IEICE TRANS. FUNDAMENTALS, VOL.E96–A, NO.6 JUNE 2013

PAPER Special Section on Discrete Mathematics and Its Applications

A Compact Encoding of Rectangular Drawings with Edge Lengths

Shin-ichi NAKANO[†a)] ***and*** **Katsuhisa YAMANAKA**[††b)]**,** ***Members***

SUMMARY 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 compact code to store the drawings is desired. The most compact code for rectangular drawings needs at most $4f-4$ bits, where f is the number of inner faces of the drawing. The code stores only the graph structure of rectangular drawings, so the length of each edge is not encoded. A grid rectangular drawing is a rectangular drawing in which each vertex has integer coordinates. To store grid rectangular drawings, we need to store some information for lengths or coordinates. One can store a grid rectangular drawing by the code for rectangular drawings and the width and height of each inner face. Such a code needs $4f-4+f\lceil\log W\rceil+f\lceil\log H\rceil+o(f)+o(W)+o(H)$ bits*, where W and H are the maximum width and the maximum height of inner faces, respectively. In this paper we design a simple and compact code for grid rectangular drawings. The code needs $4f-4+(f+1)\lceil\log L\rceil+o(f)+o(L)$ bits for each grid rectangular drawing, where L is the maximum length of edges in the drawing. Note that $L\le\max\{W,H\}$ holds. Our encoding and decoding algorithms run in $O(f)$ time.

key words: *graph, algorithm, encoding, rectangular drawing, grid 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. Rectangular drawings have an application for floorplans [2], which may have a huge number of faces, so 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 [3]–[5].

There are papers for compact encodings of rectangular drawings [3]–[5]. The most compact code needs at most $4f-4$ bits [3], where f is the number of inner faces of the drawing. The code stores only the graph structures of rectangular drawings, so the length of each edge is not encoded.

A *grid rectangular drawing* is a rectangular drawing in which each vertex has integer coordinates. See an example in Fig. 1. To store grid rectangular drawings we need to store some information for lengths or coordinates. One can store a grid rectangular drawing by the code for (1) rectangular drawings, (2) the width and height of each inner face, and (3) the lengths of the bitstrings to represent widths and heights. Such a code needs $4f-4+f\lceil\log W\rceil+f\lceil\log H\rceil+o(f)+o(W)+o(H)$ bits, where W and H are the maximum width and the maximum height of inner faces, respectively.

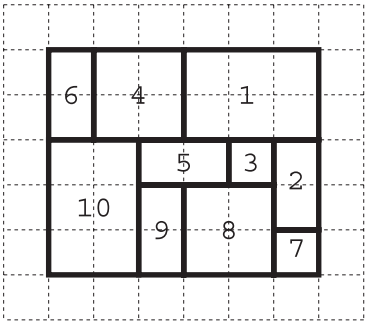

Fig. 1 An example of a grid rectangular drawing.

In this paper we design a simple and compact code for grid rectangular drawings. The code needs at most $4f-4+(f+1)\lceil\log L\rceil+o(f)+o(L)$ bits for each grid rectangular drawing, where L is the maximum length of edges in the drawing. Note that $L\le\max\{W,H\}$ holds and L may much smaller than W and H. Our encoding and decoding algorithms run in $O(f)$ time. Here we assume $\lceil\log L\rceil$ is less than the word size. Our encoding uses the encoding of [3] with appending information for edge lengths. For the paper to be self-contained we explain the encoding of [3] in our notation, which is conceptually simpler.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 explains the coding of [3] in our notation, then presents our encoding. Finally Sect. 4 is a conclusion.

2. Definition

The encoding of [3] first defines a numbering for inner faces of a rectangular drawing. Then remove each inner face following the numbering (See Fig. 2). On the removal of each face we store some information, which is enough to reconstruct the original drawing. In this section we explain the numbering.

A rectangle is *rectilinear* if it consists of only horizontal and vertical line segments. Let R be a rectilinear rectangle. The *upward ray* of R is the vertical half line with the bottom end point at the upper left corner of R. Similarly the

Manuscript received September 12, 2012.

[†]The author is with the Department of Computer Science, Gunma University, Kiryu-shi, 376-8515 Japan.

[††]The author is with the Department of Electrical Engineering and Computer Science, Iwate University, Morioka-shi, 020-8551 Japan.

a) E-mail: nakano@cs.gunma-u.ac.jp

b) E-mail: yamanaka@cis.iwate-u.ac.jp

DOI: 10.1587/transfun.E96.A.1032

*log denotes logarithm to the base 2.

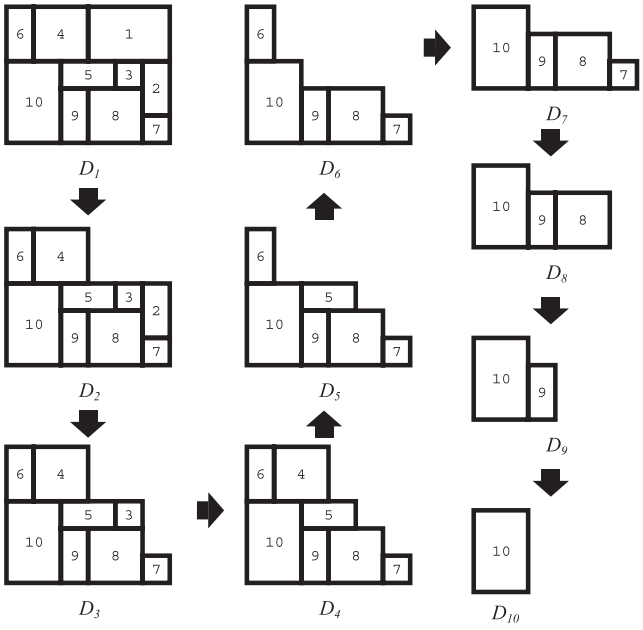

Fig. 2 An example of the removing sequence.

rightward ray of R is the horizontal half line with the left end point at the lower right corner of R. The *upper right area* of R is the area bounded by (1) the upward ray of R, (2) the upper horizontal line segment of R, (3) the right vertical line segment of R, and (4) the rightward ray of R.

Let S be a set of rectilinear rectangles on a plane. A rectangle R in S is *clear* if the upper right area of R intersect the proper inside of no other rectangle in S.

If the upward ray of a rectangle R_B contains the right vertical line segment of a rectangle R_U, then we say R_U is an *upward predecessor* of R_B. Intuitively we can remove R_B only after removing R_U. Similarly, if the rightward ray of a rectangle R_L contains the upper horizontal line segment of a rectangle R_R, then we say R_R is a *rightward predecessor* of R_L. Intuitively we can remove R_L only after removing R_R. A clear rectangle is *ready* if it has neither an upward predecessor nor a rightward predecessor.

We have the following lemma.

Lemma 2.1: Let S be a set of non-overlapping rectilinear rectangles. S has at least one ready rectangle.

Proof. Let R be the rectangle with the highest lower left corner. If S has two or more such rectangles then choose the rightmost one.

If R is ready we are done. Otherwise, (1) R has some rightward predecessor or (2) the upper right area of R intersects with the proper inside of some rectangle.

Let R' be the such rectangle with the highest lower left corner. If S has two or more such rectangles then choose the rightmost one. By the choice of R the lower left corner of R' is lower than the lower left corner of R, and the lower left corner of R' is located to the right of the lower left corner of R. Intuitively R' is located to the lower right of R.

If R' is ready then we are done. Otherwise, (1) R' has some rightward predecessor or (2) the proper inside of some rectangle intersects with the upper right area of R'. Let R'' be the such rectangle with the highest lower left corner. If S has two or more such rectangle then choose the rightmost one. Again R'' is located to the lower right of R'.

Repeating this, we always find a ready rectangle. □

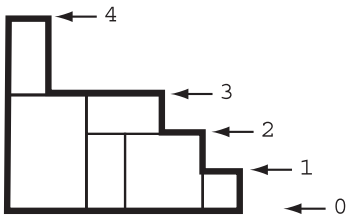

Fig. 3 An illustration for the base height.

The *removable rectangle* of S is the ready rectangle having the highest lower left corner. Thus any S has a unique removable rectangle.

Let D be a given rectangular drawing consisting of f inner faces, and S the set of rectangles corresponding to the inner faces of D. Remove the removable rectangle from S one by one, and assign integers $1, 2, \ldots, f$ to the inner faces according to the ordering. See an example in Fig. 2. One can compute the numbering in $O(f)$ time by maintaining the list of ready rectangles.

Let D be a given rectangular drawing and D_i the subdrawing consisting of the inner faces assigned numbers $i, i+1, \ldots, f$. See Fig. 2.

A *staircase* is a sequence of alternating horizontal and vertical line segments which is x-monotone and y-monotone.

Lemma 2.2: The boundary of the outerface of each $D_1, D_2, \ldots, D_f$ consists of (1) a staircase, (2) the leftmost vertical line segment, and (3) the lowest horizontal line segment.

Proof. See an example in Fig. 2. We can prove by induction. □

Now we define the *base height* for each inner face in D. Let R_i be the inner face in D assigned number i.

If $i = f$ then the base height of R_i is 0. Otherwise $i < f$ holds, and the y-coordinate of the lower horizontal line segment of R_i is equal to the y-coordinate of some horizontal line segment on the boundary of the outerface of D_{i+1}. Suppose that the y-coordinate of the lower horizontal line segment of R_i is equal to the k-th lowest horizontal line segment on the boundary of the outerface of D_{i+1}. Then the base height of R_i is defined to be k. Intuitively R_i is on the k-th step of the staircase of the boundary. See Fig. 3.

Next we define the type for each inner face. First introduce two dummy line segments into D_i: the vertical line segment with the bottom end at the upper left corner of D_i, and the horizontal line segment with the left end at the lower right corner of D_i. Now the upper left corner and the lower

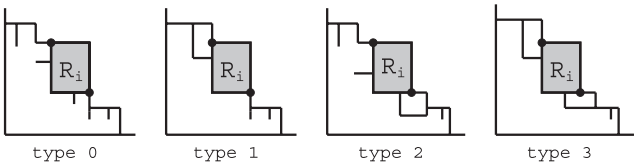

Fig. 4 The four types.

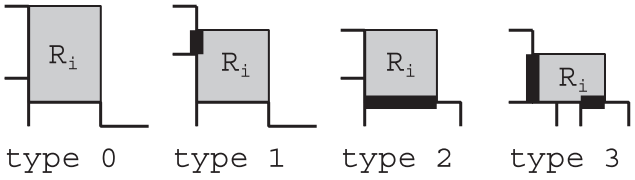

Fig. 5 Examples of hands.

Table 1 Base heights, types and the length of hands for the grid rectangular drawing in Fig. 1.

	face									
assigned number	1	2	3	4	5	6	7	8	9	10
base height	1	1	2	3	2	3	0	0	0	0
type	0	0	0	2	2	3	3	2	3	3
length of right hand	-	-	-	1	1	1	1	2	1	2
length of upper hand	-	-	-	-	-	2	1	-	2	3

right corner of any inner face R_i in D_i always has exactly three edges. The type of R_i is uniquely defined as follows. See Fig. 4. Note that if $i = f$ then R_i is always type 3.

Type 0 The upper left corner of R_i has no upward edge and the lower right corner of R_i has no rightward edge in D_i.

Type 1 The upper left corner of R_i has no leftward edge and the lower right corner of R_i has no rightward edge in D_i.

Type 2 The upper left corner of R_i has no upward edge and the lower right corner of R_i has no downward edge in D_i.

Type 3 The upper left corner of R_i has no leftward edge and the lower right corner of R_i has no downward edge in D_i.

Finally we define the *hands* of each inner face as follows.

If the upper left corner v of R has no leftward edge, then R_i has the *upper hand* which is the common vertical line segment of (1) R_i and (2) the face located to the left of v. If the lower right corner v of R_i has no downward edge, then R_i has the *right hand* which is the common horizontal line segment of (1) R_i and (2) the face located to the bottom of v. See examples in Fig. 5 and Table 1. The upper hands and the right hands are shown as thick lines in Fig. 5. If R_i is type 0 then R_i has no hands. In Table 1, base heights, types and the length of hands of inner faces of the grid rectangular drawing in Fig. 1 are shown.

3. Encoding of Drawings with Lenghts

Let D be a given grid rectangular drawing. First we encode

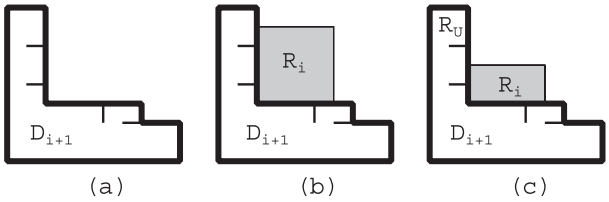

Fig. 6 Illustration for reconstruction.

f and $\lceil \log L \rceil$ by the gamma code [1], where L is the maximum length of edges in the drawing. The gamma code for integer i consists of the binary representation B_i of i appending the string of $|B_i| - 1$ zeros as the prefix. For instance the gamma code of 9 is 0001001. Thus the code for positive integer i needs at most $2\lceil \log(i+1) \rceil - 1$ bits. Then we remove each inner face in the order defined in Sect. 2. On the removal of each inner face R_i we encode (i) the base height of R_i, (ii) the type of R_i and (iii) the length of the hands. Now we show those are enough information to reconstruct the original drawing D. Note that by only (i) and (ii) we can encode rectangular drawings (without lengths) [3].

Let R_i be an inner face in D assigned number i. If $i = f$ then we can easily reconstruct D_f, since it is just one rectangle and its height and width equal to the length of the upper hand and the right hand. Otherwise $i < f$ holds, and assume we have D_{i+1}. We can reconstruct D_i by suitably introducing R_i into D_{i+1}, as follows. We show just one case, since other cases are similar. Assume D_{i+1} is shown in Fig. 6(a). Since we know the base height of R_i, say 2, we can find the horizontal line segment of D_{i+1} on which we are going to introduce R_i. We also know the type of R_i, say type 3. Hence, we can append R_i as shown in Fig. 6(b). If we append R_i so that R_i has some upward predecessor R_U, as shown in Fig. 6(c), then R_i is not removable in D_i and it contradicts to the assignment of the number i to R_i. Thus R_i has no upward predecessor. Similarly R_i has no rightward predecessor. Thus the only choice to append R_i is as in Fig. 6(b). Since we also know the length of hands we can suitably append R_i into D_{i+1} to reconstruct D_i. Thus we can reconstruct $D_f, D_{f-1}, \ldots, D_1 = D$.

Now we estimate the length of (i).

Lemma 3.1: One can encode the base heights of inner faces into a binary string of length at most $2(f-1)$.

Proof. Let D be a grid rectangular drawing, and $R_1, R_2, \ldots, R_f$ the inner faces in D in the order defined in Sect. 2. Now we encode the base heights of inner faces. We can observe that (1) the base height of R_f is always 0, (2) the base height of R_1 is either 0 or 1, and (3) the difference d_i of the two consecutive base heights, namely the base height of R_i minus the base height of R_{i+1}, is either -1 or 0 or positive integer. We are going to encode only these differences.

We encode -1 as 1, 0 as 01, 1 as 001, 2 as 0001, ... That is, the code of the difference d_i is the unary code of $d_i + 2$. Note that $d_i + 2$ is always a positive integer. We encode the base heights of inner faces as the sequence of the differences encoded by the above method. Suppose $(d_{f-1}, d_{f-2}, \ldots, d_1)$

is the sequence of the differences. For the grid rectangular drawing in Fig. 1 the sequence is $(0, 0, 0, 3, -1, 1, -1, -1, 0)$.

Assume that the base height of R_1 is 0. (The other case, the base height of R_1 is 1, is similar.) The base height of R_f is always 0. Now $\sum_{i=1}^{f-1} d_i = 0$ holds. Thus for each difference $d_i > 0$ we can assign distinct d_i of (-1)s among the differences.

Now we estimate the total length of the above code. For the difference 0 we need 2 bits. For the difference $d_i > 0$ we need $d_i + 2$ bits, but there are distinct d_i of (-1)s among the differences, each of which needs only 1 bit, so we need $2d_i + 2$ bits in total for the $d_i + 1$ differences, consisting of (1) a d_i and (2) d_i of $(-1)s$. Thus we need 2 bits for each difference on average and $2(f - 1)$ bits in total. □

Similarly we estimate the length of (iii).

Lemma 3.2: One can encode the lengths of hands into a binary string of length at most $(f + 1)\lceil \log L \rceil$.

Proof. Since the length of a hand is positive integer, say k, we encode it with binary code of $k - 1$. Thus we need $\lceil \log L \rceil$ bits for a hand. Now we show that the number of hands is at most $f + 1$. Assume D_{i+1} has k steps. If R_i is type 1 or 2 then D_i has also k steps. If R_i is type 0 then D_i has $k - 1$ steps. If R_i is type 3, then D_i has $k + 1$ steps. The number of steps in D_f and D_1 is always two. Thus the number of type 0 rectangles equals to the number of type 3 rectangles (excluding R_f). Each type 0 rectangle has no hand, each type 1 or 2 rectangle has one hand, and each type 3 rectangle has two hands. Thus we need one hand for each rectangle (excluding R_f) on average, and R_f has two hands. Thus the number of hands is always $f + 1$. □

Now we estimate the length of our code. (1) We need $2f-2$ bits to encode the sequence of the base heights of inner faces by Lemma 3.1. (2) We need $2f - 2$ bits to encode the types of inner faces excluding R_f. Each type needs 2 bits and R_f is always type 3 so we need not encode it. (3) We need at most $(f+1)\lceil \log L \rceil$ bits to encode the lengths of the hands by Lemma 3.2. We also need to store f and $\lceil \log L \rceil$. We store f and $\lceil \log L \rceil$ in gamma code [1], which need $2\lceil \log (f + 1) \rceil + 2\lceil \log (\lceil \log L \rceil + 1) \rceil - 2$ bits.

We have the following theorem.

Theorem 3.3: One can encode a grid rectangular drawing into a binary string of length at most $4f-4+(f+1)\lceil \log L \rceil + o(f) + o(L)$.

With a suitable data structure the encoding and decoding run in $O(f)$ time. Here we assume $\lceil \log L \rceil$ is less than the word size.

By maintaining the list of ready rectangles, we can encode in $O(f)$ time.

Now we explain our decoding. To introduce each inner face R_i to D_{i+1}, we need to find the step to which we put R_i. These can be done by maintaining the list of horizontal line segments of the outer face. Since we know the difference d_i between the two base heights of R_i and R_{i+1}, the step to which we put R_i can be found in $O(d_i)$ time using the list. The total time to find the steps is $O(f)$. Proof is similar to the proof of Lemma 3.1.

We have the following theorem.

Theorem 3.4: Our encoding and decoding algorithms run in $O(f)$ time.

4. Conclusion

In this paper we have designed a simple code for grid rectangular drawings. The code needs at most $4f - 4 + (f + 1)\lceil \log L \rceil + o(f) + o(L)$ bits, where f is the number of inner faces, and L is the maximum lengths of edges in the drawing. Both encoding and decoding run in $O(f)$ time.

References

[1] P. Elias, "Universal codeword sets and representations of the integers," IEEE Trans. Inf. Theory, vol.IT-21, no.2, pp.194–203, 1975.
[2] H. He, "On floor-plan of plane graphs," SIAM J. Comput., vol.28, pp.2150–2167, 1999.
[3] T. Takahashi, R. Fujimaki, and Y. Inoue, "A $(4n-4)$-bit representation of a rectangular drawing or floorplan," Proc. COCOON 2009, LNCS, 5609, pp.47–55, 2009.
[4] K. Yamanaka and S. Nakano, "Coding floorplans with fewer bits," IEICE Trans. Fundamentals, vol.E89-A, no.5, pp.1181–1185, May 2006.
[5] K. Yamanaka and S. Nakano, "A compact encoding of rectangular drawings with efficient query supports," Proc. AAIM 2007, LNCS, 4508, pp.68–81, 2007.

Shin-ichi Nakano received his B.E. and M.E. degrees from Tohoku University, Sendai, Japan, in 1985 and 1987, respectively. In 1987 he joined Seiko Epson Corp. and in 1990 he joined Tohoku University. In 1992, he received Dr. Eng. degree from Tohoku University. Since 1999 he has been a faculty member of Department of Computer Science, Faculty of Engineering, Gunma University. His research interests are graph algorithms and graph theory. He is a member of IPSJ, JSIAM, ACM, and IEEE Computer Society.

Katsuhisa Yamanaka is an assistant professor of Department of Electrical Engineering and Computer Science, Faculty of Engineering, Iwate University. He received B.E., M.E. and Dr. Eng. degrees from Gunma University in 2003, 2005 and 2007, respectively. His research interests include combinatorial algorithms and graph algorithms.