Vol.2011-AL-136 No.1
2011/9/6

A Compact Encoding of Rectangular Drawings with Edge Lengths

Shin-ichi Nakano[†1] and Katsuhisa Yamanaka [†2]

A rectangular drawing is a plane drawing of a graph in which every face is a rectangle. Rectangular drawings have an application for floorplans. The most compact code for a rectangular drawings needs at most $4f - 4$ bits, where f is the number of inner faces of the drawing. This code encodes only the graph structure of a rectangular drawing, so the length of each edge is not encoded.

A grid rectangular drawing is a rectangular drawing in which each vertex has integer coordinates. One can encode a grid rectangular drawing including the length of each edge by the code for rectangular drawings appending the sequence of the lengths of edges. Such a code needs at most $4f + L$ bits, where L is the total length of the edges in the drawing.

In this paper we design a straightforward code for grid rectangular drawings including the length of each edge. The code needs at most $4f + L/2$ bits for each grid rectangular drawing. Our encoding and decoding algorithms are straightforward and run in $O(f)$ time.

1. Introduction

Studying representation issues on graphs is very natural [6]. In this paper we study on a compact representation of drawn graphs. 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]. By compact representation we can store the huge size of current floorplans into RAM, not disks.

There are some papers for compact encodings of rectangular drawings. For example see [3–5]. The most compact code needs at most $4f - 4$ bits[3], where f is the number of inner faces of the drawing. Those encode 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. As in [3–5] we only consider rectangular drawings having no vertex shared by four rectangles.

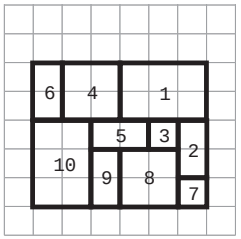

Fig. 1 An example of a grid rectangular drawing.

In this paper we design a straightforward encoding of grid rectangular drawings including the length of each edge. The encoding needs at most $4f + L/2$ bits for each grid rectangular drawing D, where f is the number of inner faces of D, and L is the total length of the edges in D. Our main idea is (1) encoding the lengths of only subset of edges, called "hands", is enough to reconstruct the given grid rectangular drawing, and (2) the total length of hands is bounded by $L/2$. Our encoding and decoding algorithms are straightforward and run in $O(f)$ time.

The rest of the paper is organized as follows. Section 2 defines a numbering of faces of a rectangular drawing. Section 3 give some definitions. Section 4 presents our first encoding. In Section 5 we improve the encoding. Finally Section 6 is a conclusion.

2. The Numbering of Faces

A numbering of inner faces of a rectangular drawing is known [3]. To encode the drawing we remove each inner face in the order of this numbering. See Fig. 2. At each removal we store some information which is needed to reconstruct the original drawing.

For the paper to be self-contained we explain the numbering in our terms. We need some definitions.

A rectangle is *rectilinear* if it consists of horizontal and vertical line segments.

†1 Department of Computer Science, Gunma University.
nakano@cs.gunma-u.ac.jp
†2 Department of Electrical Engineering and Computer Science, Iwate University.
yamanaka@cis.iwate-u.ac.jp

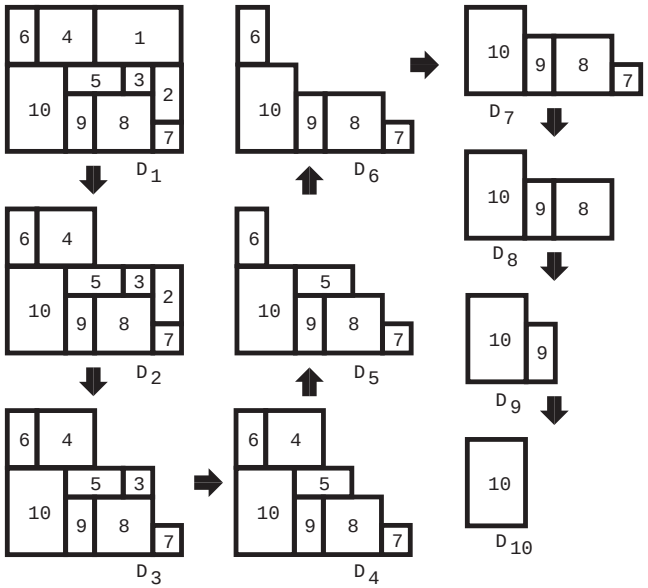

Fig. 2 An example of the removing sequence.

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 *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 upper right 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 in S is *clear* if the upper right area of R does not intersect the proper inside of any 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 top horizontal line segment of a rectangle R_R, then we say R_R is a *rightward predecessor* of R_L. 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, R has some rightward predecessor or 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. □

The *removable rectangle* of S is the ready rectangle having the highest lower left corner. By the definition the removable rectangle is unique.

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. Now we define a numbering of inner faces as follows.

Repeatedly remove the removable rectangle of S from S. In the order of the removal we assign integers $1, 2, \cdots, f$ to the inner faces. See an example in Figs. 1 and 2.

3. Base Height, Type and Hand

We need more definitions to explain our encoding. Let D be a given rectangular drawing consisting of f inner faces, and D_i the drawing induced by the inner faces assigned numbers $i, i+1, \cdots, f$.

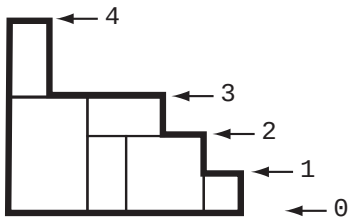

Fig. 3 An illustration for the base height.

A *staircase* is a sequence of alternating horizontal and vertical line segments such that it intersects every horizontal line at most once and every vertical line at most once.

Lemma 3.1 For $i = 1, 2, \cdots, f$, the boundary of the outerface of D_i 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 R in D. Assume that the number assigned to R is i. We have two cases.

If the lower horizontal line segment of R is contained in the lowest horizontal line segment of D, then the base height of R is 0. Thus if $i = f$ then the base height of R is always 0, since the lower horizontal line segment of R is contained in the lowest horizontal line segment of D.

Otherwise R is on some horizontal line segment on the boundary of the outerface of D_{i+1}.

Assume that the lower horizontal line segment of R is contained in the k-th lowest horizontal line segment of the boundary of the outerface of D_{i+1}. Then the base height of R is defined to be k. Intuitively R is on the k-th step of the staircase of the boundary of the outerface of D_{i+1}. See Fig. 3.

Next we define the type for each inner face R in D. For convenience we introduce the following two dummy line segments into D. The vertical line segment with bottom end at the upper left corner of D, and the horizontal line segment with left end at the lower right corner of D. Now the upper left corner and the lower right corner of any inner face has exactly three edges. The type of R is uniquely defined among the following four types. See Fig. 4.

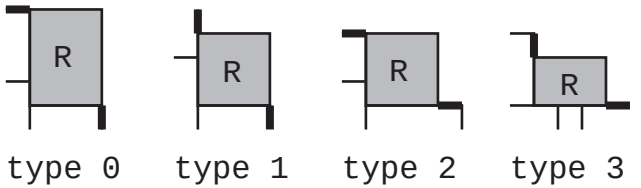

Fig. 4 The four types.

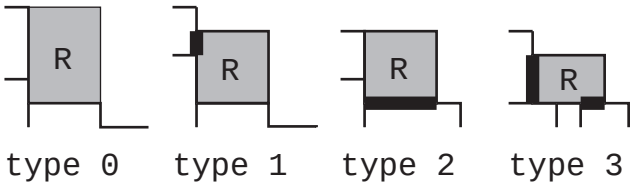

Fig. 5 Examples of hands.

Type 0 The upper left corner of R has no upward edge, and the lower right corner of R has no rightward edge.

Type 1 The upper left corner of R has no leftward edge, and the lower right corner of R has no rightward edge.

Type 2 The upper left corner of R has no upward edge, and the lower right corner of R has no downward edge.

Type 3 The upper left corner of R has no leftward edge, and the lower right corner of R has no downward edge.

Note that R_f is always type 3.

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

If the upper left corner vertex v of R has no leftward edge, then R has the *upper hand* which is the common vertical boundary line segment of (1) R and (2) the face located to the left of v.

If the lower right corner vertex v of R has no downward edge, then R has the *right hand* which is the common horizontal boundary line segment of (1) R and (2) the face located to the bottom of v.

See some examples in Fig. 5 and Table 1. The upper hands and the right hands are shown as thick lines. If R is type 0 then R has no 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

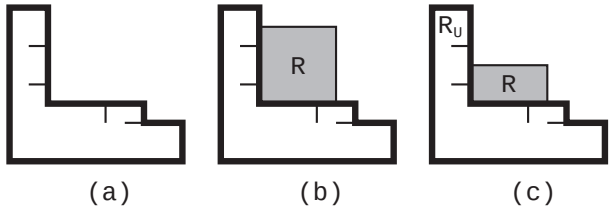

Fig. 6 Illustration for reconstruction.

4. Encoding

Let D be a given grid rectangular drawing. We remove each inner face in the order defined in Section 2. For the removal of each rectangle R we encode (1) the base height of R, (2) the type of R and (3) the length of the hands. Now we show those are enough information to reconstruct the original drawing D.

Let R be a rectangle in D, and i the number assigned to R. 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, assume we have D_{i+1}. We can also reconstruct D_i by suitably introducing R into D_{i+1}, as follows. Even though we have several cases we show just one case, since those cases are similar. We have a drawing D_{i+1} as shown in Fig. 6(a) and are going to introduce R on the suitable location on the boundary of D_{i+1}, which is shown by thick lines. Since we know the base height of R, say 2, we can find the horizontal line segment on which we introduce R. See Fig. 6(b). We also know the type of R, say type 3. We need to introduce R so that R has neither upward predecessor nor rightward predecessor. Since if we introduce R so that R has upward predecessor R_U, as shown in Fig. 6(c), then R is not removable in D_i and it contradicts to the assignment of number i to R. Similarly R cannot have rightward predecessor. Thus the only choice is shown in Fig. 6(b). Since we also know the length of hands we can suitably introduce R into D_{i+1} to have D_i.

Similarly for each case we can suitably introduce R into D_{i+1}. Therefore we can inductively reconstruct D.

The *unary representation* of a positive integer k consists of the $k-1$ consecutive 0s followed by a 1. For instance the unary code of 3 is 001. We can encode any sequence of positive integers by concatenating their unary codes, and can reconstruct the positive integers from the code. Note that if we encode the sequence of positive integers by concatenating their binary codes we cannot reconstruct the positive integers.

If we encode the sequence of lengths of the heights and widths of inner rectangles by the concatenation of their unary representations, then we need L bits in total, where L is the total length of the line segments of D. On the other hand our code encodes only the lengths of hands, which is a subset of the line segments of D, so the code is more compact. Let $H(D)$ be the total length of the hands of D, and $L(D)$ the total length of line segments of D. We have the following lemma.

Lemma 4.1 Let $D^0, D^{90}, D^{180}, D^{270}$ be the rectangular drawings derived from D by rotating clockwise $0, 90, 180, 270$ degrees.

$$\min\{H(D^0), H(D^{90}), H(D^{180}), H(D^{270})\} \leq L(D)/2$$

Proof. We can show each edge (x, y) of D becomes a hand at most twice in $D^0, D^{90}, D^{180}, D^{270}$. Assume (x, y) is a horizontal edge. The case (x, y) is a vertical edge is similar. Also assume x is located to the left of y. We have the following nine cases.

Note that a horizontal edge (u, v) is the right hand of some face R if the right end vertex v has no downward edge, and a vertical edge (u, v) is the upper hand of some face R if the top end vertex u has no leftward edge.

Case NN x has no upward edge, and y has no upward edge: See Fig. 7(a).

Case NE x has no upward edge, and y has no rightward edge: See Fig. 7(b).
Case NS x has no upward edge, and y has no downward edge: See Fig. 7(c).
Case WN x has no leftward edge, and y has no upward edge: See Fig. 7(d).
Case WE x has no leftward edge, and y has no rightward edge: See Fig. 7(e).
Case WS x has no leftward edge, and y has no downward edge: See D^{180} of Fig. 7(b).
Case SN x has no downward edge, and y has no upward edge: See Fig. 7(f).
Case SE x has no downward edge, and y has no rightward edge: See D^{180} of Fig. 7(d).
Case SS x has no downward edge, and y has no downward edge: See D^{180} of Fig. 7(a).

□

Thus after a suitable rotation we can encode the sequence of the lengths of the hands into a binary string of at most $L(D)/2$ bits.

Now we give our code for a grid rectangular drawing. Let R_f be the face assigned number f. Note that the base height of R_f is 0 and the type of R_f is 3, so we do not encode them. Our code consists of the following four sections. (1) the sequence of the base heights of inner faces except R_f. (2) the sequence of types of inner faces except R_f. (3) the sequence of the lengths of the hands of inner faces. (4) two bits to denote which drawing among the four rotated drawings to use.

5. Improvement

Let D be a grid rectangular drawing, and $R_1, R_2, \cdots, R_f$ the rectangles in D in the order defined in Section 2. If we encode the sequence of the base heights of inner faces as the concatenation of their unary codes then it may need much bits. (Again we cannot encode the sequence of the base heights of inner faces as the concatenation of their binary codes since we cannot reconstruct the base heights from the code.) We can observe the base height of R_f is always 0, the base height of R_1 is either 0 or 1, and the difference of the two consecutive base heights, 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, $\cdots$. That is, the code of k is the unary code of $k+2$. Note that $k+2$ is always a positive integer. We encode the sequence of the base heights of inner faces as the sequence of the differences encoded by the above method. We have the following lemma.

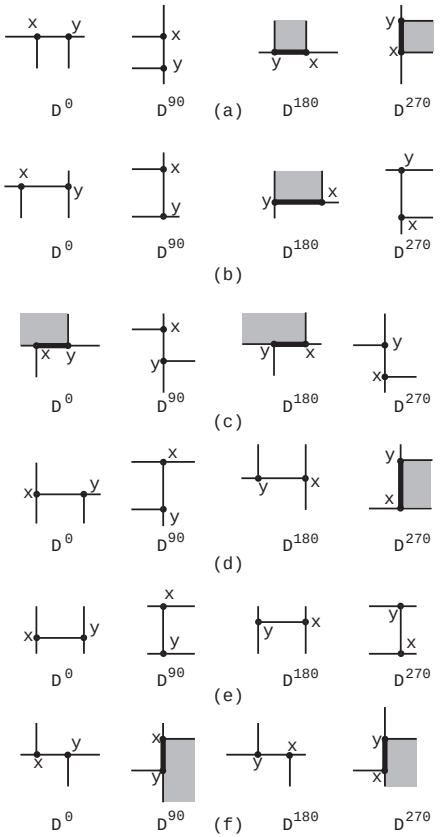

Fig. 7 Illustration for Lemma 3.

Lemma 5.1 One can encode the sequence of the base heights of inner faces into a binary string of length at most $2(f-1)$.

Proof. By amortized analysis. Let d_i be the base height of R_i minus the base height of R_{i+1}. Now $(d_1, d_2, \cdots, d_{f-1})$ is the sequence of the difference. Each d_i is either -1 or 0 or positive integer.

Assume that the height of R_1 is 0. (The other case the height of R_1 is 1 is similar.) The base height of R_f is always 0. Thus $\sum_{i=1,2,\cdots,f-1} d_i = 0$ holds, Now $\sum_{i\in I^+} d_i - \sum_{i\in I^-} d_i = 0$ holds, where $I^+ = \{i|d_i > 0, 1 \leq i < f\}$ and $I^- = \{i|d_i = -1, 1 \leq i < f\}$. Thus for each difference $k > 0$ we can assign distinct k of (-1)s among the differences.

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

Now we compute the length of our code. (1) We need $2f - 2$ bits to encode the sequence of the base heights of inner faces except R_f. The base height of R_f is always 0 so we need not encode it. (2) We need $2f - 2$ bits to encode the sequence of the types of inner faces except 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 $L(D)/2$ bits to encode the sequence of the lengths of the hands in some rotated drawing. (4) We need two bits to denote which drawing to use among the four rotated drawings. The derived binary string is the code for the given grid rectangular drawing D. We have the following theorem.

Theorem 5.2 One can encode a grid rectangular drawing into a binary string of length at most $4f + L(D)/2 - 2$.

With a suitable data structure the encoding and decoding run in $O(f)$ time.

6. Conclusion

In this paper we have designed a straightforward code for grid rectangular drawings. The code needs at most $4f + L/2$ bits, where f is the number of inner faces, and L is the total lengths of edges in the drawing. Both encoding and decoding run in $O(f)$ time.

If we know the number of instances in a class C, then the lower bound of the length of bits to encode an instance in C, which is the information-theoretically minimum number of bits to encode an instance in C, is $\log |C|$. Since the class of grid rectangular drawings with f faces has infinite instances we have no such lower bound.

To represent the lengths of hands we can use Gamma code [1] instead of the unary code. The Gamma code for integer i consist of the binary representation B_i of i appending the string of $|S_i| - 1$ zeros as the prefix. For instance the Gamma code of 9 is 0001001. Thus the code for positive integer n needs at most $2\lceil\log(n+1)\rceil - 1$ bits. If the length of hands are suitably long we can encode the length of hands, which is a positive integer sequence, by a sequence of Gamma codes. Then we can encode a grid rectangular drawing into a binary string of length at most $4f + \sum_{e\in E} \lceil\log(\ell(e) + 1) - 1\rceil$, where E is the set of edges and $\ell(e)$ is the length of an edge e.

References

[1] Elias, P.: Universal codeword sets and representations of the integers, Information Theory, IEEE Transactions, 21, pp.194–203 (1975)
[2] He, H.,: On floor-plan on plane graphs, SIAM Journal on Computing, 28, pp.2150–2167 (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)
[6] Spinrad, J. P.: Efficient graph representations, AMS (2003)