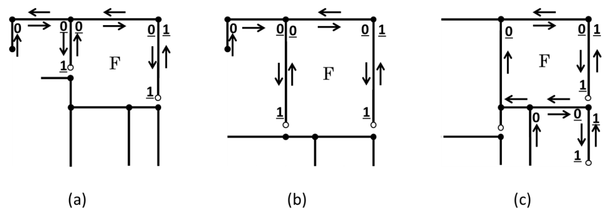

Fig. 8 Illustrations for the last part of the traversal.

Thus even if we remove those last part from the bitstring we can still reconstract the bitstring, then the tree R'', then the original rectanglar drawing R. Thus the length of the code is at most $k+1+4(f-k)+1-B+1-6 = 4f-3k-3-B \leq 4f-3-B \leq 4f-4$ bits.

4. Conclusion

In this paper we have designed a simple code for rectangular drawings. The code needs $4f - 3 - B$ bits, where f is the number of inner faces, and B is the number of inner faces having bottom line segments on the bottommost horizontal line segment. Both encoding and decoding algorithms run in $O(f)$ time. The number of rectangular drawings with no degree four vertices is $\Omega(11.56^f) = \Omega(2^{3.53f})$[1]. So we need at least $3.53f + c$ bits to encode a rectangular drawing for some constant c.

References

[1] K. Amano, S. Nakano, and K. Yamanaka, *On the number of rectangular drawings: Exact couting and lower and upper bounds,* IPSJ SIG Notes, AL-115(5):3340,2007.
[2] He, H.: On floor-plan of plane graphs, SIAM Journal on Computing, 28, pp.2150-21 67 (1999)
[3] Takahashi, T., Fujimaki, R., and Inoue, Y.: A $(4n-4)$-bit representation of a rectangular drawing or floorplan, Proc. of COCOON 2009, LNCS, 5609, pp.47-55 (2009)
[4] Yamanaka, K., Nakano, S.: Coding floorplans with fewer bits, IEICE TRANS. FUNDAMENTALS, E89-A, pp.1181-1185 (2006)
[5] Yamanaka, K., Nakano, S.: A compact encoding of rectangular drawings with efficient query supports, Proc. of AAIM 2007, LNCS, 4508, pp.68-81 (2007)