Algorithmica
DOI 10.1007/s00453-011-9525-2

Optimal Polygonal Representation of Planar Graphs

C.A. Duncan · E.R. Gansner · Y.F. Hu · M. Kaufmann · S.G. Kobourov

Received: 12 August 2010 / Accepted: 25 April 2011

Abstract In this paper, we consider the problem of representing planar graphs by polygons whose sides touch. We show that at least six sides per polygon are necessary by constructing a class of planar graphs that cannot be represented by pentagons. We also show that the lower bound of six sides is matched by an upper bound of six sides with a linear-time algorithm for representing any planar graph by touching hexagons. Moreover, our algorithm produces convex polygons with edges having at most three slopes and with all vertices lying on an $O(n) \times O(n)$ grid.

Keywords Planar graphs · Contact graphs · Graph drawing · Polygonal drawings

A preliminary version of this paper appeared in LATIN 2010, Oaxaca, Mexico.

C.A. Duncan
Dept. of Computer Science, Louisiana Tech University, Ruston, LA, USA
e-mail: duncan@latech.edu

E.R. Gansner (✉) · Y.F. Hu
AT&T Labs – Research, Florham Park, NJ, USA
e-mail: erg@research.att.com

Y.F. Hu
e-mail: yifanhu@research.att.com

M. Kaufmann
Wilhelm-Schickhard-Institut for Computer Science, Tübingen University, Tübingen, Germany
e-mail: mk@informatik.uni-tuebingen.de

S.G. Kobourov
Dept. of Computer Science, University of Arizona, Tucson, AZ, USA
e-mail: kobourov@cs.arizona.edu

Published online: 11 May 2011

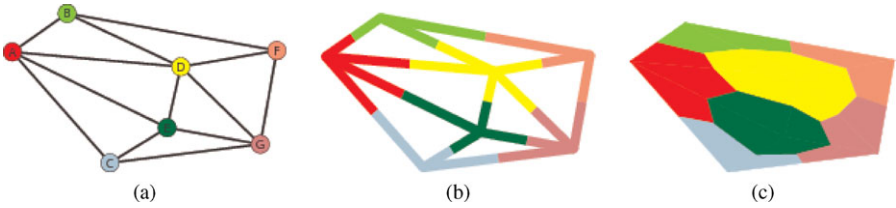

Fig. 1 (**a**) A drawing of a planar graph. (**b**) We apportion the edges to the endpoints by cutting each edge in half. (**c**) We then apportion the faces to form polygons

1 Introduction

For both theoretical and practical reasons, there is a large body of work considering how to represent planar graphs as *contact graphs*, i.e., graphs whose vertices are represented by geometrical objects with edges corresponding to two objects touching in some specified fashion. Typical classes of objects might be curves, line segments or isothetic rectangles. An early result is Koebe's theorem [24], which shows that all planar graphs can be represented by touching disks.

In this paper, we consider contact graphs whose objects are simple polygons, with an edge occurring whenever two polygons have non-trivially overlapping sides. As with treemaps [5], such representations are preferred in some contexts [6] over the standard node-link representations for displaying relational information. Using adjacency to represent a connection can be much more compelling, and cleaner, than drawing a line segment between two nodes. For ordinary users, this representation suggests the familiar metaphor of a geographical map.

It is clear that any graph represented this way must be planar. As noted by de Fraysseix et al. [9], it is also easy to see that all planar graphs have such representations for sufficiently general polygons. Starting with a straight-line planar drawing of a graph, we can create a polygon for each vertex by taking the midpoints of all adjacent edges and the centers of all neighboring faces. Note that the number of sides in each such polygon is proportional to the degree of its vertex. Moreover, these polygons are not necessarily convex; see Fig. 1.

It is desirable, for aesthetic, practical and cognitive reasons, to limit the complexity of the polygons involved, where "complexity" here means the number of sides in the polygon. Fewer sides, as well as wider angles in the polygons, make for simpler and cleaner drawings. In related applications such as floor-planning [28], physical constraints make polygons with very small angles or many sides undesirable. One is then led to consider how simple such representations can be. How many sides do we really need? Can we insist that the polygons be convex, perhaps with a lower bound on the size of the angles or the edges? If limiting some of these parameters prevents the drawing of all planar graphs, which ones can be drawn?

1.1 Our Contribution

This paper provides answers to some of these questions. Previously, it was known [16, 28] that triangulated planar graphs can be represented using non-convex octagons. On

the other hand, it is not hard to see that one cannot use triangles (e.g., K_5 minus one edge cannot be represented with triangles [13]).

Our main result is showing that hexagons are necessary and sufficient for representing all planar graphs. For necessity we construct a class of graphs that cannot be represented using five or fewer sides. For sufficiency, we prove the following:

Theorem 1 *For any planar graph G on n vertices, we can construct in linear time on an $O(n) \times O(n)$ grid a touching hexagons representation of G with convex regions. Moreover, if the graph is a triangulation, the representation is also a tiling.*

Note, if the input graph is not triangulated, there might be convex holes. We, in fact, prove this theorem using two different methods. First, in Sects. 3 and 4, we describe a linear-time algorithm that produces a representation using convex hexagons along with a linear-time compaction algorithm to reduce the initial exponential area to an $O(n) \times O(n)$ integer grid. Second, in Sect. 5, we show how modifying Kant's algorithm for hexagonal grid drawings of 3-connected, 3-regular planar graphs [21] produces a similar result by different means. In both variations, the drawings use at most three slopes for the sides, for example, 1, 0 and -1.

1.2 Related Work

As remarked above, there is a rich literature related to various types of contact graphs. There are many results considering curves and line segments as objects (cf. [17, 18]). For closed shapes such as polygons, results are rarer, except for axis-aligned (or *isothetic*) rectangles. In a sense, results on representing planar graphs as "contact systems" can be dated back to Koebe's 1936 theorem [24] which states that any planar graph can be represented as a contact graph of disks in the plane.

The focus of this paper is side-to-side contact of polygons. The algorithms of He [16] and Liao *et al.* [28] produce contact graphs of this type for triangulated graphs, with nodes represented by the union of at most two isothetic rectangles, thus giving a polygonal representation by non-convex octagons.

We now turn to contact graphs using isothetic rectangles, which are often referred to as *rectangular layouts*. This is the most extensively studied class of contact graphs, due in part to its relation to application areas such as VLSI floor-planning [26, 35], architectural design [31] and geographic information systems [12], but also due to the mathematical ramifications and connections to other areas such as rectangle-of-influence drawings [29] and proximity drawings [2, 20].

Graphs allowing rectangular layouts have been fully characterized [30, 33] with linear algorithms for deciding if a rectangular layout is possible and, if so, constructing one. The simplest formulation [6] notes that a graph has a rectangular layout if and only if it has a planar embedding with no filled triangles. Thus, K_4 has no rectangular layout. Buchsbaum *et al.* [6] also show, using results of Biedl *et al.* [4], that graphs that admit rectangular layouts are precisely those that admit a weaker variation of planar rectangle-of-influence drawings.

Rectangular layouts required to form a partition of a rectangle are known as *rectangular duals*. In a sense, these are "maximal" rectangular layouts; many of the results concerning rectangular layouts are built on results concerning rectangular duals.

Graphs admitting rectangular duals have been characterized [15, 25, 27] and there are linear-time algorithms [15, 23] for constructing them.

Another view of rectangular layouts arises in VLSI floor-planning, where a rectangle is partitioned into rectilinear regions so that region adjacencies correspond to a given planar graph. It is natural to try to minimize the complexities of the resulting regions. The best known results are due to He [16] and Liao *et al.* [28] who show that regions need not have more than 8 sides. Both of these algorithms run in $O(n)$ time and produce layouts on an $O(n) \times O(n)$ integer grid where n is the number of vertices.

Rectilinear cartograms can be defined as rectilinear contact graphs for vertex-weighted planar graphs, where the area of a rectilinear region must be proportional to the weight of its corresponding node. Even with this extra condition, de Berg *et al.* [3] show that rectilinear cartograms can always be constructed in $O(n \log n)$ time, using regions having at most 40 sides. The resulting regions, however, are highly non-convex and can have poor aspect ratio. Recently, Alam *et al.* [1] describe lower bounds and matching constructive algorithms that minimize the complexity of the polygons in point-contact and side-contact representations of subclasses of vertex-weighted planar graphs.

An upper bound of six for the minimum number of sides in a touching polygon representation of planar graphs (without weights) can be obtained from the vertex-to-side triangle contact graphs of de Fraysseix *et al.* [9], although this is not discussed in that paper. The top edge of each triangle can be converted into a raised 3-segment polyline, clipping the tips of the triangles touching it from above, thereby turning the triangles into side-touching hexagons. This approach might prove difficult for generating hexagonal representations as it involves computing the amounts by which each triangle may be raised so as to become a hexagon without changing any of the adjacencies. Moreover, the nature of such an algorithm would produce many "holes," potentially making such drawings less appealing, or requiring further modifications. Gonçalves *et al.* [14] describe a similar approach after presenting an algorithm to create primal-dual triangle contact representations, where each node and face are represented as triangles.

1.3 Preliminaries

Touching Hexagons Graph Representation Throughout this paper, we assume we are dealing with a connected planar graph $G = (V, E)$. We would like to construct a set of closed simple polygons R whose interiors are pairwise disjoint, along with an isomorphism $\mathcal{R} : V \to R$, such that for any two vertices $u, v \in V$, the boundaries of $\mathcal{R}(u)$ and $\mathcal{R}(v)$ overlap non-trivially if and only if $\{u, v\} \in E$. For simplicity, we adopt a convention of the cartogram community and define the *complexity* of a polygonal region as the number of sides it has. We call the set of all graphs having such a representation where each polygon in R has complexity 6 *touching hexagons graphs*.

Canonical Labeling Our algorithms begin by first computing a *planar embedding* of the input graph $G = (V, E)$ and using that to obtain a *canonical labeling* of the

vertices. A planar embedding of a graph is simply a clockwise order of the neighbors of each vertex in the graph. Obtaining a planar embedding can be done in linear time using the algorithm by Hopcroft and Tarjan [19]. The canonical labeling or order of the vertices of a planar graph was defined by de Fraysseix *et al.* [11] in the context of straight-line drawings of planar graphs on an integer grid of size $O(n) \times O(n)$. While the first algorithm for computing canonical orders required $O(n \log n)$ time [10], Chrobak and Payne [7] have shown that this can be done in $O(n)$ time.

In this section we review the canonical labeling of a planar graph as defined by de Fraysseix *et al.* [10]. Let $G = (V, E)$ be a fully triangulated planar graph embedded in the plane with exterior face u, v, w. A canonical labeling of the vertices $v_0 = u, v_1 = v, v_2, \ldots, v_{n-1} = w$ is one that meets the following criteria for every $2 < i < n$:

1. The subgraph $G_{i-1} \subseteq G$ induced by $v_0, v_1, \ldots, v_{i-1}$ is 2-connected, and the boundary of its outer face is a cycle C_{i-1} containing the edge (u, v);
2. The vertex v_i is in the exterior face of G_{i-1}, and its neighbors in G_{i-1} form an (at least 2-element) subinterval of the path $C_{i-1} - (u, v)$.

The canonical labeling of a planar graph G allows for the incremental placement of the vertices of G on a grid of size $O(n) \times O(n)$ so that when the edges are drawn as straight-line segments there are no crossings in the drawing. The two criteria that define a canonical labeling are crucial for the region creation step of our algorithm.

Kant generalized the definition for triconnected graphs, partitioning the vertices into sets V_1 to V_K that can be either singleton vertices or chains of vertices [22].

2 Lower Bound of Six Sides

In this section we show that at least six sides per polygon are sometimes needed in a touching polygons representation of a planar graph. We begin by constructing a class of planar graphs that cannot be represented by four-sided polygons and then extend the argument to show that the class also cannot be represented by five-sided polygons.

2.1 Four Sides Are not Enough

Consider the fully triangulated graph G^k in Fig. 2(a). It has three nodes on the outer face A, B and C, and contains a chain of nodes $1, \ldots, k$ which are all adjacent to A and B. Consecutive nodes in the chain, i and $i + 1$, are also adjacent. The remaining nodes of G^k are degree-3 nodes l_i and r_i inside the triangles $\Delta(A, i, i + 1)$ and $\Delta(B, i, i + 1)$.

Theorem 2 *For $k \geq 33$, there does not exist a touching polygons representation for G^k in which all regions have complexity four or less.*

Proof Assume, for the sake of contradiction, that we are given a touching polygons drawing for G^k in which all regions have complexity four or less. Without loss of generality, we assume that the drawing has an embedding that corresponds to the one shown in Fig. 2(a). Let Q_A, Q_B and Q_C denote the quadrilaterals representing nodes

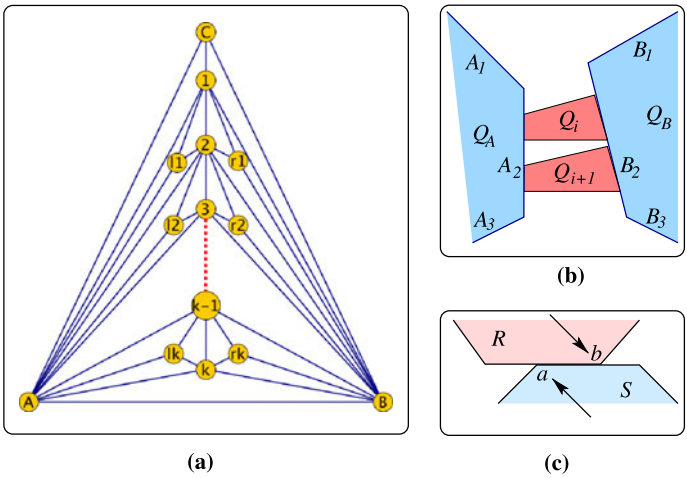

Fig. 2 (**a**) The graph G^k that provides the counterexample. (**b**) A pair of subsequent fair quadrilaterals adjacent to the same sides of Q_A and Q_B. (**c**) Illustration for Observation 2 shows one of three possible cases for two touching regions

A, B and C, and let Q_i denote the quadrilateral representing node i. Once again, without loss of generality, let Q_A lie in the left corner, Q_B in the right corner and Q_C at the top of the drawing.

We start with an observation.

Observation 1 Any corner of a quadrilateral can be adjacent to at most two disjoint quadrilaterals that (non-trivially) touch one of its sides. Since there are $c = 8$ corners of Q_A and Q_B, we have at most 16 quadrilaterals of the chain $Q_1, \ldots, Q_k$ that are adjacent to corners of Q_A and/or Q_B.

We now consider the quadrilaterals that are *not* adjacent to any of these corners.

Let Q_i be a quadrilateral that is not adjacent to any of the corners of the polygonal chains A_1, A_2, A_3, A_4 and B_1, B_2, B_3, B_4. Two of its corners are adjacent to the same side A_p of Q_A and the other two are adjacent to the same side B_q of Q_B, $1 \leq p, q \leq 4$. We call such a quadrilateral a *fair quadrilateral*.

Lemma 1 *For $k \geq 33$, in any touching quadrilaterals representation of G^k there exists a pair of fair quadrilaterals Q_i and Q_{i+1} that are adjacent to the same sides of Q_A and Q_B.*

Proof We can partition the set of fair quadrilaterals into 16 equivalence classes $C_{p,q}$, $1 \leq p, q \leq 4$, that denote the sets of fair quadrilaterals that are adjacent to the same sides of Q_A and Q_B. The equivalence class $C_{p,q}$ denotes that the pairs of sides (A_p, B_q) are used.

Observe that if Q_i is in an equivalence class C and Q_{i+1} is not fair, then since Q_{i+1} must be adjacent to a corner, Q_{i+2} cannot be in the equivalence class C. Thus, when we sweep through the chain of quadrilaterals $Q_1, \ldots, Q_k$, we simultaneously

proceed through the equivalence classes. By the pigeonhole principle, if there are at least 17 fair quadrilaterals, then at least two of them must be in the same equivalence class. Combining that with the fact that there are at most 16 quadrilaterals that are not fair completes our proof. □

Before continuing with the proof of Theorem 2, we include the following observation, partially illustrated in Fig. 2(c):

Observation 2 If there are two regions R, S touching in some nontrivial interval $I = (a, b)$ then at a, there is a corner of R or S. The same holds for corner b.

Using Observation 2, we see that each interval that is shared by two adjacent polygons ends at two of the corners of the two polygons. Now, let (Q_i, Q_{i+1}) be a pair of fair same-sided quadrilaterals, touching sides A_p and B_q. Since Q_i is fair, the two corners associated with the adjacency of Q_A must belong to Q_i and the other two corners of Q_i are associated with the adjacency with Q_B. The same applies for Q_{i+1}. Since Q_i and Q_{i+1} have to be adjacent, the two sides next to each other touch. From Observation 2, at least two corners of Q_i or Q_{i+1} are involved in the adjacency. For reference, label these two corners as c_1 and c_2. The quadrilateral Q_{l_i}, corresponding to node l_i, must touch quadrilaterals Q_A, Q_i and Q_{i+1}. If c_1 (or c_2) were also associated with an adjacency to Q_A then Q_{l_i} could not be adjacent to all three quadrilaterals simultaneously. Therefore, c_1 and c_2 must correspond to adjacencies with Q_B. A similar argument for Q_{r_i} shows that neither c_1 nor c_2 can correspond to adjacencies with Q_B either. However, this is a contradiction as all four corners of both quadrilaterals are either associated with the adjacency with Q_A or with Q_B. □

2.2 Five Sides Are not Enough

If we allow the regions to be pentagons, we must sharpen the argument a little more.

Lemma 2 *For $k \geq 71$, in any touching pentagons representation for G^k, there exists a triple of fair pentagons P_i, P_{i+1}, P_{i+2} adjacent to the same sides of P_A and P_B.*

Proof We prove this along the same lines as the proof for Lemma 1. As before, we can see that for a total of 10 corners, at most 20 pentagons of the inner chain are not fair. The number of equivalence classes of pentagons with sides solely on the same side of P_A and P_B is at most $5 \times 5 = 25$. Recall that pentagons belonging to the same equivalence class are sequential. Since we aim now for triples and not just for pairs, using the pigeonhole principle, if we have more than $2 \times 25 = 50$ fair pentagons at least three must belong to the same equivalence class. Therefore, as long as $k > 20 + 50 = 70$, there exists a triple of fair same-sided pentagons. □

Theorem 3 *For $k \geq 71$, there does not exist a touching polygons representation for G^k in which all regions have complexity five or less.*

Proof From Lemma 2, let (P_i, P_{i+1}, P_{i+2}) be a triple of fair same-sided pentagons, touching sides A_p and B_q. From Observation 2, we know that each interval that is

shared by two polygons ends at two of the corners of the two polygons. Consequently, four of the five corners for P_i, P_{i+1} and P_{i+2} are adjacent to A_p or B_p. Since P_i and P_{i+1} have to be adjacent, the two sides next to each other touch. However, since there exist the polygonal regions representing r_i and l_i, as before, the interval where P_i and P_{i+1} touch is disjoint from the regions P_A and P_B. As each region can have at most five corners, four of which are adjacent to either P_A or P_B, from Observation 2 we know that one corner from the adjacency with P_i and P_{i+1} belongs to P_i and one belongs to P_{i+1}. Similarly, we know that the adjacencies of r_{i+1} and l_{i+1} imply that one corner of the adjacency of P_{i+1} and P_{i+2} belongs to P_{i+1} and the other belongs to P_{i+2}. Due to planarity, we also know that P_i and P_{i+2} lie on opposite sides of P_{i+1}. As these corners cannot be adjacent to P_A or P_B, we see that P_{i+1} must have six distinct corners, two adjacent to P_A, two to P_B, one to P_i and one to P_{i+2}, a contradiction. □

Note that six-sided polygons are indeed sufficient to represent the graph in Fig. 2(a). In particular, for fair polygons P_i and P_{i+1}, we can use three segments on the lower side of P_i, while the upper side of P_{i+1} consists of only one segment completely overlapping the middle of the three segments from the lower side of P_i.

3 Touching Hexagons Representation

In this section, we present a linear-time algorithm that takes as input a planar graph $G = (V, E)$ and produces a representation of G in which all regions are convex hexagons. This algorithm and the fact that every touching hexagons graph is necessarily planar proves that the class of planar graphs is equivalent to the class of touching hexagons graphs.

3.1 Algorithm Overview

We assume that the input graph $G = (V, E)$ is a fully triangulated planar graph with $|V| = n$ vertices. If the graph is planar but not fully triangulated, we can augment it to a fully triangulated graph with the help of dummy vertices and edges, run the algorithm below and remove the polygons that correspond to dummy vertices.

Traditionally, planar graphs are augmented to fully triangulated graphs by adding edges to each non-triangular face. Were we to take this approach, however, when we remove the dummy edges we would have to perturb the resulting space partition to remove polygonal adjacencies. As this is difficult to do, we convert our input graph to a fully triangulated one by adding one additional vertex to each face and connecting it to all vertices in that face. The above approach works if the input graph is biconnected. Singly-connected graphs must first be augmented to biconnected graphs as follows. Consider any articulation vertex v, and let u and w be consecutive neighbors of v in separate biconnected components. Add a new vertex z and the edges (z, u) and (z, w). Iterating for every articulation point biconnects G and results in an embedding in which each face is bounded by a simple cycle. Since determining articulation

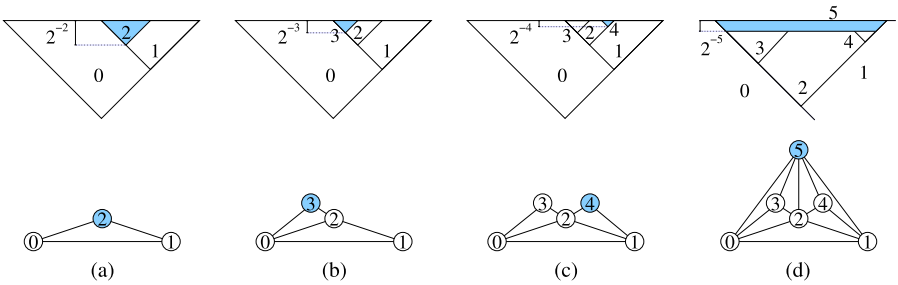

Fig. 3 Incremental construction of the touching hexagons representation of a graph. *Shaded vertices on the bottom row* and *shaded regions on the top row* are processed at each step. In general, the region defined at step i is carved at distance 2^{-i} from the active front *on the top*. Note that the *top row* forms a horizontal line at all times

points and adding vertices and edges to faces can be done in linear time, the augmentation step incurs only a linear amount of additional time to the main algorithm and adds at most a linear number of vertices and edges to the original graph.

The algorithm has two main phases. The first phase computes the canonical labeling. In the second phase we create regions with slopes 0, 1, -1 out of an initial isosceles right-angle triangle, by processing vertices in the canonical order. Each time a new vertex is processed, a new region is carved out of one or more already existing regions. At the end of the second phase of the algorithm we have a right-angle isosceles triangle that has been partitioned into exactly $n = |V|$ convex regions, each with at most 6 sides. We show that creating and maintaining the regions requires linear time in the size of the input graph. We illustrate the algorithm with an example; see Fig. 3.

3.2 Region Creation

In this section we describe the n-step incremental process of inserting new regions in the order given by the canonical labeling, where $n = |V|$. The regions will be carved out of an initial triangle with coordinates $(0, 0)$, $(-1, 1)$, $(1, 1)$. The process begins by the creation of R_0, R_1, and R_2, which correspond to the first three vertices, v_0, v_1, v_2; see Fig. 3(a). Note that the first three vertices in the canonical order form a triangular face in G and hence must be represented as mutually touching regions.

At step i of this process, where $2 < i < n$, region R_i will be carved out from the current set of regions. Define a region as "active" at step i if it corresponds to a vertex that has not yet been connected to all its neighbors. An invariant of the algorithm is that all active regions are non-trivially tangent to the top side of the initial triangle, which we refer to as the "active front."

By criterion 2 of the canonical labeling and the active regions invariant, the current node v_i is connected to two or more consecutive vertices on the outer face of G_{i-1} and consecutive regions on the active front. Let v_a and v_b be the leftmost and rightmost neighbors of v_i on the outer face with corresponding (active front) faces R_a and R_b. The new region R_i is defined to be an isosceles trapezoid formed by carving a horizontal line segment that is at distance $1/2^i$ from the active front and intersects

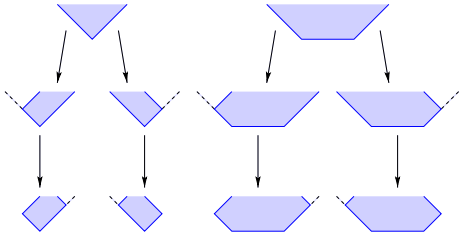

Fig. 4 There are a ten possible region shapes, falling into three categories: 2 opening, 4 static, and 4 closing. The arrows indicate carvings from one region to another

the right side of R_a and the left side of R_b. The left (respectively, right) side of the trapezoid has slope -1 (respectively, $+1$). If the right side of R_a has slope $+1$, a portion of its region is necessarily carved out by R_i. The same applies if the left side of R_b has slope -1. The regions between R_a and R_b have their upper segment carved and no longer being tangential to the active front are removed from the set of active regions. In addition, R_i is added to the list of active regions. In Fig. 3(d), for example, both R_0 and R_1 have appropriate slopes and so are not carved and R_2, R_3, and R_4 are all removed from the active front.

Note, that if $d_{G_i}(v_i) = 2$, then the length of the horizontal segment is 0 and the shape is an isosceles triangle. In this case, the geometry is such that exactly one of R_a or R_b must necessarily be carved. See Figs. 3(a–c).

Lemma 3 *The above algorithm produces convex regions with at most* 6 *sides.*

Proof The convexity of the regions is obvious from the fact that regions are created by a (linear) partitioning cut of a previous convex shape. Note that the above algorithm leads to the creation of at most ten different types of regions; see Fig. 4. Each region has a horizontal top segment, a horizontal bottom segment (possibly of length 0), and sides with slopes -1 or 1. Moreover, each region can be characterized as either opening (the first two in top row), static (the next four in the middle row), or closing (the last four in the bottom row), depending on the angles of the two sides connecting it to the top horizontal segment. At each iteration, each new region R_i is an opening region. In the new region's creation, all affected regions except for R_a and R_b are carved with a horizontal line segment lying just below the top segment thus having no effect on the shapes of these regions and removing them from the active front. Consequently, the only new region shapes possible stem from cutting R_a and R_b when necessary with a slope -1 or $+1$ line respectively. As the lower vertex formed by each cut is at least half the distance to the active front from the previous vertices (those not on the active front), the only possible shapes are those shown in Fig. 4. Observe that closing regions cannot be carved at all.

□

3.3 Running Time

The above algorithm can be implemented in linear time. We provide here a simple description; further details can be found in [8].

Note that both a planar embedding [19] and a canonical labeling of the embedding [7] for a planar graph can be computed in linear time. The remainder of the al-

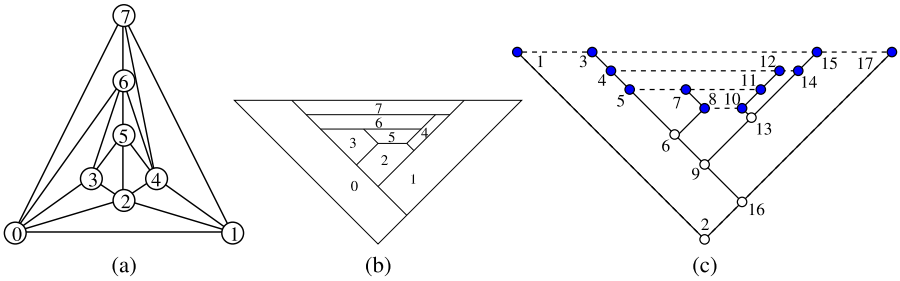

Fig. 5 An example of a capped binary tree. (**a**) The original graph. (**b**) The hexagonal representation (not drawn to scale to conserve space). (**c**) The corresponding capped binary tree with *shaded nodes* representing cap nodes and *dashed lines* representing capped sets. The nesting of all capped sets is (1, 3, (4, (5, 7, (8, 10), 11), 12, 14), 15, 17). We draw our binary tree upwards with the root at the bottom to correlate better with the hexagonal drawing algorithm

gorithm's time is spent in creating and maintaining the regions. Recall that each new region is created by carving out an area from a set of other regions in the outer face. The creation of each region can be done in constant (amortized) time by charging the process to the regions that are carved. Thus, we only need to bound the number of times a region can be modified. As can be seen from the acyclic hierarchy of regions in Fig. 4, there is a limited set of possible shapes that a region can take as it is carved. Including the final horizontal cut removing the region from the outer face, each region can be modified, and hence charged, at most three times. Noting that each region corresponds to a unique node, we obtain the following lemma:

Lemma 4 *For any planar graph G on n vertices, we can construct in linear time a touching hexagons representation of G with convex regions. Moreover, if the graph is a triangulation, the representation is also a tiling.*

4 Compaction Algorithm for Quadratic Area

The algorithm given in Sect. 3.2 provides a touching hexagons representation of any planar graph. The incremental process carves out polygons within an ever smaller band of active front, therefore in practice the drawing is highly skewed, leading to exponential area. In this section we describe a compaction algorithm to get a drawing on an $O(n) \times O(n)$ grid.

When looking at the vertices and edges created in the algorithm for touching hexagons, if the horizontal edges are ignored, then the resulting graph is a "binary" tree, in the sense that each vertex has a degree of no more than 2. See Fig. 5. From this observation, we can generalize the compaction problem to the tree drawing routine described below.

We start with some definitions. Order the nodes according to their inorder traversal. A *cap set* is an ordered subset of the nodes such that

1. The first (resp. last) node has exactly one child, the left (resp. right) child.
2. All other nodes are leaf nodes, with the addition that for the outermost cap the first and last nodes are also leaf nodes.
3. The ordering of nodes in the cap set follows the same inorder traversal ordering.
4. Any two cap sets are non-overlapping. However, one may be nested in another, in the sense that if one set C goes from node a to node b and is contained in a second set C' then there exist two *consecutive* nodes i, j in C' such that $i < a < b < j$.

Define a *capped binary tree* as a binary tree where every node either has two children (proper) or is assigned to a specific cap set. For convenience, we often refer to individual *cap nodes* in a cap set or to *cap node pairs* (u, v) of neighboring (consecutive) cap nodes in a cap set.

Figure 5 illustrates the correlation to the hexagons created by the algorithm in Sect. 3.2 and provides an example of a capped binary tree, where the nodes in the tree represent the vertices formed in the drawing (not the hexagon faces), the edges are precisely the (non-horizontal) edges of the drawing, and each cap set is a maximal connected component of vertices and horizontal edges of the drawing.

The *capped binary tree drawing problem* is to take a capped binary tree and draw it on an integer grid such that 1) there are no edge crossings except at common endpoints; 2) each right (resp. left) edge is drawn with a slope of $+1$ (resp. -1); and 3) all nodes in a cap set are drawn with the same y-coordinate such that they can be connected by a horizontal line segment without crossing any other edges (except at the nodes in the set).

Before proving that we can draw capped binary trees on an $n/2 \times n/2$ grid, where n is the number of nodes in the tree, we first present a divide-and-conquer compaction algorithm to accomplish this. The algorithm is inspired by the layered tree drawing algorithm [34] with the additional aforementioned constraints.

Let G_{T6G} be the graph derived from the touching hexagons algorithm, formed by taking the vertices as the intersections of the regions and the edges as (portions of) the sides of the regions connecting the vertices; see Fig. 5(c) where the edges are both the solid and dashed lines. Let G_T be the corresponding capped binary tree, formed by removing the horizontal edges. Our compaction algorithm, described in detail below, proceeds by incrementally removing and placing a subset of the leaf nodes from a subtree G_c initially set to G_T. For each of the leaf nodes of G_c removed, the resulting placement requires the node to be connected to all of its child nodes (if any) in the original tree G_T and also might require adjusting the position of one of its subtrees. Our process works by only adjusting the horizontal positions of any node, thus preserving a subtree's vertical position. For performance reasons, we actually delay the horizontal shifting by merely recording the shift needed for a subtree in its root. The shifts are then propagated through the tree in a final post-processing stage. Initially, the x and y positions as well as the horizontal shift of every node is set to 0.

Before proceeding with the details of the algorithm, we clarify precisely those leaf nodes that are removed and placed at each iteration. Define the *active front node set* F of G_c as the maximum subset of leaf nodes of G_c, such that a cap node is in F if and only if all the nodes in its cap set are also in F. The initial active front is precisely those vertices at the upper edge of the outer triangular region. For example, in Figure 5(c), this would be the set $(1, 3, 15, 17)$.

1. For each node v in the active front node set F of G_c,
   a) if v is a leaf in G_T, we do nothing (v remains at $(0, 0)$).
   b) if v has one subtree in G_T and if it is to the right, extend a slope $+1$ line from the root of this right subtree by 1 unit down and left to get the position of v. If it instead has a left child, extend a slope -1 line, down and right.
   c) if v has two subtrees in G_T, shift the right subtree horizontally so that the two subtrees have a "separation"[1] of either distance 1 or 2, and the slope -1 (resp. $+1$) line from the root of the left (resp. right) subtree meet at a grid point, the assigned point for v. Record the shift used at the root of the right tree.
2. For each cap set C in the front, set h to be the maximum of the absolute values of y coordinates of the cap nodes in C. For every cap node $v \in C$,
   a) if v is a leaf node in G_T, set $y(v) = -h$;
   b) otherwise, by construction, node v must have only one subtree. If it is to the right, extend the slope $+1$ line from the root of the subtree till it intersects with the line $y = -h$, and record the coordinates of the intersection point as the coordinates for v. If it is to the left, extend the slope -1 line instead.
3. Delete F and its connecting edges from G_C, renaming the resulting tree G_C. If G_c is not empty, go to Step 1.
4. Propagate the horizontal shifts from each node to its subtree via a pre-order traversal starting at the root of G_T to obtain a final integer grid position for each node.

This algorithm yields a drawing of the G_{T6G} on a grid. In [8], we present a detailed technical execution of this algorithm on the graph from Fig. 5. Because the algorithm processes entire cap sets at a time, because Step 2 places all nodes in the same cap set at the same (lowest) height, and because it only shifts nodes horizontally, all cap nodes are drawn at the same vertical position. Further, because the algorithm also only connects the tree edges using line segments with slope ± 1 and applies any horizontal shifts to the entire subtree via the final propagation step, all tree edges are drawn with slopes ± 1. Consequently, the drawing produced by this algorithm is a valid capped binary tree drawing.

We also need to show that the grid size used is reasonable. To bound the area, we must first elaborate on the compaction step (Step c) that combines two trees such that their separation is either distance 1 or 2. This separation is not between the two roots of the subtrees but between the closest two nodes. In essence, we wish to compact the two subtrees as close as possible. For clarity and simplicity, we present here a simple linear-time compaction algorithm leading to an overall quadratic performance. Using only doubly-connected circular linked lists, it is not difficult to improve the performance to an amortized constant time yielding the necessary linear-time bound. The details of the improved version can be found in [8].

Figure 6 illustrates the process. We initially place the two subtrees to be merged at a sufficient distance apart. For each y-value in the grid, we determine the difference between the rightmost node in the left subtree and the leftmost node in the right subtree (if either subtree does not have such a node, we take the difference to be infinite). Let m be the smallest such difference. We shift the right subtree horizontally

[1] We elaborate on what separation entails shortly.

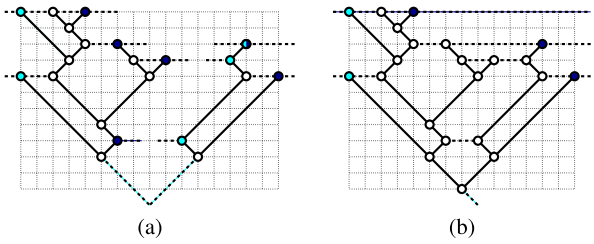

Fig. 6 An example of compacting two subtrees together during Step 1c. (**a**) The two subtrees at an initial separation, highlighting cap nodes whose neighbors are in a different subtree. (**b**) The resulting tree after merging, with new cap node pairs

$m-1$ or $m-2$ units so that the root of the two subtrees meet at a grid point. Clearly this can be done in linear time.

The observation needed for the faster version is that the minimum difference must occur at specific cap nodes, whose neighboring cap node is in the opposite subtree. Maintaining only this set of potential cap nodes can be achieved via linked lists and by zipping up neighboring cap nodes during the merge process from the root to the leaf nodes of the subtrees.

Lemma 5 *Given any capped binary tree T, we can compute in linear time a capped binary tree drawing of T on an $(n-1) \times (n-1)/2$ grid.*

Proof The linear-time algorithm comes directly from the above discussion and the improved constant amortized time compaction step. However, it still remains to prove that the resulting drawing is sufficiently compact. We do this by inductively analyzing the separation between neighboring cap nodes, which are "joined" during the process described above. This proof is reminiscent of the one given by Kant [21].

For every cap node pair (u, v), consecutive nodes in a cap set, let its *interior cap set* C_i be the cap set (if one exists) whose first node is the next cap node in the inorder traversal of T from u. Note that from the definition of the nesting of cap sets, the last node in C_i would be the last cap node before v. If no such set exists, then let C_i be $\{\text{lca}(u, v)\}$, where $\text{lca}(u, v)$ represents the least common ancestor of u and v. If u has a child node, it must be a left child and if v has a child node, it must be a right child. Since any subtree necessarily has at least one cap node (a leaf of that subtree), this lca is also the only node in the inorder traversal between u and v. We refer to u and v as the *exterior cap nodes* of C_i. Observe that the interior cap set will have its y-coordinate value closer to the root in the final drawing. For example, in Fig. 5(c), the cap node pair $(4, 12)$ has the interior cap set $\{5, 7, 11\}$, whereas the cap node pair $(12, 14)$ has as the interior cap set the lca $\{13\}$.

Our proof uses the following inductive claim. After every iteration of our algorithm, for any subtree T' and any cap node pair (u, v) in T' that is part of a cap set whose exterior cap nodes are not also in T' or is the outermost cap set, the (horizontal) distance between (u, v) is no more than twice the number of cap node pairs in the inorder traversal of T', inclusive of the pair (u, v).

If this claim holds, then the drawing is on an $(n-1) \times (n-1)/2$ grid because the final drawing is a single tree with one row of cap nodes at the top whose width cannot exceed twice the number of cap node pairs in the tree. To bound the number of cap node pairs, notice that the graph formed by the binary tree edges combined with the horizontal (capping) edges forms a 3-regular graph, excluding the root and leftmost

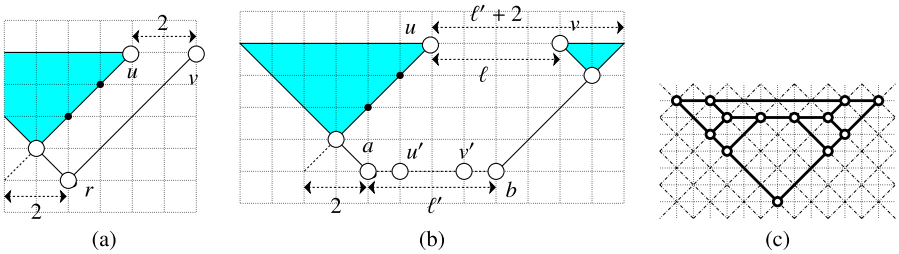

Fig. 7 Example of the merging of two subtrees during successive cap node pairs for (**a**) a single lca node and (u, v) and (**b**) (u', v') and (u, v). (**c**) A simple drawing of a capped binary tree highlighting both the normal grid (*horizontal/vertical lines*) and the rotated, space-efficient, grid (*diagonal lines*)

and rightmost vertices which have degree two. Since this graph has $(3n - 3)/2$ edges and $n - 1$ of the edges are tree edges, that leaves exactly $(n - 1)/2$ horizontal edges. Each horizontal edge corresponds to a unique cap node pair. The height follows from the ± 1 slope of the non-horizontal edges.

Initially, every node is in its own subtree so the claim holds. Inspecting the algorithm reveals that the only place where the claim could change is in Step 1c where two subtrees T_1 and T_2 are merged. In addition, since the trees are simply shifted to merge, the only possible change is due to the introduction of new cap node pairs, a cap node from each subtree is aligned with its neighbor in the other tree. In fact, since the merging process zips nested cap sets in succession, we are only concerned with the final width of the last cap pair merged. We again prove this by induction on the zipping process.

We claim that the width of the cap node pair (u, v) merged is no more than twice the number of cap node pairs in its inorder traversal from u to v. Let (u, v) be the first cap node pair merged. Since the interior cap set of (u, v) is simply r, which is the only node in the inorder traversal between u and v, the resulting width at this stage is at most 2; see Fig. 7(a). Thus, our claim holds after the first merged cap node pair.

We now progress inductively. Let (u, v) be the next cap node pair merged with width ℓ, (u', v') be the previous pair, and C be the interior cap set of (u, v). Notice that $(u', v') \in C$. By induction, we know that the entire width ℓ' of C is no more than twice the number of cap node pairs in the inorder traversal from the first to last cap nodes in C. In addition, since u has no right subtree and v has no left subtree, the next cap node in the inorder traversal is the first cap node in C and the last in the traversal is the last cap node in C. Therefore, we know that the number of cap node pairs in the inorder traversal from u to v is the same as the number for C plus one, the one for (u, v). Therefore, we need only to prove that $\ell \leq \ell' + 2$.

Let a and b be the first and last cap node in C; see Fig. 7(b). By the definition of a capped binary tree, we know that a and b each have only one child, a left and a right respectively. In addition, by Step 2 we know that one of the two child nodes is only one unit above its parent. Without loss of generality assume it is the left child of a. This means that the node is also one unit to the left of a. Node u is a descendant of this left child but from the definition of the capped binary tree, u can be found by traversing successive right children only. Therefore, the path from this left child to u follows a straight line of slope $+1$. This follows parallel with the line from b through

its right child. The (horizontal) distance from u to this line is exactly $\ell' + 2$. Since the path from this right child to v follows left children only (if any), the distance from u to v is $\ell \leq \ell' + 2$. This completes our proof. □

For a clearer understanding and better symmetry with the construction technique used in Sect. 3, we used edges with slopes ± 1 and 0. As Fig. 7(c) illustrates, by using a grid that is rotated 45° producing tree edges that are drawn rectilinear and capped edges drawn with slope -1, we can improve the area bound slightly.

Corollary 1 *Given any capped binary tree T, we can compute in linear time a (rotated) capped binary tree drawing of T on an $(n-1)/2 \times (n-1)/2$ grid.*

Since the initial construction step does not need to create the hexagons explicitly, that step can be used simply to determine the combinatorial representation of the capped binary tree. This prevents any issues with numerical precision and representation. Combining Lemmas 4 and 5 yields our first proof for Theorem 1.

5 Another Hexagonal Representation Using $O(n) \times O(n)$ Area

In this section, we present an alternative approach to proving Theorem 1. This approach is based on Kant's algorithm for hexagonal grid drawings of 3-connected, 3-regular planar graphs [21]. Although the modification needed is direct, we feel that our previous approach is a more intuitive and constructive technique that yields better fundamental insight into the nature of the problem.

In Kant's algorithm the drawing is obtained by looking at the dual graph and processing its vertices in the canonical order. In the final drawing, however, there are two non-convex faces, separated by an edge not drawn as a straight-line segment. We address these problems by adding some extra vertices in a pre-processing step. Once the dual of this augmented graph is embedded, the faces corresponding to the extra vertices can be removed to yield the desired $O(n) \times O(n)$ grid drawing.

Let $H = (V, E)$ be a 3-connected, 3-regular planar graph. Note that the dual $D(H)$ is fully triangulated, as each face in the dual corresponds to exactly one vertex in H. So, for f faces in H, we have f vertices in $D(H)$. We first compute a canonical ordering on the vertices of $D(H)$ as defined by de Fraysseix *et al.* [9]. Let $v_1, \ldots, v_f$ be the vertices in $D(H)$ in this canonical order.

Kant's algorithm now constructs a drawing for H on the hexagonal grid such that all edges but one have slopes 0°, 60° or −60°, with the one edge with bends lying on the outer face. The typical structure of those drawings is shown in Fig. 8(a). Although we focus our description using the hexagonal grid, to place the nodes on the rectilinear grid, the corresponding slopes are 0°, 90° and −45°.

The algorithm incrementally constructs the drawing by adding the faces of H in reverse order of the canonical order of the corresponding vertices in $D(H)$. We let w_i be the vertices of H. Let face F_i correspond to vertex v_i in $D(H)$. The algorithm starts with a triangular region for the face F_f that corresponds to vertex v_f. The vertex w_x that is adjacent to F_f, F_1 and F_2 is placed at the bottom. Let w_y and w_z

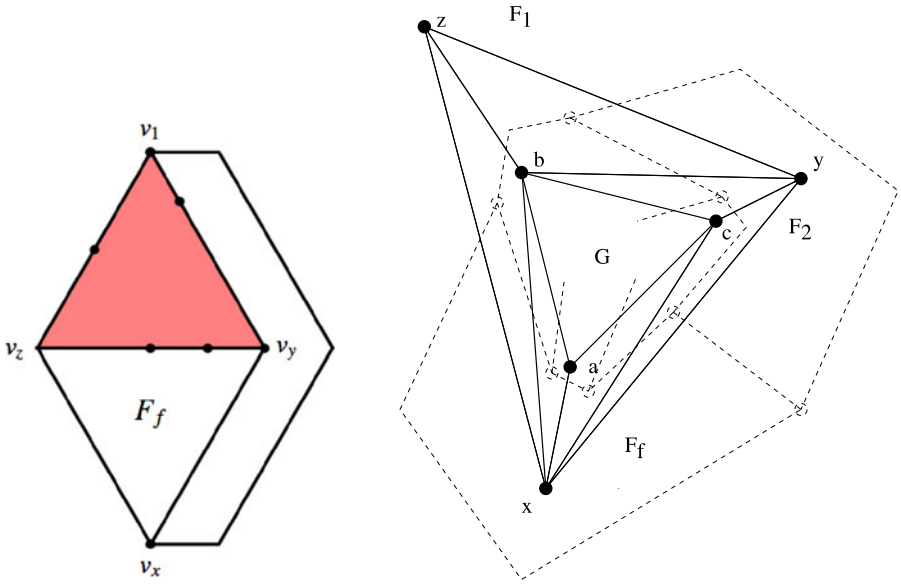

Fig. 8 (**a**) Polygonal structure obtained from Kant's algorithm. (**b**) Graph G augmented by vertices x, y and z together with its dual which serves as the input graph for Kant's algorithm

be the neighbors of w_x in F_f. These three vertices form the corners of the first face F_f. (w_x, w_z) and (w_x, w_y) are drawn upward with equal lengths and slopes $-\sqrt{3}$ and $\sqrt{3}$, respectively. All the edges on the path between w_y and w_z along F_f are drawn horizontally between the two vertices. From this first triangle, all other faces are added in reverse canonical order to the upper boundary of the drawing region. If a face is completed by only one vertex w_i, this vertex is placed appropriately above the upper boundary such that it can be connected by two edges with slopes $-\sqrt{3}$ and $\sqrt{3}$, respectively. If the face is completed by a path, then the two end segments of the path have slopes $-\sqrt{3}$ and $\sqrt{3}$, while the other edges are horizontal. The construction ends when w_1 is inserted, corresponding to the outer face F_1. Note that there is an edge between w_1 and w_x, which is drawn using some bends. This edge is adjacent to the faces F_1 (the outer face) and F_2.

From this construction, we can observe that the angles at faces $F_f, \ldots, F_3$ have size $\leq 180°$ as the first two edges do not enter the vertex from above, and the last edge leaves the vertex upwards. Hence, we have the following result.

Lemma 6 *The faces $F_f, \ldots, F_3$ are convex, and as the slopes of the edges are $\pm\sqrt{3}$ or 0, they are drawn with at most six sides.*

This property is exactly what we are aiming for, as the vertices of our input graph G should be represented by convex regions of at most six sides. Unfortunately, Kant's algorithm creates two non-convex faces F_1 and F_2 separated by an edge which is

not drawn as a line segment. Furthermore, the face F_f is drawn as large as all the remaining faces $F_3, \ldots, F_{f-1}$ together.

Kant gave an area estimate for the result of his algorithm which is the same for both hexagonal and rectilinear grids. A corollary of Kant's algorithm is the following:

Corollary 2 *For a given* 3*-connected*, 3*-regular planar graph* H *of* n *vertices*, $H - w_x$ *can be drawn within an area of* $(n/2-1) \times (n/2-1)$.

To apply Kant's result to the problem of constructing a touching hexagons representation, we enlarge the embedded input graph G so that the dual of the resulting graph G' can be drawn using Kant's algorithm in such a way that the original vertices of G correspond to the faces $F_3, \ldots, F_{f-1}$.

We add 3 vertices corresponding to faces F_1, F_2 and F_f in Kant's algorithm. Since G is fully triangulated, let a, b and c be the vertices at the outer face of G in clockwise order. We add the vertices x, y and z in the outer face and connect to G so that z corresponds to the outer face F_1, y to F_2, and x to F_f. First, we add x and connect it to a, b and c such that b and c are still in the outer face. Then we add y and connect it to x, b and c such that b is still in the outer face. Finally, we add z and connect it to x, y and b such that x, y and z now form the outer face; see Fig. 8(b).

Since the vertices x, y and z are on the outer face, we can choose which one is first, second and last in the canonical order. We then apply Kant's algorithm with the canonical order $v_1 = z$, $v_2 = y$ and $v_f = x$. After construction, we remove the regions corresponding to vertices x, y and z, yielding a hexagonal representation of G.

Given any (connected) planar graph G, we can make it fully triangulated using the technique described in Sect. 3.1. We can then remove the added vertices and edges. Since Kant's algorithm runs in linear time, and our emendations can be done in linear time, we get another proof for Theorem 1. We again use at most three slopes for each representation with sides having slopes $\pm\sqrt{3}$ or 0 (or 0, $+\infty$ and -1).

For a triangulated input graph $G = (V, E)$, we have n vertices and, by Euler's formula, $2n-4$ faces. Since we enhanced our graph to $n+3$ vertices, we have $f = 2n+2$ faces. Those faces are the vertices in the dual $D(G)$ which is the input to Kant's algorithm. His area estimation gives an area of $(f/2-1) \times (f/2-1)$ for f vertices when we coalesce the faces F_1, F_2 and F_f into a single outer face by removing the corresponding vertices and edges. Thus, we get an area bound of $n \times n$ using exactly the same argument as Kant [21].

6 Conclusion and Future Work

Thomassen [32] had shown that not all planar graphs can be represented by touching pentagons, where the external boundary of the figure is also a pentagon and there are no holes. Our results are more general, as we do not insist on the external boundary being a pentagon or on there being no holes between pentagons. It is possible to derive algorithms for convex hexagonal representations for general planar graphs from several earlier papers, e.g., de Fraysseix *et al.* [9], Thomassen [32], and Kant [21]. However, these do not immediately lead to algorithmic solutions to the problem of

computing a graph representation with convex low-complexity touching polygons. To the best of our knowledge, this problem has never been formally considered.

In this paper, we presented several results about touching k-sided graphs. We showed that, for general planar graphs, six sides are necessary and sufficient, and that the algorithm for creating a touching hexagons representation can be modified to yield an $O(n) \times O(n)$ drawing area. Finally, we discussed a different algorithm for general planar graphs which yields a similar drawing area.

Several interesting related problems are open. What is the complexity of deciding whether a given planar graph can be represented by touching triangles, quadrilaterals, or pentagons? In the context of rectilinear cartograms, the vertex-weighted problem has been carefully studied. However, the same problem without the rectilinear constraint has received less attention. Finally, it would be interesting to characterize the subclasses of planar graphs that allow for touching triangles, touching quadrilaterals, and touching pentagons representations.

Acknowledgements We would like to thank Therese Biedl for pointing out the very relevant work by Kant and Thomassen and the anonymous referees for their helpful and thoughtful comments.

References

1. Alam, M.J., Biedl, T., Felsner, S., Kaufmann, M., Kobourov, S.G.: Proportional contact representations of planar graphs. Tech. Rep. CS 2011-11, 2011, http://www.cs.uwaterloo.ca/research/tr/2011/CS-2011-11.pdf
2. Battista, G.D., Lenhart, W., Liotta, G.: Proximity drawability: a survey. In: Graph Drawing. Lecture Notes in Computer Science, vol. 894, pp. 328–39. Springer, Berlin (1994)
3. de Berg, M., Mumford, E., Speckmann, B.: On rectilinear duals for vertex-weighted plane graphs. Discrete Math. **309**(7), 1794–1812 (2009)
4. Biedl, T., Bretscher, A., Meijer, H.: Rectangle of influence drawings of graphs without filled 3-cycles. In: Graph Drawing. Lecture Notes in Computer Science, vol. 1731, pp. 359–368. Springer, Berlin (1999)
5. Bruls, M., Huizing, K., van Wijk, J.J.: Squarified treemaps. In: Proceedings of the Joint Eurographics/IEEE TVCG Symposium on Visualization, VisSym, pp. 33–42 (2000)
6. Buchsbaum, A.L., Gansner, E.R., Procopiuc, C.M., Venkatasubramanian, S.: Rectangular layouts and contact graphs. ACM Trans. Algorithms **4**(1), 8:1–8:28 (2008)
7. Chrobak, M., Payne, T.: A linear-time algorithm for drawing planar graphs. Inf. Process. Lett. **54**, 241–246 (1995)
8. Duncan, C.A., Gansner, E.R., Hu, Y.F., Kaufmann, M., Kobourov, S.G.: Optimal polygonal representation of planar graphs. ArXiv e-prints, Apr. 2011, arXiv:1104.1482
9. de Fraysseix, H., de Mendez, P.O., Rosenstiehl, P.: On triangle contact graphs. Comb. Probab. Comput. **3**, 233–246 (1994)
10. de Fraysseix, H., Pach, J., Pollack, R.: Small sets supporting Fary embeddings of planar graphs. In: Proceedings of the 20th Symposium on the Theory of Computing (STOC), pp. 426–433 (1988)
11. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**(1), 41–51 (1990)
12. Gabriel, K.R., Sokal, R.R.: A new statistical approach to geographical analysis. Syst. Zool. **18**, 54–64 (1969)
13. Gansner, E.R., Hu, Y., Kobourov, S.: On touching triangle graphs. In: Graph Drawing. Lecture Notes in Computer Science, vol. 6502, pp. 250–261. Springer, Berlin (2011), http://dx.doi.org/10.1007/978-3-642-18469-7_23
14. Gonçalves, D., Lévêque, B., Pinlou, A.: Triangle contact representations and duality. In: Graph Drawing. Lecture Notes in Computer Science, vol. 6502, pp. 262–273. Springer, Berlin (2011). http://dx.doi.org/10.1007/978-3-642-18469-7_24

15. He, X.: On finding the rectangular duals of planar triangular graphs. SIAM J. Comput. **22**(6), 1218–1226 (1993)
16. He, X.: On floor-plan of plane graphs. SIAM J. Comput. **28**(6), 2150–2167 (1999)
17. Hliněný, P.: Classes and recognition of curve contact graphs. J. Comb. Theory, Ser. B **74**(1), 87–103 (1998)
18. Hliněný, P., Kratochvíl, J.: Representing graphs by disks and balls (a survey of recognition-complexity results). Discrete Math. **229**(1–3), 101–124 (2001)
19. Hopcroft, J., Tarjan, R.E.: Efficient planarity testing. J. ACM **21**(4), 549–568 (1974)
20. Jaromczyk, J.W., Toussaint, G.T.: Relative neighborhood graphs and their relatives. Proc. IEEE **80**, 1502–1517 (1992)
21. Kant, G.: Hexagonal grid drawings. In: 18th Workshop on Graph-Theoretic Concepts in Computer Science, pp. 263–276 (1992)
22. Kant, G.: Drawing planar graphs using the canonical ordering. Algorithmica **16**, 4–32 (1996) (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia)
23. Kant, G., He, X.: Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. Theor. Comput. Sci. **172**, 175–193 (1997)
24. Koebe, P.: Kontaktprobleme der konformen Abbildung. Ber. Verh. K. Sächs. Ges. Wiss. Leipz., Math.-Phys. Kl. **88**, 141–164 (1936)
25. Koźmiński, K., Kinnen, W.: Rectangular dualization and rectangular dissections. IEEE Trans. Circuits Syst. **35**(11), 1401–1416 (1988)
26. Lai, Y.-T., Leinwand, S.M.: Algorithms for floorplan design via rectangular dualization. IEEE Trans. Comput.-Aided Des. **7**, 1278–1289 (1988)
27. Lai, Y.-T., Leinwand, S.M.: A theory of rectangular dual graphs. Algorithmica **5**, 467–483 (1990)
28. Liao, C.-C., Lu, H.-I., Yen, H.-C.: Compact floor-planning via orderly spanning trees. J. Algorithms **48**, 441–451 (2003)
29. Liotta, G., Lubiw, A., Meijer, H., Whitesides, S.H.: The rectangle of influence drawability problem. Comput. Geom. Theory Appl. **10**, 1–22 (1998)
30. Rahman, M., Nishizeki, T., Ghosh, S.: Rectangular drawings of planar graphs. J. Algorithms **50**(1), 62–78 (2004)
31. Steadman, P.: Graph-theoretic representation of architectural arrangement. In: The Architecture of Form, pp. 94–115. Cambridge University Press, Cambridge (1976)
32. Thomassen, C.: Plane representations of graphs. In: Progress in Graph Theory, pp. 43–69 (1982)
33. Thomassen, C.: Interval representations of planar graphs. J. Comb. Theory, Ser. B **40**, 9–20 (1988)
34. Walker, J.Q., II: A node-positioning algorithm for general trees. Softw. Pract. Exp. **20**(7), 685–705 (1990)
35. Yeap, G.K., Sarrafzadeh, M.: Sliceable floorplanning by graph dualization. SIAM J. Discrete Math. **8**(2), 258–280 (1995)