Where to Draw the Line

Ashim Garg
Ph.D. Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-96-13
Aprill 1996

Where to Draw the Line

by
Ashim Garg
B. E., Roorkee University, 1990
Sc. M., Brown University, 1992

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1996

Copyright
by
Ashim Garg
1996

Abstract

Graph Drawing (also known as *Graph Visualization*) tackles the problem of representing graphs on a visual medium such as computer screen, printer etc. Many applications such as software engineering, data base design, project planning, VLSI design, multimedia etc., have data structures that can be represented as graphs. With the ever increasing complexity of these and new applications, and availability of hardware supporting visualization, the area of graph drawing is increasingly getting more attention from both practitioners and researchers.

In a typical drawing of a graph, the vertices are represented as symbols such as circles, dots or boxes, etc., and the edges are drawn as continuous curves joining their end points. Often, the edges are simply drawn as (straight- or poly-) lines joining their end points (and hence the title of this thesis), followed by an optional transformation into smooth curves. The goal of research in graph drawing is to develop techniques for constructing good drawings of the input graphs. Informally speaking, a drawing is considered "good" if it is able to communicate effectively and clearly, the information being displayed in the drawing. Several desirable properties of a good drawing, that have been identified are: small area, minimum edge crossings, large angles between edges, symmetry, small number of edge-bends, upwardness, clustering or aligning of related vertices etc. In this thesis, we have addressed the problem of constructing drawings that exhibit some of these properties, and have presented several new techniques that provide answers to these problems. The problems we have considered relate to the following drawings:

- Planar upward drawings of directed acyclic graphs, i.e., planar drawings in which edges flow in the same direction (usually along y-axis).
- Planar straight-line drawings of planar graphs, in which the angles between edges incident on the same vertex have large magnitudes.
- Drawings of angle graphs, i.e., graphs with angles specified between edges incident on the same vertex.
- Planar orthogonal drawings (edges drawn as a chain of horizontal and vertical line-segments) of degree-four graphs, with minimum number of edge-bends.

Acknowledgements

This thesis is the most important milestone of the first twenty five years of my life. In a sense, its conception was the goal of all my endeavors during this period. Over these years, a number of people have affected me both directly and indirectly, and both knowingly and unknowingly, and therefore have contributed to this thesis. I am deeply indebted to all of them. Unfortunately, because of the lack of space, I am able to mention only some of them here.

I am extremely grateful to my mother *InduBala Jain* and my father *Rajendra Mohan Garg*, whose contributions to my life can never be expressed in words. It is rather unfortunate that my mother could not see the completion of this thesis, but I am sure that she would have been even happier than me, on its completion. I am very thankful to my brother *Amit Garg*, without whose compassion and support, I wouldn't have been able to finish my thesis-work. He stood steadfastly by me during my difficult times, without asking anything in return. I also thank my sister *Parul Garg* for her boundless affection towards me, and for enriching my life with her cheerful and spirited approach to living. Ever since my early childhood, I have always looked upon and admired my uncles *Rakesh Kumar*, *Ajai Kumar Jain* and *Alok Kumar Jain* for a variety of reasons. They have always been my sources of inspiration, and this thesis owes as much to them as to my efforts.

I entered Brown with very little research experience. My advisor *Roberto Tamassia* not only taught me how to do research, but more importantly, taught me what it means to be a researcher. I am extremely grateful to him. I am also deeply indebted to my co-researcher *Isabel Cruz* for her constant support and encouragement during my thesis-work. I also thank my office-mate *Lloyd Greenwald* for his companionship during my stay at Brown, and for always answering very patiently all sorts of inane questions that I pestered him with.

I would like to thank *Steve P. Reiss* and *Ioannis G. Tollis* for being on my thesis-committee, despite their busy schedule.

Finally, I would like to thank the proponents of *existentialism*, especially *Jean Paul Sartre*, whose ideas helped me understand life better, and improved me as a person.

Contents

Abstract iii

Acknowledgements iv

List of Figures ix

1 Introduction 1

2 Upward Planar Drawings of Directed Graphs 7
- 2.1 Introduction . . . 7
- 2.2 Preliminaries . . . 9
  - 2.2.1 Upward Embedding . . . 9
  - 2.2.2 Tendrils and Wiggles . . . 10
- 2.3 An Auxiliary Undirected Flow Problem . . . 12
- 2.4 Upward Planarity Testing . . . 18
- 2.5 Conclusions . . . 24

3 Upward Planar Grid Drawings of Trees 25
  - 3.0.1 Previous Work . . . 25
  - 3.0.2 Our Results . . . 26
- 3.1 Preliminaries . . . 27
- 3.2 Polyline Drawings . . . 29
  - 3.2.1 Upward Layerings . . . 29
  - 3.2.2 Drawing Algorithm . . . 31
  - 3.2.3 Order Preserving Drawings . . . 34
- 3.3 Orthogonal Drawings . . . 38
  - 3.3.1 Drawing Algorithm . . . 38
  - 3.3.2 A Superlinear Lower Bound on the Area . . . 41
- 3.4 Experimental Results . . . 43
- 3.5 Conclusion . . . 44

4 Angular Resolution 47
- 4.1 Introduction . . . 47
- 4.2 Definitions . . . 49
- 4.3 Upper Bound on the Angular Resolution . . . 50
- 4.4 Tradeoff between Area and Angular Resolution . . . 53

4.5 Constructing Drawings with Large Angular Resolution 57
4.5.1 A Geometric Lemma 57
4.5.2 Nested-Star Graphs 59
4.5.3 Series-Parallel Graphs 64
4.5.4 Outerplanar Graphs 72
4.6 Conclusion 79

5 Angle Graphs **81**
5.1 Introduction 81
5.1.1 Some Definitions 82
5.1.2 Previous Work 83
5.1.3 Our Results 84
5.2 Planarity Testing of Angle Graphs 85
5.2.1 NP-Hardness of Planarity Testing 88
5.3 Testing Triconnected Planar Graphs for High Angular Resolution 100
5.4 Testing a Series-Parallel Angle Graph for Consistency 103
5.5 Area Requirement of Angle Graphs 113
5.6 Multilayered Angle Graphs 114
5.6.1 Some Gadgets 115
5.6.2 Completed Course 116
5.6.3 Angle Graph A(T) 117
5.7 Discussion 124

6 Rectilinear Planarity Testing **125**
6.1 Introduction 125
6.2 Preliminaries 126
6.2.1 Rectilinear Embedding 127
6.2.2 Tendrils and Wiggles 127
6.3 Rectilinear Planarity Testing 128
6.4 Conclusions 133

7 Bend-Minimization **134**
7.1 Introduction 134
7.2 Preliminaries 136
7.3 Network Flows 137
7.4 Bend-Minimization and Network Flow 143
7.5 Algorithm compute-flow 147
7.6 Conclusion 150

8 Conclusions and Open Problems **151**

Bibliography **161**

List of Tables

3.1 Area-requirements for various types of planar grid drawings of a rooted tree with N nodes. 28
3.2 Experimental results on the drawings of complete binary trees and Fibonacci trees produced by the implementation of the algorithm of Corollary 2 with $\alpha = 1/2$, and comparison with the theoretical upper bounds. . 45

5.1 The color of the j^{th} reference, value, inner and separation edges of Subcourse S_i. 118

List of Figures

1.1 Different drawing standards: (a) Straight-line drawing; (b) Polyline drawing; (c) Orthogonal drawing 2

2.1 Example of a planar straight-line upward drawing of a directed graph. . 7
2.2 Examples of: **(a)** Tendril T_3; and **(b)** Wiggle W_3. 11
2.3 **(a)** Tendril T_1. **(b)** Graph H_k. **(c)** Constructing T_k from T_{k-1}. 11
2.4 **(a)** Switch-flow network $\mathcal{N}$ with parameter $\theta = 4$ associated with the the NOT-ALL-EQUAL-3-SAT instance $\mathcal{S}$ with clauses $c_1 = y_1x_2y_3$, $c_2 = y_1y_2x_3$, and $c_3 = x_1x_2x_3$. The clause edges with nonzero capacity range are shown with thick lines. **(b)** Feasible flow for $\mathcal{N}$ corresponding to the satisfying truth assignment (y_1, x_2, x_3) for $\mathcal{S}$. Only the edges with nonzero flow are shown. **(c)** Planar switch-flow network $\mathcal{P}$ associated with $\mathcal{S}$. **(d)** Feasible flow for $\mathcal{P}$ corresponding to the satisfying truth assignment (y_1, x_2, x_3) for $\mathcal{S}$. Only the edges with nonzero flow are shown. 14
2.5 Schematic illustration of the edges incident on the literal and clause vertices of network N: **(a)** literal vertices x_i and y_i; **(b)** clause vertex c_j. . . 15
2.6 Proof of Lemma 5 : **(a)** drawing $\psi(2, m)$; **(b)** drawing $\psi(k, m)$. The small circles show the lowest crossings of the clause edges in the drawing of $\psi(k-1, m)$. 17
2.7 Orientation $\vec{\mathcal{P}}$ of the network $\mathcal{P}$ shown in Fig. 2.4 (c). 19
2.8 Dual digraph $\vec{\mathcal{D}}$ of network $\vec{\mathcal{P}}$ shown in Fig. 2.7 20
2.9 **(a)** Schematic illustration of digraph $\vec{\mathcal{G}}$ obtained from $\vec{\mathcal{D}}$ by replacing edges with tendrils and wiggles. **(b)** The two faces of $\vec{\mathcal{G}}$ associated with literal vertices x_i and y_i of $\mathcal{P}$. **(c)** The face of $\vec{\mathcal{G}}$ associated with a clause vertex of $\mathcal{P}$. **(d)** The face of $\vec{\mathcal{G}}$ associated with a crossing vertex of $\mathcal{P}$. . . 22

3.1 Example of planar upward drawings of the same binary tree: **(a)** straight-line; **(b)** polyline; **(c)** orthogonal. 29
3.2 **(a)** Example of upward layering λ of a left-heavy tree. **(b)** Drawing associated with λ. 30
3.3 Illustration of the execution of round 4 of the upward layering algorithm of Theorem 4 . Here, $N = 50$, $\alpha = 1/2$ and $k = 4$. **(a)** Tree. **(b)** Two blocks (of size at most 4) are created in Step 2c , and nodes x and v are marked in Step 2d 32
3.4 Drawing of the complete binary tree with 63 nodes produced by the implementation of the algorithm of Corollary 2 with $\alpha = 1/2$. 35

3.5 Order-preserving drawings: **(a)** Tree for the lower bound of Theorem 6 . **(b)** Each pair of consecutive edges of the chain increases the height by at least one unit. 36
3.6 Example of construction of order-preserving drawings. Here, $m = 5$. **(a)** Tree T with subtrees $T_1, T_2, \ldots, T_5$. The number inside the triangle representing subtree T_i indicates the number of nodes of T_i. **(b)** Drawing of T, where the rectangles represent the drawings of the T_i's. **(c)** An ordered tree T'. **(d)** Drawing of T' constructed by the algorithm of Theorem 7 . . 37
3.7 Illustration of the algorithm of Theorem 8 : **(a)** Example of a drawing produced by the algorithm of Lemma 12 . **(b)** A binary tree T and the separators that join blocks, drawn with thick lines. Here, $N = 55$, $\log_2 N = 5$, one block has size 6, and the remaining blocks have size 7. The block size B is obtained for T by using the constraint $\log_2 N \leq B \leq 2\log_2 N$. **(c)** Truncated separator tree S' of the tree T of part (b). The nodes of S' correspond to the separators of T. **(d)** Stacking of the drawings of the blocks of T and routing of the separators. The rectangles represent the drawings of the blocks. **(e)** Drawing of T constructed by the algorithm of Theorem 8 40
3.8 Tree for the lower bound of Theorem 9 42
3.9 Drawing of subchain S in the proof of Theorem 8 43
3.10 Experimental results on the area of the drawings of complete binary trees produced by the algorithm of Corollary 2 with $\alpha = 1/2$, and comparison with the theoretical upper bounds 44
3.11 Experimental results on the area of the drawings of Fibonacci trees produced by the algorithm of Corollary 2 with $\alpha = 1/2$, and comparison with the theoretical upper bounds 46

4.1 Dividing a triangle ΔABC by introducing a vertex D in its interior and joining D with A, B and C. 51
4.2 (a) Graph G_0; (b) Graph G_k; 52
4.3 Graph G_n^d: (a) G_0; (b) Constructing G_k from G_{k-1}. Notice that G_n^d is defined as $G_{\frac{n-3}{d-4}}$, whenever $n-3$ is a multiple of $d-4$. 54
4.4 Let β_i be the internal angle $\angle A_k B_i C_i$ of the triangle $\Delta A_k B_i C_i$. (a) Case 1: $\beta_i < 90°$; (b) Case 2: $\beta_i \geq 90°$ and $d_i > \sqrt{\Delta_i}$; (c) Case 2: $\beta_i \geq 90°$ and $a_i > \sqrt{\Delta_i}$ 55
4.5 Triangle ΔABC in Lemma 15 57
4.6 A nested-star graph G: (a) G consists of a single triangle; (b) G is a composition of three nested-star graphs G_1, G_2 and G_3; 60
4.7 Proof of Lemma 16 for a nested-star graph G: (a) G consists of a single triangle; (b) G is a composition of three nested-star graphs G_1, G_2 and G_3; 60
4.8 Constructing Drawing $D(G, \Delta)$, when the nested-star graph G consist of a: (c) A single triangle; (d) nested-star graphs $U_1, U_2, \ldots U_m$ and $V_1, V_2, \ldots V_m$ 61
4.9 (a) A nested-star graph G with degree 7; (b) A drawing of G, with angular resolution $60°/(7-1) = 10°$, constructed by the algorithm of Section 4.5.2 . 64

4.10 A series-parallel graph G when it is: (a) a single edge (s,t), (b) a series-composition of series-parallel graphs G_1 and G_2, and (c) a parallel-composition of series-parallel graphs G_1 and G_2. (d) Corresponding Q node when G is a single edge; (e) Corresponding S node when G is a series-composition of G_1 and G_2; (f) Corresponding P node when G is a parallel-composition of G_1 and G_2. 65
4.11 Example of a SPQ-tree $SPQ(G)$ and its equivalent compact SPQ-tree $T(G)$: (a) A series-parallel graph G; (b) SPQ-tree $SPQ(G)$; (c) Compacting vertex u into vertex v; (d) The compact SPQ-tree $T(G)$. 66
4.12 Drawing a series-parallel graph G: (a) Drawing G when it is a single edge; (b) Drawing G when it is a series composition of series-parallel graphs G_1 and G_2; 67
4.13 Drawing a series-parallel graph G when it is a parallel composition of series-parallel graphs $G_1, G_2, \ldots, G_m$: (a) Case **i**; (b) Case **ii**. 68
4.14 Case **i**. Series-parallel graph G is a parallel composition of series-parallel graphs $G_1, G_2, \ldots, G_m$, where each G_i consists of at least two edges: (a) Composition of G, (b) Compact SPQ-tree $T(G)$ associated with G. 69
4.15 Case **i**. Series-parallel graph G is a parallel composition of series-parallel graphs $G_1, G_2, \ldots, G_m$, where G_1 consists of only one edge: (a) Composition of G, (b) Compact SPQ-tree $T(G)$ associated with G. 69
4.16 A drawing, with angular resolution $2 \cdot 60°/(2 \cdot 6 - 1) = 11°$, of the series-parallel graph G of Fig. 4.11 (a), constructed by the algorithm of Section 4.5.3 . Notice that G has degree 6. 72
4.17 Proof of Property **P1**: (a) If there is an edge (u,v); (b) If u and v have a common immediate neighbor t 74
4.18 Proof of Property **P2**: (a) w_1 is in the interior of the cycle $r \rightsquigarrow uw_2v \rightsquigarrow r$ (b) w_2 is in the interior of the cycle $r \rightsquigarrow uw_1v \rightsquigarrow r$ 75
4.19 Proof of (a) Property **P3**; (b) Property **P4** 75
4.20 Proof of Property **P5**: (a) Vertices of path $p : a_1a_2 \ldots a_m$ are immediate successors of q vertices $w_1w_2 \ldots w_q$, where $q \geq 3$; (b) p does not contain the common immediate successor of u and v. 76
4.21 Drawing an outerplanar graph G: (a) Placing on Circle C_1, the vertices belonging to layer 1; (b) Placing on Circle C_i, the vertices of a maximal path $p : v_1v_2 \ldots v_q$ belonging to layer i 77
4.22 (a) A maximal outerplanar graph G with degree 5; (b) A drawing of G, with angular resolution $180°/(5-1) = 45°$, constructed by the algorithm of Theorem 14 79

5.1 An angle graph (a) that is not consistent; (b) that is consistent. 83
5.2 (a) Interlacing biconnected components B_1 and B_2, (b) B_1 directly forced inside B_2, and B_2 directly forced inside B_1, (c) B_1 is forced inside B_3. These figures are taken from [110] 86
5.3 Proof of Theorem 15 : (a) Angle graph A with biconnected components B_1 and B_2, and articulation vertex u. The edges of B_1 are shown as thick lines, and the edges of B_2 are shown as thin lines. (b) A drawing D of A. The dotted lines are not part of D, and are used for the proof. 87

5.4 (a) A horse shoe H with size $r_1 \times r_2$; (b) A drawing of H when it is left heavy; (c) A drawing of H when it is right heavy. 89
5.5 A Crown 90
5.6 A Beam 90
5.7 A Wiggle 91
5.8 A Connector 91
5.9 A Holder 92
5.10 (a) A high level view of Angle Graph A(T); (b) Constructing A(T). . . . 92
5.11 Constructing Variable Subgraph V_i from Variable Subgraph V_{i-1} 93
5.12 Constructing Clause Subgraph S_j from Clause Subgraph S_{j-1} 94
5.13 Bipartite graph H: (a) A drawing D of H; (b) Drawing D'; (c) Planar Drawing D''. 96
5.14 A planar drawing D of the associated angle graph A(T) of an instance T of the 3-SAT problem, where T consists of the variables x_1, x_2 and x_3, and the clauses $(x_1, \bar{x_2}, x_3)$, $(x_1, x_2, \bar{x_3})$ and $(\bar{x_1}, x_2, \bar{x_3})$. D corresponds to the truth-assignment x_1 =`true`, x_2 =`false` and x_3 =`false`. This figure is not to scale. Also, to reduce the visual complexity, we have not shown all the vertices and edges of A(T), and have filled some regions of the drawing with shading. In the planar drawing of A(T) described in the *only if* part of the proof of Lemma 19 , C_1, C_2 and C_3 are drawn at x-coordinates that are to the left of the x-coordinates at which X_1 is drawn. 97
5.15 Constructing Graph G from A(T): (a) Fan F_6; (b) Attaching fans $\mathcal{F}_9$, $\mathcal{F}_{18}$ and $\mathcal{F}_{45}$ to "fix" the angle between the edges (shown as thick lines) incident on v to 45°, 90°, and 225° respectively; (c) Connecting fans introduced in the same face- The connectors are shown as thick lines. 101
5.16 A series-parallel graph G: (a) Single edge (s, t); (b) Series composition of G_1 and G_2; (c) Parallel composition of G_1 and G_2. 103
5.17 Illustration of the concepts of: (a) a sector, (b) *lies inside* and *reverse included*, and (c) *opposite* sectors. 105
5.18 (a) A series-parallel angle graph G; (b) $Sector(G)$, assuming that su is aligned with the reference axis. 106
5.19 $sector(A)$, shown as darkly shaded region, when the series-parallel angle graph A is a series-composition of series-parallel angle graphs A_1 and A_2 ($sector(A_1)$ and $sector(A_2)$ are shown as lightly shaded regions): (a) $sector(A_1)$ and $sector(A_2)$ are opposite; (b) $sector(A_1)$ and $sector(A_2)$ are not opposite, and $\alpha_2 - \beta_1 \leq 180$; ($c$) $sector(A_1)$ and $sector(A_2)$ are not opposite, and $\alpha_2 - \beta_1 > 180$; 107
5.20 $sector(A)$, shown as darkly shaded region, when the series-parallel angle graph A is a series-composition of series-parallel angle graphs A_1 and A_2 ($sector(A_1)$ and $sector(A_2)$ are shown as lightly shaded regions): (a) $sector(A_1)$ is a disk; (b) $\beta_1 < \alpha_2$ and $\beta_2 < \alpha_1 + 360$; (c) $\beta_1 < \alpha_2$ and $\beta_2 \geq \alpha_1 + 360$; (d) $\beta_1 \geq \alpha_2$. 108

5.21 Proof of Theorem 19 : Series-parallel graph A is a series-composition of series-parallel graphs A_1 and A_2. (a) *Case 1: sector*(A_1) and *sector*(A_2) are opposite; (b) Constructing a drawing of A for Case 1. (c) *Case 1: sector*(A_1) and *sector*(A_2) are not opposite; (d) Constructing a drawing of A for Case 2. 111
5.22 Angle graph A_m: (a) A_1; (b) Constructing angle graph A_m from A_{m-1}. . 114
5.23 (a) A subcourse S; (b) A right heavy subcourse; (c) A left heavy subcourse. In the figure, Except for the reference, value, inner and separation edges, an edge is shown as a thick line if it is a red edge, and is shown as a thin line if it is a blue edge. 116
5.24 (a) A Starting Location; (b) An Ending Location; (c) A Hurdle; (d) A Runner. Blue edges are shown as thin lines and red edges are shown as thick lines. 117
5.25 A high level view of (a) Completed Course G_n; (b) Angle graph A(T). . . 118
5.26 A biplanar drawing D of the associated angle graph A(T) of an instance T of the 3SAT problem, where T consists of the variables x_1, x_2 and x_3, and the clauses $(x_1, \bar{x_2}, x_3)$, $(x_1, x_2, \bar{x_3})$ and $(\bar{x_1}, x_2, \bar{x_3})$. D corresponds to the truth-assignment x_1 =`true`, x_2 =`false` and x_3 =`false`. The red edges are shown as thick lines, and the blue edges are shown as thin lines. 119
5.27 Runner R_j spends two edges to span Hurdle H in any biplanar drawing. e is the j^{th} output edge of S_n. Notice that the outgoing edge of R_j in S_n is blue in any biplanar drawing. The red edges are shown as thick lines and the blue edges are shown as thin lines. 121
5.28 Subcourse S_i spanned by a Runner R_j in a biplanar Drawing. g is the j^{th} output edge of S_{i-1}. b, c, d, e and f are the j^{th} input, reference, value, inner and outer edges respectively of S_i. The red edges are shown as thick lines and the blue edges are shown as thin lines. (a) S_i is neutral to R_j: R_j spends 0 edges; (b) S_i is compatible with R_j: R_j spends 2 edges. An example is when variable x_i occurs only negated in c_j and S_i is right heavy; (c) S_i is not compatible with R_j: R_j spends 4 edges. An example is when variable x_i occurs only negated in c_j and S_i is left heavy. 122

6.1 Example of a planar rectilinear drawing of a graph. 125
6.2 Example of rectilinear tendril T_3. 128
6.3 **(a)** Rectilinear Tendril T_1. **(b)** Graph H_k. **(c)** Constructing T_k from T_{k-1}. 129
6.4 The four different rectilinear embeddings of rectilinear tendril T_1. 129
6.5 Example of a rectilinear wiggle W_3. 129
6.6 Schematic illustration of graph $\mathcal{G}$ obtained from $\mathcal{F}$ by replacing edges with rectilinear tendrils and wiggles: **(a)** the two faces of $\mathcal{G}$ associated with literal vertices x_i and y_i of $\mathcal{P}$. **(b)** the face of $\mathcal{G}$ associated with a clause vertex of $\mathcal{P}$. **(c)** the face of $\mathcal{G}$ associated with a crossing vertex of $\mathcal{P}$. 130

7.1 (a) An embedded degree 4 planar graph G. (b) An orthogonal representation of G. 137

7.2 (a) A flow network $N(V, E, c, u)$ with source s and sink t. (b) The shallowest layered network $L(N, \mathbf{0})$ of N with respect to zero flow; L is also the admissible flow network of N with respect to zero flow because each arc of N has cost 1. (c) A Blocking flow f of L. (d) Residual network $R(N, f)$ of N with respect to f. In parts (a), (b), and (d) we have labeled each arc e by the pair $(c(e), u(e))$. In part (d) we have labeled each arc by the amount of flow in it, and have shown only the arcs with non-zero flow. 138
7.3 Algorithm *SAP*. 140
7.4 Algorithm *Blockflow*. 141
7.5 Algorithm *PD*. 142
7.6 Bend-minimization and network flow: (a) An embedded degree 4 planar graph G; (b) Node-face arcs of the associated flow-network N(G) of G; (c) Face-face arcs of N(G); (d) Source-node and face-sink arcs of N(G); (e) Flow-network N(G); (f) A min-cost max-flow in N(G) and the corresponding bend-minimum planar orthogonal representation of G(shown as broken lines). Notice that each arc of N(G) has cost 1. In parts (b)-(f), we have shown the arcs of N(G) as solid lines, and the edges of G as broken lines. In parts (b)-(d), we have have labeled each arc by its capacity. In part (f), we have labeled each arc by the amount of flow in it, and have shown only the arcs with non-zero flow. 144
7.7 Algorithm *min-bend* 145
7.8 Algorithm *Orthogonalize* 145
7.9 Algorithm *compute-flow*. 147

Chapter 1

Introduction

It is well-known that human beings are very good in processing visual information, as is succinctly and effectively captured in the adage "a picture is worth a thousand words". Hence, in applications that involve human interaction, visual display of information can be very useful. Examples of applications include software engineering, data bases, VLSI circuit-design, project planning, scientific data analysis, and graphics design. In fact, with the ever growing capability of the visual-display technology, the list of such applications is growing every day, and so is the complexity of visual information that is needed to be displayed.

A *graph* is a structure consisting of a set of vertices and a set of edges connecting these vertices. Many applications consist of structures that can be modeled very naturally by graphs. Some examples of graphs that occur in real-life situations are subroutine call graphs, inheritance hierarchies and class inter-relationship diagrams in object-oriented databases and object-oriented programs, VLSI circuits, Pert networks, and covering digraphs of partial orders.

Graph Drawing (also known as *Graph Visualization*) addresses the problem of constructing good drawings (synonyms: diagrams, pictures, layouts) of graphs. Informally speaking, a drawing is considered "good" if it is able to convey to the user clearly and completely, the information that is being or is intended to be provided in the form of the drawing. A good graph drawing utility is one that given as input, a graph containing the information intended to be conveyed to user, produces a good drawing of the graph. An extensive survey on the area of graph drawing is presented by Di Battista, Eades, Tamassia and Tollis in [30].

While it is difficult, perhaps impossible, to measure exactly, the "goodness" of a drawing, it has been found, see e.g. [3, 28], that two criteria are considered important by the human designers:

- *Aesthetic criteria* which are independent of the information being displayed, and are concerned with the "beauty" of the drawing. Some aesthetic criteria that have been identified as important are small number of crossings, symmetry, upwardness (for directed acyclic graphs), large angles between edges incident on the same vertex (in straight-line drawings), small number of bends (in polyline drawings),

small total edge length, small area, uniform distribution of vertices etc.

- *Semantic constraints*, on the other hand, are certain layout rules that aim to provide semantic information on the basis of layout. Typical layout rules are, clustering or aligning "related" vertices, placing the most important vertex in the center of the drawing etc.

To classify the different kind of graph drawings, various *graphic standards* have been proposed in literature [30]. Vertices of the graph are generally represented as symbols such as circles, dots, or boxes, etc., and edges are drawn as continuous curves joining their end points. Often, the edges are simply drawn as (straight- or poly-) lines joining their end points (and hence the title of this thesis), followed by an optional transformation into smooth curves. A *straight-line* drawing is one in which edges are drawn as straight-line segments (Fig. 1.1(a)). In a *polyline* drawing, each edge is drawn as a polygonal chain of straight-line segments (Fig. 1.1(b)). A vertex of a polygonal chain, that is not an end point of the chain, is called a *bend* of the drawing. A *k-gonal* drawing is a polyline drawing in which each line segment makes an angle that is an integer multiple of $2\pi/k$ with the x-axis. An *orthogonal* drawing is a *4-gonal* drawing, i.e., a polyline drawing in which each edge is drawn as a chain of horizontal and vertical straight-line segments (Fig. 1.1(c)). A *planar* drawing is a drawing in which no two edges cross except at their common end point. A *grid* drawing is a polyline drawing in which each vertex and each bend is assigned integer coordinates.

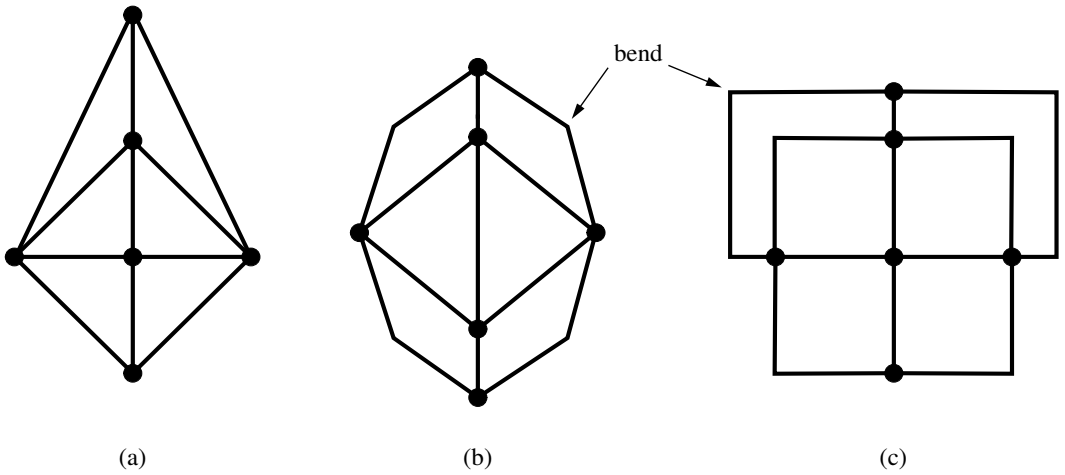

Figure 1.1: Different drawing standards: (a) Straight-line drawing; (b) Polyline drawing; (c) Orthogonal drawing

The current graph drawing techniques are, broadly speaking, based on two type of approaches: *the algorithmic approach* and *the declarative approach.*

The *declarative approach* (see for example, [28, 25, 40, 66]. Also see [30] for a detailed bibliography) allows users to specify constraints such as clustering or aligning a set of vertices etc., so that the drawings constructed satisfy these constraints. The declarative approach provides a lot of flexibility in type of drawings that can be constructed using

them. However, the techniques based on declarative approach, because of the inherent complexity of solving constraints, are generally slow, do not always provide a solution, or sometimes provide non-intuitive solutions, and it is difficult to guarantee any kind of performance bounds (for running-time etc.) for them. A current "hot" topic for research is to devise declarative approach based techniques with improved efficiency, more amenability to performance-analysis etc. (see for example, [23, 56]).

The *algorithmic approach* takes a more pragmatic path. This approach emphasizes on developing algorithms for constructing drawings that satisfy only a fixed set of aesthetic criteria. These algorithms first formalize the aesthetic criteria under consideration as optimization goals, and then construct a drawing optimizing these goals. Because each algorithm has to typically optimize only a fixed set of goals, these algorithms can be quite efficient and can also be analyzed to give performance guarantees. Also, the algorithms are usually decomposable into smaller units called *methods*, so that one can devise new algorithms using the methods of the existing algorithms [8, 34]. Because of these advantages, research in graph drawing has mainly concentrated on devising algorithms based on the algorithmic approach. This thesis also focuses on the algorithmic approach.

This thesis mainly addresses the problem of constructing four different kinds of drawings: upward planar drawings, planar straight-line drawings with large angular resolution, angle-preserving drawings, and orthogonal planar drawings with minimum number of bends. It also contains some results that hold individually for other kinds of drawings.

In Chapter 2, we consider the problem of constructing upward planar drawings of directed graphs. An *upward drawing* is one in which all the edges flow in one direction (usually along the positive y-axis). Fig. 2.1 shows an upward planar drawing. Because of the "unidirectionality" of upward drawings, the user doesn't have to go back and forth in y-direction while traversing a directed path. Hence, it is easier to navigate through such drawings. Upward drawings are also very suitable for displaying hierarchical structures. Upward drawings find application in displaying structures such as object-oriented models, subroutine call graphs, organization charts, directory-hierarchy, pert networks, partial orders, etc. The *upward planarity testing* problem, i.e., the problem of testing whether a given directed graph (*digraph*, in short) admits an upward planar drawing, has intrigued researchers for many years. Polynomial-time algorithms are known for upward planarity testing of triconnected digraphs [7, 9], bipartite digraphs [35], st-digraphs [36, 73], single-source multiple-sink digraphs [64, 10], and outerplanar digraphs [86]. However, despite the effort of many researchers, it was not previously known whether one can test a general digraph for upward planarity in polynomial time. Using the concept of a special kind of flow network, we show that the upward planarity testing problem for general digraphs is NP-complete and hence a polynomial-time algorithm is unlikely to exist.

The most natural way of drawing trees is to draw them upward, as an illustration of a tree in any text book will demonstrate. The issue of area is important because of the finite resolution of the displaying and printing devices. Trees are very popular data-structures, making appearance in a number of applications. It is therefore no surprise that the problem of drawing trees has been studied quite extensively, and a number of results are available. See for example, [112, 89, 42, 98, 44, 84, 15, 109]. Also see [30] for a detailed bibliography. As for the area-requirement of upward planar grid drawings

of trees, the well-known algorithms of Reingold and Tilford [89], and Wetherell and Shannon [112] give drawings with area $O(n^2)$, where n is the number of nodes in the tree. Crescenzi, Di Battista, and Piperno [21, 22] have given algorithms for constructing a straight-line upward planar grid drawing with area $O(n \log n)$ of a given general tree, and with area $O(n)$ of a given complete binary or Fibonacci or AVL tree. In Chapter 3, we study the area-requirements of upward planar grid drawings of trees, within two graphic standards, namely polyline and orthogonal, and provide matching upper and lower bounds for the area-requirements of both kinds of drawings. More specifically, we give linear-time algorithms for constructing upward planar polyline grid drawings with area $O(n)$, upward planar orthogonal grid drawings with area $O(n \log \log n)$, and upward planar polyline grid drawings with area $O(n \log n)$ that preserve the left-to-right order of the children of each node. We also prove that these bounds are tight within a constant factor.

The *angular resolution* of a straight-line drawing is equal to the magnitude of the smallest angle between any two edges incident on a vertex. For example, the angular resolution of the drawing of Fig. 1.1(a) is 20°. It is important for a drawing to have large angular resolution because it is difficult for the eye to distinguish between two lines separated by only a small angle. A large angular resolution is also important in wireless communication where it is difficult to receive and send signals at a tight angle [48]. *Degree* of a graph is equal to the maximum number of edges incident on a vertex of the graph. Formann, Hagerup, Haralambides, Kaufmann, Leighton, Simvonis, Welzl, and Woeginger have shown in [48] that every degree-d planar graph admits a straight-line (*nonplanar*) drawing with angular resolution $\Omega(1/d)$. Malitz and Papakostas [83] show that a degree-d planar graph admits a planar straight line drawing with angular resolution $\Omega(1/7^d)$ (notice that this bound is independent of the number of vertices in the graph). It is however still not known whether a degree-d planar graph admits a planar straight line drawing with angular resolution that is a polynomial in d, and is independent of the number of vertices in the graph. In Chapter 4, we consider the problem of constructing planar straight-line drawings with large angular resolution. We provide the first non-trivial upper bound on angular resolution, namely, we show that there exists an infinite family of degree-d planar graphs that requires angular resolution $O(\sqrt{\log d/d^3})$ in any planar straight-line drawing. We also show a trade-off between the area and angular resolution of a drawing which implies that there exists a family of degree-d planar graphs that requires exponential area for any planar straight-line drawing with angular resolution that is independent of d and n, where n is the number of vertices in a graph. We also give linear-time algorithms for constructing drawings of outerplanar, star-nested, and series-parallel graphs with angular resolution that is a polynomial in the degree of the graph and is independent of the number of vertices in the graph.

In Chapter 5, we study the problem of constructing angle-preserving drawings, i.e. drawings of angle graphs. An *angle graph* is a graph having a fixed embedding and equipped with angles between consecutive edges incident on its vertices. An equilateral triangle, for example, is an angle graph in which the angle between any two consecutive edges incident on a vertex is either 60° or 300°. A *drawing* of an angle graph is a straight-line drawing that preserves its angles. Angle graphs capture the notion of the desired

"shape" of the drawing and hence are important for developing systems for generating layouts with user-specified constraints. In addition, angle graphs seem to be useful as intermediate-stage products for constructing drawings of graphs, as demonstrated by their implicit use in the well-known algorithms of [99] and [102] for constructing planar orthogonal drawings. We give several new results on drawings of angle graphs. Using a gadget based approach, in which we use smaller graphs to construct a bigger graph, we have shown that it is NP-hard to determine whether a given angle graph admits a planar drawing. We also study the problem of testing whether a given series-parallel angle graph admits a drawing, and provide a linear-time testing-algorithm. Using the NP-hardness result, we also show that given a *triconnected* planar graph, it is NP-hard to construct a planar straight-line drawing of the graph with maximum angular resolution, extending a similar NP-hardness result of [69, 68] that holds for *biconnected* planar graphs. This result also demonstrates the benefits of studying drawings of angle graphs. A *multilayered* angle graph is one whose edges are assigned to a set of layers available. A *multiplanar* multilayered angle graph is one that admits a drawing in which edges belonging to the same layer do not cross. We show that the problem of testing a multilayered angle graph for multiplanarity is NP-hard, even if, quite surprisingly, we restrict the number of layers to two, and the angles to be multiples of $90°$.

In Chapters 6 and 7, we study the problem of constructing orthogonal planar drawings with small number of bends. Such drawings are attractive for many reasons. They tend to give more aesthetically pleasing drawings in general [31, 62]. Minimizing bends is also important in VLSI layouts where bends act as "hot points" because of their higher current density. In view of the trade-off between the angular resolution of a drawing and its area-requirement [58, 83], these drawings offer a good compromise: they have large angles (multiples of $90°$), small area, and few bends (every degree 4 graph admits am orthogonal drawing with $O(n^2)$ area and $O(n)$ bends [109, 75]). Finally, because each angle is a multiple of $90°$, orthogonal planar drawings have very nice combinatorial properties leading to several simple algorithms for constructing them. Because of these advantages, many commercial packages, e.g. [32, 106], are available that use orthogonal drawings for displaying graphs. However, it was not previously known whether one can test in polynomial time if a given graph admits a *rectilinear* planar drawing, i.e., an orthogonal planar drawing with no bends. This problem is known in literature as the *rectilinear planarity testing* problem. Previously, polynomial time testing-algorithms were known only for special categories of graphs, namely, embedded planar graphs [99], series-parallel graphs [33], and degree-3 graphs [33]. Borrowing from our techniques used for showing the NP-completeness of the upward planarity testing problem, in Chapter 6 we show that the rectilinear planarity testing problem for general graphs is also NP-complete, and hence a polynomial time algorithm is unlikely to exist. In fact, we also show that given a n-vertex graph, even approximating the number of bends within $O(n^{1-\epsilon})$ of the optimal, for any ϵ greater than 0, is NP-hard.

Our NP-hardness result of Chapter 6 necessitates the development of efficient heuristics for constructing planar orthogonal drawings with small number of bends. Because of the importance of these drawings, and due to the efforts of a large number of researchers, many heuristics are already available in literature. See for example, [95, 109, 111, 99, 97, 102, 79, 80, 81, 78, 45, 68, 13, 33]. A well-known heuristic

constructs an orthogonal planar drawing in two steps– *Step 1*: convert the given planar graph into an *embedded* planar graph (i.e., a planar graph with a fixed embedding), and *Step 2*: construct a planar orthogonal drawing with minimum number of bends of the embedded planar graph of Step 1. This heuristic is very simple and also seems to give good drawings in practice [31, 62]. Central to the success of this heuristic is minimizing the number of bends in Step 2. Tamassia has given a very famous algorithm in [99] that constructs a bend-minimum planar orthogonal drawing of an embedded planar graph. This algorithm uses a transformation to a minimum cost maximum flow problem on a related network such that, a minimum cost maximum flow on this network gives a bend-minimum planar orthogonal drawing of the embedded planar graph. This algorithm however has a time-complexity of $O(n^2)$, where n is the number of vertices in the graph. Experimental studies [31, 62] have also commented on its high running time in real-life situations. In Chapter 7, we present a new algorithm with time-complexity $O(n^{1.75} \log n)$ for constructing a bend-minimum planar orthogonal drawing of an n-vertex embedded planar graph. Our algorithm is based on Tamassia's algorithm, and essentially gains in time by using a faster minimum cost maximum flow computation on the related network.

Our conclusions and some open problems are given in Chapter 8.

Chapter 2

Upward Planar Drawings of Directed Graphs

2.1 Introduction

In this chapter, we study the problem of testing a directed graph (*digraph*, in short) for upward planarity i.e., testing whether it admits a planar upward drawing. A drawing is *planar* if no two edges cross. A graph (or digraph) is *planar* if it admits a planar drawing. A drawing of a digraph is *upward* if every edge is monotonically nondecreasing in the y-direction. A digraph is *upward planar* if it admits a planar upward drawing. Fig. 2.1 shows a planar straight-line upward drawing.

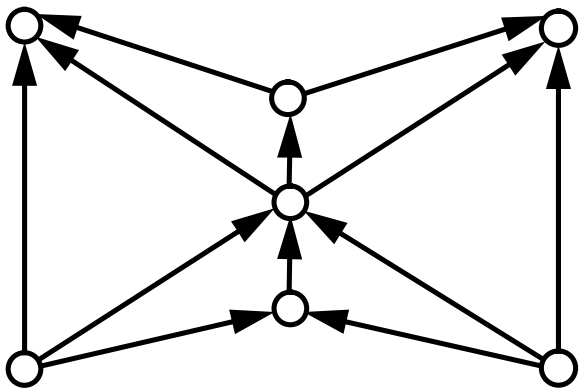

Figure 2.1: Example of a planar straight-line upward drawing of a directed graph.

Testing upward planarity is a fundamental problem in the effective visualization of various graph and network structures. For example, upward planarity is useful for the display of order diagrams and subroutine-call graphs. In this chapter we show that the following problem is NP-complete:

Upward planarity testing: testing whether a digraph is upward planar.

This problem has challenged researchers in order theory, topological graph theory, computational geometry, and graph drawing for many years. We show in Chapter 6 that the *rectilinear planarity testing* problem, i.e., testing whether a graph admits an orthogonal straight-line drawing with no bends, is also NP-complete. Our intractability results motivate the following observations:

- Testing whether a graph admits a planar drawing or an upward drawing can be done in linear time. Combining the two properties makes the problem NP-hard.
- Every planar graph admits a planar straight-line drawing. Hence, we can say that planarity is equivalent to straight-line planarity, and both properties can be verified in linear time. We can view upward and rectilinear planarity as derived from straight-line planarity by adding further constraints, which make the problem become apparently much more difficult.

Previous results on upward planarity testing are summarized below (also see [54] for an extensive survey). In the rest of this section, we denote with n the number of vertices of the graph being considered.

Combinatorial results on upward planarity of covering digraphs of lattices were first given in [74, 88]. Further results on the interplay between upward planarity and ordered sets are surveyed by Rival [90, 91, 92]. Lempel, Even, and Cederbaum [76] relate the planarity of biconnected undirected graphs to the upward planarity of st-digraphs. A combinatorial characterization of upward planar digraphs is provided in [36, 73]: namely, a digraph is upward planar if and only if it is a spanning subgraph of a planar *st-digraph*. This characterization implies that upward planarity testing is in NP.

Di Battista, Liu, and Rival [35] show that every planar bipartite digraph is upward planar. Papakostas [86] gives a polynomial-time algorithm for upward planarity testing of outerplanar digraphs. Bertolazzi, Di Battista, Liotta, and Mannino [7, 9] give a polynomial-time algorithm for testing upward planarity of triconnected digraphs and of digraphs with a fixed embedding. Concerning single-source digraphs, Thomassen [107] characterizes upward planarity in terms of forbidden circuits. Hutton and Lubiw [64] combine Thomassen's characterization with a decomposition scheme to test upward planarity of a single-source digraph in $O(n^2)$ time. Bertolazzi, Di Battista, Mannino, and Tamassia [10] show that upward planarity testing of a single-source digraph can be done optimally in $O(n)$ time. They also give a parallel algorithm that runs in $O(\log n)$ time on a CRCW PRAM with $n \log\log n / \log n$ processors.

Di Battista, Tamassia, and Tollis [36, 37] give algorithms for constructing upward planar drawings of planar st-digraphs, and investigate area bounds and symmetry display. Tamassia and Vitter [104] show that the above drawing algorithms can be efficiently parallelized. Upward planar drawings of series-parallel digraphs are studied in [5, 6].

Our proof techniques are based on a two-phase reduction from the known NP-complete problem NOT-ALL-EQUAL-3-SAT. In the first phase, we reduce NOT-ALL-EQUAL-3-SAT to an auxiliary undirected flow problem. In the second phase, we reduce this undirected flow problem to the upward planarity testing of a special class of digraphs. The

latter reduction is interesting in its own and provides new insights on the characterization by flow networks of the angles formed by the edges of upward planar drawings [7, 9]. Most of the results and techniques presented in this chapter can also be found in [57, 59].

The rest of this chapter is organized as follows. Preliminary definitions and results are provided in Section 2.2. The reduction from NOT-ALL-EQUAL-3-SAT to the auxiliary flow problem is given in Section 2.3. Section 2.4 describes the reductions from the auxiliary flow problem to upward planarity testing. Conclusive remarks are given in Section 2.5.

2.2 Preliminaries

We assume standard concepts and definitions on NP-completeness [50]. Our result uses reductions from the following well-known NP-complete problem:

NOT-ALL-EQUAL-3-SAT Given a set of clauses with three literals each, is there a truth assignment such that each clause has at least one true literal and one false literal?

An *embedding* of a planar graph is the collection of circular permutations of the edges incident upon each vertex in a planar drawing of the graph. An *embedded graph* is a planar graph equipped with an embedding. We do not distinguish between a graph and its embedding if the embedding is unique and the meaning is clear from the context.

We denote by $G - \{v\}$ the subgraph of graph G obtained by removing vertex v and its incident edges from G.

The *angles* of an embedded graph are the pairs of consecutive edges incident on the same vertex. Such angles are mapped to geometric angles in a straight-line drawing of the graph.

The following definitions are from [7, 9]. An *upward embedding* of a digraph $\vec{G}$ is an embedding of $\vec{G}$ where each angle formed by pairs of incoming or outgoing edges is assigned a label in the set $\{small, large\}$, such that there exists a planar straight-line upward drawing of $\vec{G}$ where each angle labeled *small* has measure $< \pi$ and each angle labeled *large* has measure $> \pi$. Each upward embedding has a unique external face.

A *source* of a a digraph $\vec{G}$ is a vertex with all outgoing edges and a *sink* of $\vec{G}$ is a vertex with all incoming edges. A *switch* of $\vec{G}$ is a source or sink of $\vec{G}$. A source or sink of a face f of $\vec{G}$ is called a *local* source or sink of f. Note that a local switch of f may or may not be a switch of $\vec{G}$.

2.2.1 Upward Embedding

In this section we give Lemma 1 that will be extensively used in the proofs.

First, some definitions. In an embedding of a digraph $\vec{G}$, a vertex has the *bimodal property* if the cyclic order of the edges incident on it can be partitioned into two sets of consecutive edges, one (possibly empty) consisting entirely of its incoming edges and

the other (also possibly empty) consisting entirely of its outgoing edges. An embedding of $\vec{G}$ is *bimodal* if each vertex in it has the bimodal property.

Consider an assignment of labels from the set {*small*, *large*} to the angles between incoming or outgoing edges of an embedding of $\vec{G}$. For a face f of the embedding, let $L(f)$ and $S(f)$ be the number of angles of f with label *large* and *small*, respectively. Face f is said to be *consistently assigned* if:

$$L(f) - S(f) = \begin{cases} -2 & \text{if } f \text{ is an internal face,} \\ +2 & \text{if } f \text{ is the external face.} \end{cases}$$

An assignment of labels to the angles of an embedding of digraph $\vec{G}$ is a *consistent assignment* if:

- all the angles at a nonsource or nonsink vertex of $\vec{G}$ are assigned label *small*;
- exactly one angle at a source or sink vertex of $\vec{G}$ is assigned label *large*; and
- each face is consistently assigned.

We paraphrase a result of [7, 9] in the following lemma:

Lemma 1 *An embedding of a digraph $\vec{G}$ can be extended to an upward embedding if and only if it is bimodal and admits a consistent assignment of labels to its angles.*

2.2.2 Tendrils and Wiggles

We now define several graphs that will be used as gadgets in our reduction.

We show in Fig. 2.2(a) *tendril* T_k ($k \geq 1$), which is an acyclic digraph with $k+1$ sources and $k+1$ sinks. We also define tendril T_0 as a a digraph consisting of a single edge. Tendril T_k ($k \geq 0$) has a designated source and a designated sink, called the *poles* of T_k. We shall consider transformations where a directed edge (u, v) of a digraph is replaced with a tendril T_k, where the source is identified with u and the sink with t.

Lemma 2 *Tendril T_k is upward planar and admits a unique upward embedding.*

Proof: We use induction on k. Tendril T_1 is shown in Fig. 2.3(a). Tendril T_k can be constructed by adding to T_{k-1} the digraph H_k shown in Fig. 2.3(b) and by identifying edges e_{k-1} of T_{k-1} and f_k of H_k, as shown in Fig. 2.3(c). It is straightforward to verify that T_1 and H_k have a unique upward embedding by applying Lemma 1 to their $O(1)$ planar embeddings. This proves the base of the induction. For the inductive step, since T_{k-1} and H_k each have a unique upward embedding, we only need to consider the four planar embeddings of T_k obtained by flipping the upward embeddings of T_{k-1} and H_k around their common edge f_k. Applying Lemma 1 to the four faces containing both endpoints of f_k shows that only one of the above planar embeddings of T_k can be extended to an upward embedding. □

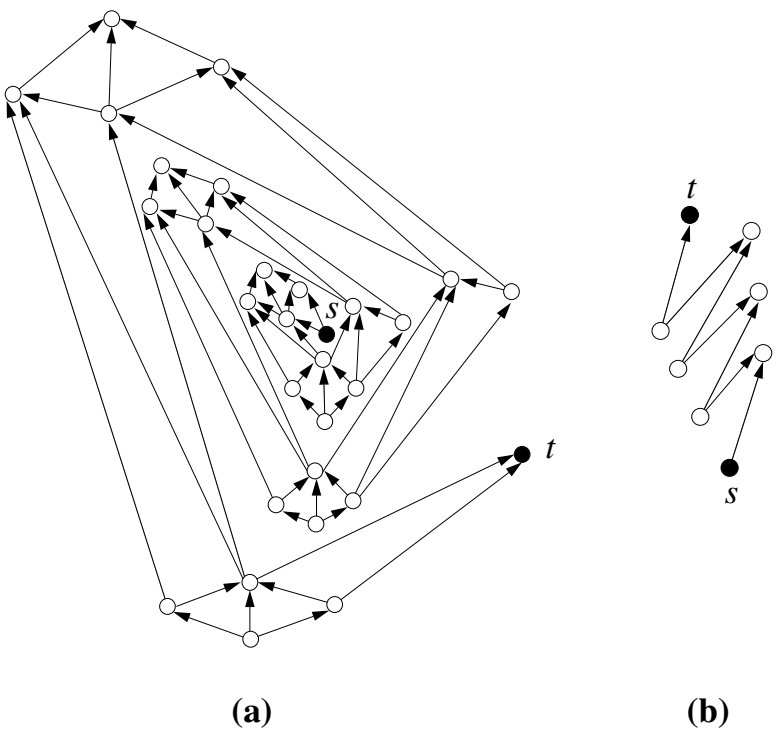

Figure 2.2: Examples of: (**a**) Tendril T_3; and (**b**) Wiggle W_3.

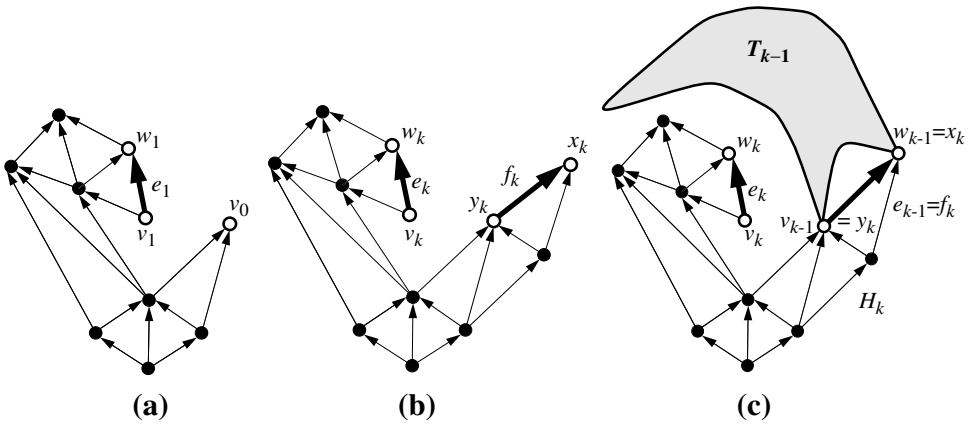

Figure 2.3: (**a**) Tendril T_1. (**b**) Graph H_k. (**c**) Constructing T_k from T_{k-1}.

In the upward planar embedding of T_k, the external face consists of two paths between s and t. One such path, called *outer path*, has $2k$ large angles and no small angles, and the other path, called *inner path*, has $2k$ small angles and no large angles. When a tendril replaces an edge of an embedded planar digraph, the outer path becomes a subpath of a face, and we say that the *contribution* of the outer path to the face is $+2k$. Similarly, we say that the contribution of the inner path to its face is $-2k$.

Figure 2.2(b) shows a *wiggle* W_k, which is an acyclic digraph consisting of a chain of $2k+1$ edges whose orientation alternates along the chain. The extreme vertices of W_k, a source s and a sink t, are called the *poles* of W_k. An *arrangement* of wiggle W_k is an upward embedding of W_k. We shall consider transformations where a directed edge (u, v) of an embedded digraph is replaced with wiggle W_k, where s is identified with u and t with v, and W_k becomes a subpath of two faces. Given an arrangement of W_k, we say that the *contribution* of W_k to a face f containing W_k is the number of large angles minus the number of small angles of W_k in f. Clearly, W_k can be arranged to give to a face any contribution c such that c is an even number between 0 and $2k$. Note that if G_k gives contribution c to a face, it gives contribution $-c$ to the other face it belongs to.

2.3 An Auxiliary Undirected Flow Problem

In this section we define two auxiliary flow problems and show that they are equivalent to NOT-ALL-EQUAL-3-SAT under polynomial-time reductions.

A *switch-flow network* is an undirected flow network $\mathcal{N}$ where each edge is labeled with a range $[c' \cdots c'']$ of nonnegative integer values, called the *capacity range* of the edge. For simplicity, we denote the capacity range $[c \cdots c]$ with $[c]$. A *flow* for a switch-flow network is an orientation of and an assignment of integer "flow" values to the edges of the network. A *feasible flow* is a flow that satisfies the following two properties:

Range property: the flow assigned to an edge is an integer within the capacity range of the edge.

Conservation property: the total flow entering a vertex from the incoming edges is equal to the total flow exiting the vertex from the outgoing edges.

Starting from an instance $\mathcal{S}$ of NOT-ALL-EQUAL-3-SAT, we construct a switch-flow network $\mathcal{N}$ as follows (see Figs. 2.4–2.5). Let the literals of $\mathcal{S}$ be denoted with $x_1, y_1, \cdots, x_n, y_n$, where $y_i = \overline{x_i}$, and the clauses of $\mathcal{S}$ be denoted with $c_1, \cdots, c_m$. Let θ be a positive integer parameter. We denote with α_i and β_i $(i = 1, \cdots, n)$ the number of occurrences of literals x_i and y_i in the clauses of $\mathcal{S}$, respectively. Note that $\sum_{i=1}^{n}(\alpha_i + \beta_i) = 3m$. Also, we define $\gamma_i = (2i-1)\theta$ and $\delta_i = 2i\theta$ $(i = 1, \cdots, n)$. Network $\mathcal{N}$ has a *literal vertex* for each literal of $\mathcal{S}$ and a *clause vertex* for each clause of $\mathcal{S}$, plus a special dummy vertex z. There are three types of edges in $\mathcal{N}$ (see Fig. 2.5):

Literal edges joining pairs of literals associated with the same boolean variable. The capacity range of literal edge (x_i, y_i) is $[\alpha_i\gamma_i + \beta_i\delta_i]$.

Clause edges joining each literal to each clause. The capacity range of clause edge (x_i, c_j) is $[\gamma_i]$ if $x_i \in c_j$, and [0] otherwise. The capacity range of clause edge (y_i, c_j) is $[\delta_i]$ if $y_i \in c_j$, and [0] otherwise.

Dummy edges joining each literal and each clause to the dummy vertex. The capacity ranges of dummy edges (z, x_i) and (z, y_i) are $[\beta_i \delta_i]$ and $[\alpha_i \gamma_i]$, respectively. The capacity range of dummy edge (z, c_j) is $[0 \cdots \eta_j - 2\theta]$, where η_j is the sum of the capacities of the clause edges incident on c_j.

The construction of network $\mathcal{N}$ from $\mathcal{S}$ is straightforward, and we have:

Lemma 3 *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses, the associated switch-flow network $\mathcal{N}$ has $O(n+m)$ vertices and $O(nm)$ edges, and can be constructed in $O(nm)$ time.*

A feasible flow in network $\mathcal{N}$ corresponds to a satisfying truth assignment for $\mathcal{S}$. Namely, we have that a literal is true whenever its incident literal edge is incoming in the feasible flow (see Fig. 2.4(b)) and its incident clause edges with nonzero capacity range are outgoing. We formalize this correspondence in the following lemma.

Lemma 4 *An instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *is satisfiable if and only if the associated switch-flow network $\mathcal{N}$ admits a feasible flow. Also, given a feasible flow for $\mathcal{N}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(nm)$, where n and m are the number of variables and clauses of $\mathcal{S}$, respectively.*

Proof:

If: Given a feasible flow in $\mathcal{N}$, we construct a truth assignment A by setting a literal true if and only if its incident literal edge is incoming in the flow. We now show that this is a satisfying assignment. Clearly, the two literals x_i and y_i associated with the same boolean variable consistently receive opposite truth assignments.

Because of the conservation property all the incident clause edges with nonzero capacity range of a true literal are outgoing, and the amount of flow in each of them is equal to the capacity. Conversely ,if a literal is false, then because of the conservation property all its incident clause edges with nonzero capacity range are incoming and the amount of flow in each of them is equal to the capacity. The three clause edges with nonzero capacity range incident on a clause vertex c_j cannot be all incoming or all outgoing because of the conservation property at vertex c_j and the choice of the capacity range for the dummy edge incident on c_j. Therefore, three literals in c_j can not be all true or all false. Hence, A is a satisfying truth assignment for $\mathcal{S}$. Also, A can be constructed in time linear in the number of edges of $\mathcal{N}$, which is $O(nm)$.

Only If: Let A be a satisfying truth assignment of $\mathcal{S}$. We construct a feasible flow f in $\mathcal{N}$ from A. In this flow,

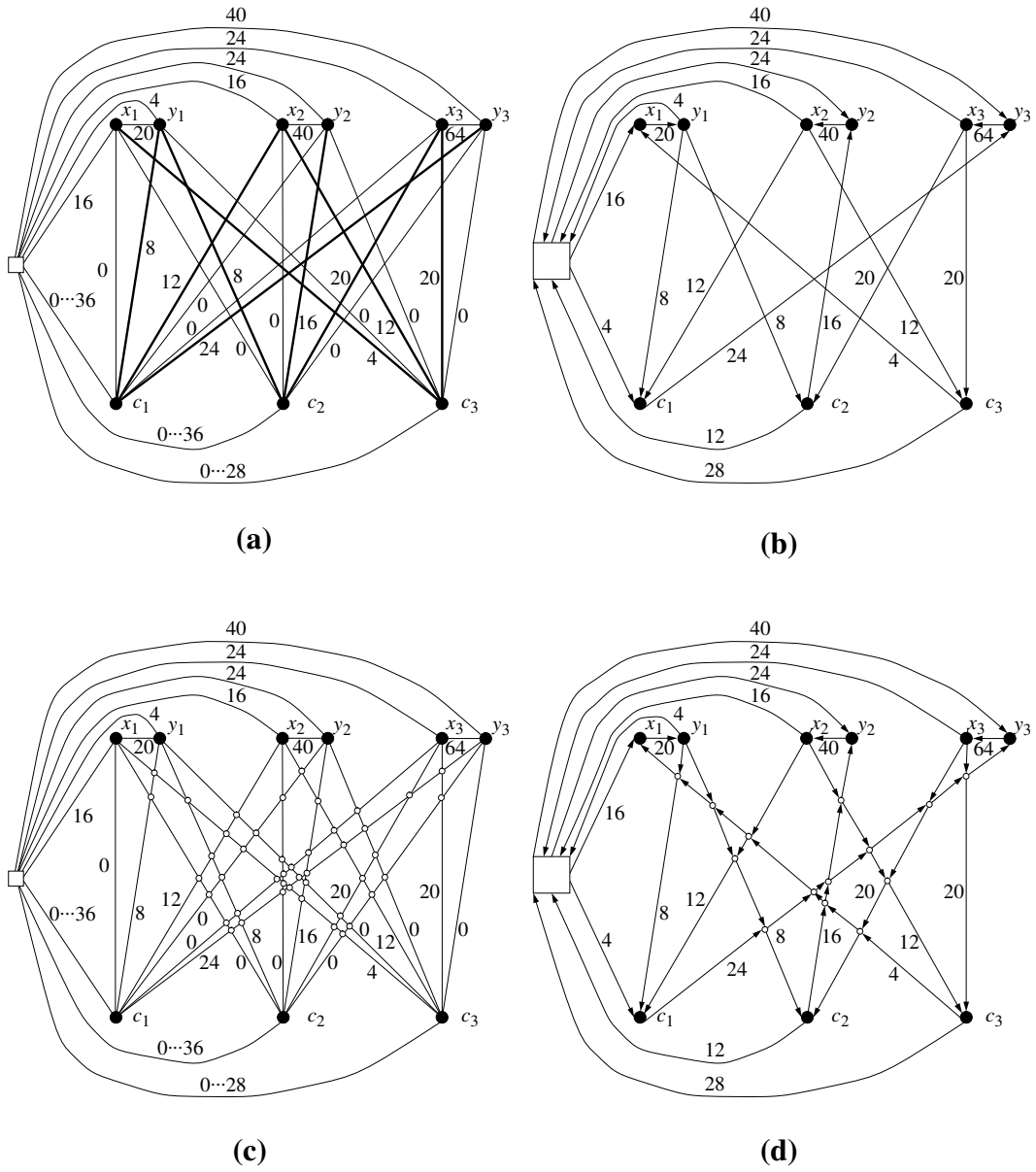

Figure 2.4: **(a)** Switch-flow network $\mathcal{N}$ with parameter $\theta = 4$ associated with the the NOT-ALL-EQUAL-3-SAT instance $\mathcal{S}$ with clauses $c_1 = y_1x_2y_3$, $c_2 = y_1y_2x_3$, and $c_3 = x_1x_2x_3$. The clause edges with nonzero capacity range are shown with thick lines. **(b)** Feasible flow for $\mathcal{N}$ corresponding to the satisfying truth assignment (y_1, x_2, x_3) for $\mathcal{S}$. Only the edges with nonzero flow are shown. **(c)** Planar switch-flow network $\mathcal{P}$ associated with $\mathcal{S}$. **(d)** Feasible flow for $\mathcal{P}$ corresponding to the satisfying truth assignment (y_1, x_2, x_3) for $\mathcal{S}$. Only the edges with nonzero flow are shown.

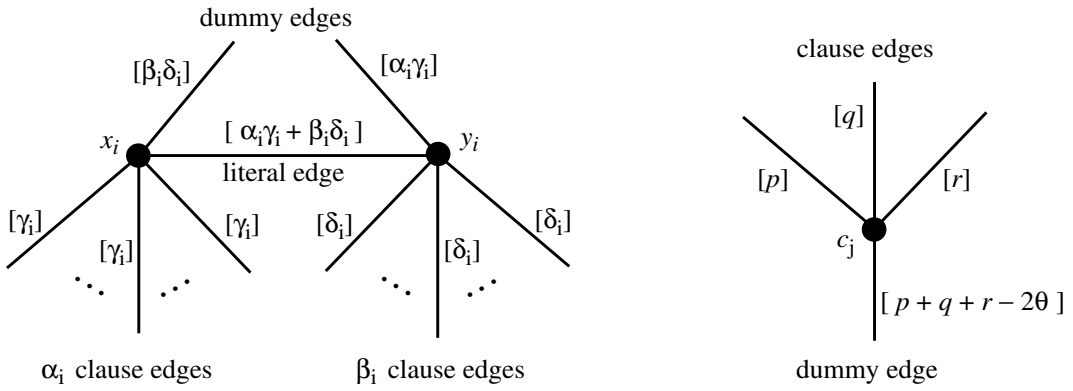

Figure 2.5: Schematic illustration of the edges incident on the literal and clause vertices of network N: **(a)** literal vertices x_i and y_i; **(b)** clause vertex c_j.

- The amount of flow through the literal, clause and the dummy edges incident on literal vertices is equal to their capacities. The flow therefore satisfies the range property in these edges.

- If a literal is true then its incident literal edge is incoming and its incident clause edges with nonzero capacity range and its dummy edge are outgoing. If a literal is false then its incident literal edge is outgoing and its incident clause edges with nonzero capacity and its dummy edge are incoming. In either case the amount of flow coming into a literal l_i is $\alpha_i\gamma_i + \beta_i\delta_i$ and the amount of flow leaving it is $\alpha_i\gamma_i + \beta_i\delta_i$. Therefore the flow satisfies the conservation property at these vertices.

- If the net flow entering a clause vertex through the clause edges is ν_j, then the flow leaving it through its dummy edge is ν_j. Conversely, if the net flow leaving a clause through the clause edges is ν_j, then the flow entering it through its dummy edge is ν_j. Since A is a satisfying assignment, the three clause edges with nonzero capacity range incident on a clause vertex can not be all incoming or all outgoing. A positive flow in a clause edge has is greater than or equal to θ. Therefore, the net flow coming into or leaving a clause vertex c_j through its clause edges is at most $\eta_j - 2\theta$. Hence, the conservation property at c_j and the range property in its dummy edge edge are both satisfied by the flow.

- The conservation property holds at the dummy vertex because of the following argument. By successive mergers of two nondummy vertices of $\mathcal{N}$ and elimination of the resultant self loops and the flow through them, we can reduce $\mathcal{N}$ into a graph with multiple edges between two vertices– the dummy vertex and another vertex u for which the conservation property holds. The net flow between these two vertices is same as in f. Since the conservation property holds for u and the dummy vertex is neither a source nor a sink, the conservation property holds for the dummy vertex also.

□

Now, starting from $\mathcal{S}$, we construct a planar switch-flow network $\mathcal{P}$ (see Fig. 2.4). We first construct a *layered drawing* $\psi_{\mathcal{N}}$ of $\mathcal{N}$ as follows (see Fig. 2.4(a)):

- Each literal and clause edge is drawn as a straight-line. The dummy edges are drawn as continuous curves.
- The clause vertices are horizontally aligned and ordered $c_1.c_2, \ldots, c_m$ from left to right.
- The literal vertices horizontally aligned above the clause vertices, and ordered $x_1, x_2, y_1, y_2, \ldots, x_m, y_m$ from left to right.
- There are crossings only between the clause edges. However, no more than two clause edges cross at the same point.

We next replace the crossings of $\psi_{\mathcal{N}}$ with vertices called the *crossing* vertices, thus splitting the clause edges at the crossing vertices. We call *fragment edges* or simply *fragments*, the edges originated by the splitting of the clause edges. Each fragment edge inherits the capacity range of the originating clause edge. We define the *facial degree* of a vertex as the total number of edges in its incident faces.

Lemma 5 *Given network $\mathcal{N}$ representing an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with $n \geq 3$ variables and $m \geq 3$ clauses, the associated planar switch-flow network $\mathcal{P}$ is triconnected, has $O(n^2m^2)$ vertices and edges, and can be constructed in $O(n^2m^2)$ time. Also, in the unique embedding of $\mathcal{P}$, the facial degree of each vertex is at most $7nm$.*

Proof: Let $\psi_{\mathcal{N}}$ be the layered drawing of $\mathcal{N}$ that is used to construct $\mathcal{P}$ (see Fig. 2.4(a)). There are $O(n^2m^2)$ crossings in $\psi_{\mathcal{N}}$. Therefore, $\mathcal{P}$ has $O(n^2m^2)$ crossing vertices and fragment edges. Consequently, $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges.

Now we show that $\mathcal{P}$ is triconnected. We denote by u_k, the crossing vertex that corresponds to the crossing between the line joining c_k and y_n, and the line joining c_{k+1} and x_1. We denote by v_k, the crossing vertex that corresponds to the crossing between the line joining y_k and c_m, and the line joining x_{k+1} and c_1. We call the vertices of type u_k and v_k as the *bounding* crossing vertices of $\mathcal{P}$. No two bounding vertices are identical because they correspond to crossing between different pairs of lines. Hence there are two vertex disjoint paths $p = x_1y_1v_1 \ldots x_ky_kv_k, x_{k+1} \ldots y_n$ and $q = c_1u_1c_2 \ldots c_ku_kc_{k+1} \ldots c_m$ in $\mathcal{P}$. From each vertex there are two vertices disjoint (possibly empty) paths, one to a clause vertex and the other to a literal vertex. For the dummy vertex z, these paths consist of the dummy edges; For every nondummy vertex, these paths consist of the fragment edges of a clause edge whose fragment edges are incident on it. We have that between any two nondummy vertices there are two vertex disjoint paths not containing z. One such path contains a (possibly empty) subpath of p, and the other path contains a (possibly empty) subpath of q. Hence graph $\mathcal{P} - \{z\}$ is biconnected. Now suppose, for a contradiction, that $\mathcal{P}$ is not triconnected. Since the graph $\mathcal{P} - \{z\}$ is biconnected, z is not a member of a separating pair of $\mathcal{P}$. Let C_1 and C_2 be two connected components of $\mathcal{P}$ obtained after removing a separating pair. Suppose z is in C_1. Hence there are at

most two vertex disjoint paths between z and the vertices in C_2. But between a vertex w of $\mathcal{P}$ and z there are at least four vertex disjoint paths (e.g., for a crossing vertex v, the four paths go through the endpoints of the two clause edges of $\mathcal{N}$ whose crossing is associated with v). Thus, we obtain a contradiction. Therefore $\mathcal{P}$ is triconnected.

Now we show that the facial degree of any vertex is at most $7nm$. All the faces incident on the dummy vertex z have at most four vertices: one is z, two of them are clause and/or literal vertices and the fourth (if present) is a bounding crossing vertex. There are $n-1+m-1$ bounding crossing vertices in $\mathcal{P}$. A simple counting argument shows that the facial degree of z is $3n+m+2(n-1+m-1)+2 = 5n+3m-2$. Let f be a face that does not contain z. Face f contains at least one crossing vertex, and at most two fragments of clause edges incident on the same clause or literal vertex. Hence, the number of edges in face f is at most $\min\{4n, 2m\}$. The degree of each nondummy vertex is at most $\max\{2n+1, m+2, 4\}$. Consequently, the facial degree of each vertex is at most $7nm$, for $n, m \geq 3$.

Finally we show how to construct $\mathcal{P}$ from $\mathcal{N}$. Suppose the literals are numbered from 1 to $2n$ so that literal $l_{2k-1} = x_k$ and $l_{2k} = y_k$. We inductively construct a layered drawing $\psi(k, m)$ of the subgraph of $\mathcal{N}$ induced by literals $l_1, \cdots, l_k$ and clauses $c_1, \cdots, c_m$ (see Fig. 2.6). Drawing $\psi(2, m)$ is shown in Fig. 2.6(a). Suppose we have already constructed drawing $\psi(k-1, m)$ (see Fig. 2.6(b)). We place literal l_k at the same height as l_{k-1} and sufficiently far to its right so that that edge (l_k, c_1) intersects each clause edge of $\psi(k-1, m)$ below its lowest crossing. Replacing the crossings of $\psi(2n, m)$ by crossing vertices gives us the planar network $\mathcal{P}$.

The construction of the network $\mathcal{P}$ does not require computing the exact coordinates of the vertices and crossings of $\psi(2n, m)$. In the actual implementation, $\mathcal{P}$ is constructed directly by maintaining and updating at each inductive step a list of the lowest crossing vertices of the clause edges. The manipulation of this list takes constant time per crossing vertex and hence can be done in total $O(n^2m^2)$ time. The rest of the construction can also be carried out in $O(n^2m^2)$ time.

□

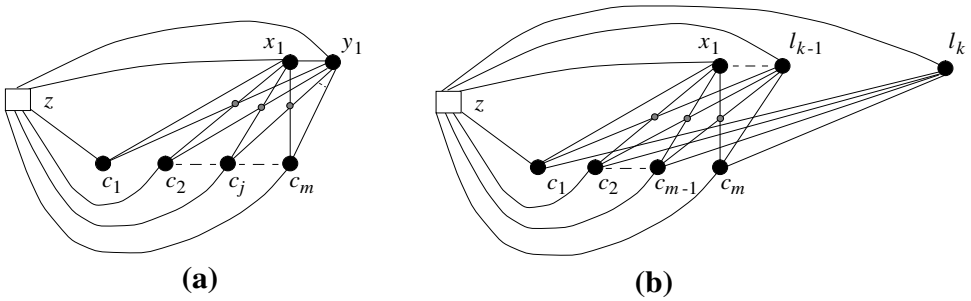

Figure 2.6: Proof of Lemma 5: **(a)** drawing $\psi(2, m)$; **(b)** drawing $\psi(k, m)$. The small circles show the lowest crossings of the clause edges in the drawing of $\psi(k-1, m)$.

Lemma 6 *Network $\mathcal{N}$ admits a feasible flow if and only if network $\mathcal{P}$ admits a feasible flow, and a feasible flow for $\mathcal{N}$ can be computed from a feasible flow for $\mathcal{P}$ in $O(n^2m^2)$*

time.

Proof: The *only if* part is trivial. Given a feasible flow in $\mathcal{N}$, we can get a feasible flow in $\mathcal{P}$ in which the flow through each fragment edge is same as in the corresponding clause edge in $\mathcal{N}$, and the flow through the dummy and literal edges is same as in the corresponding edges in $\mathcal{N}$.

For proving the *if* part suppose that $\mathcal{P}$ admits a feasible flow f. In f, at any crossing vertex, of the two fragment edges of a clause edge incident on it, one is incoming and the other is outgoing. This is so because the fragment edges of different clause edges have different capacities and the conservation property is satisfied at the vertex. Consequently all the fragment edges of a clause edge have the same flow. We can get a feasible flow in $\mathcal{N}$ in which the flow through a clause edge is same as the flow in its fragment edges in f, and the flow through the dummy edges and the literal edges is same as the flow in the corresponding edges in $\mathcal{P}$. □

By combining Lemmas 4, 5 and 6, we obtain the main result of this section.

Theorem 1 *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with $n \geq 3$ variables and $m \geq 3$ clauses, the associated planar switch-flow network $\mathcal{P}$ is triconnected, has $O(n^2m^2)$ vertices and edges, has facial degree at most $7nm$, and can be constructed in $O(n^2m^2)$ time. Instance $\mathcal{S}$ is satisfiable if and only if network $\mathcal{P}$ admits a feasible flow. Also, given a feasible flow for $\mathcal{P}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^2m^2)$.*

2.4 Upward Planarity Testing

In this section we show how to reduce the problem of computing a feasible flow in the planar switch-flow network associated with a NOT-ALL-EQUAL-3-SAT instance to the problem of testing the upward planarity of a suitable digraph.

Let $\mathcal{P}$ be the planar switch-flow network with parameter $\theta = 4$ associated with a NOT-ALL-EQUAL-3-SAT instance $\mathcal{S}$. Now, we construct on orientation $\vec{\mathcal{P}}$ of $\mathcal{P}$ as follows (see Fig. 2.7):

- Every literal edge (x_i, y_i) is oriented from x_i to y_i.
- Every fragment edge is oriented "away from" the clause vertex and "towards" the literal vertex.
- Every dummy edge incident on a literal vertex is oriented towards the dummy vertex, and every dummy edge incident on a clause vertex is oriented towards the clause vertex.

Lemma 7 *In digraph $\vec{\mathcal{P}}$, every vertex has at least one incoming and one outgoing edge, every directed cycle contains the dummy vertex, and there are exactly two faces that are*

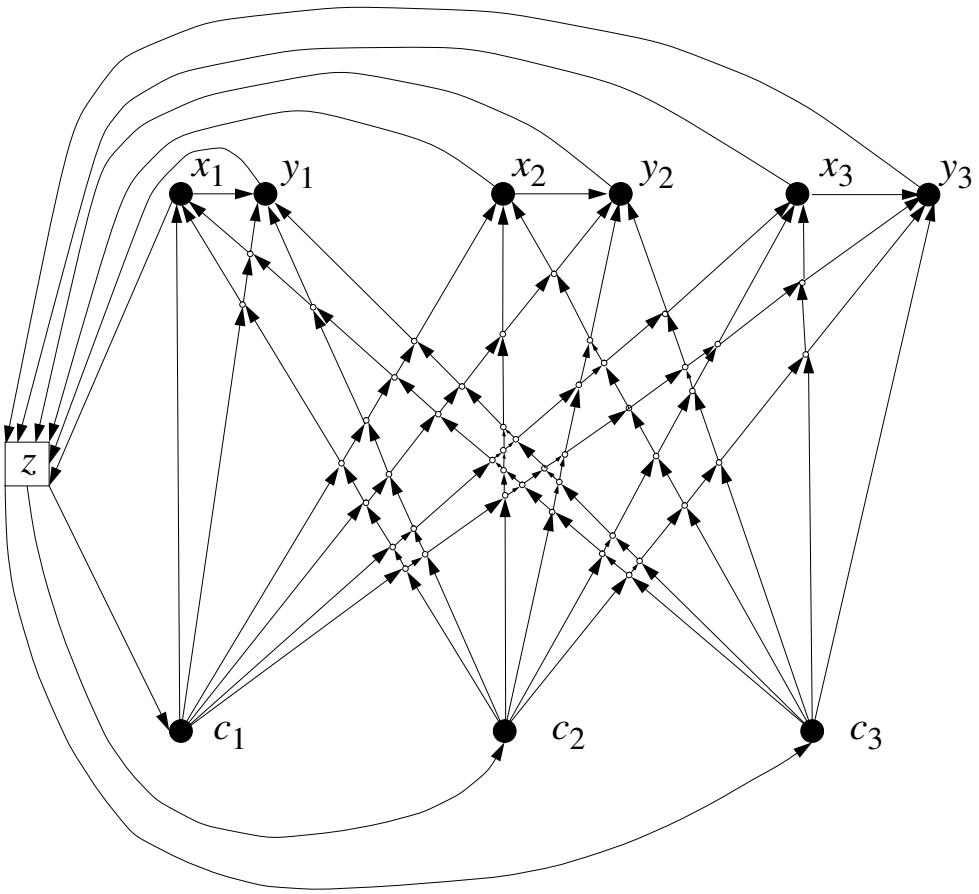

Figure 2.7: Orientation $\vec{\mathcal{P}}$ of the network $\mathcal{P}$ shown in Fig. 2.4(c).

directed cycles. Also, $\vec{\mathcal{P}}$ is bimodal and each face of $\vec{\mathcal{P}}$ consists of at most two directed paths.

Proof: Let m be the number of clauses and n be the number of variables in $\mathcal{S}$. Let z be the dummy vertex of $\vec{\mathcal{P}}$.

Each crossing vertex has two incoming and two outgoing fragment edges. Each literal vertex y_i has $m+1$ incoming edges and one outgoing edge (to z). Each literal vertex x_i has m incoming edges and two outgoing edges (to z and y_i). Each clause vertex has $2n$ outgoing edges and one incoming edge (from z). The dummy vertex has $2n$ incoming edges and m outgoing edges. Therefore, each vertex has at least one incoming and one outgoing edge. It can be easily verified that each vertex of $\vec{\mathcal{P}}$ has the bimodal property and hence $\vec{\mathcal{P}}$ is bimodal.

The digraph $\vec{\mathcal{P}} - \{z\}$ is upward planar with m sources, each being a clause vertex and n sinks, each being a literal vertex y_i. Therefore every directed cycle in $\vec{\mathcal{P}}$ contains z. There are exactly two faces in $\vec{\mathcal{P}}$ that that are directed cycles — one consisting of vertices x_1, z and c_1, and the other consisting of vertices y_n, z and c_m. It can be easily verified that the other faces of $\vec{\mathcal{P}}$ each consist of exactly two directed paths. □

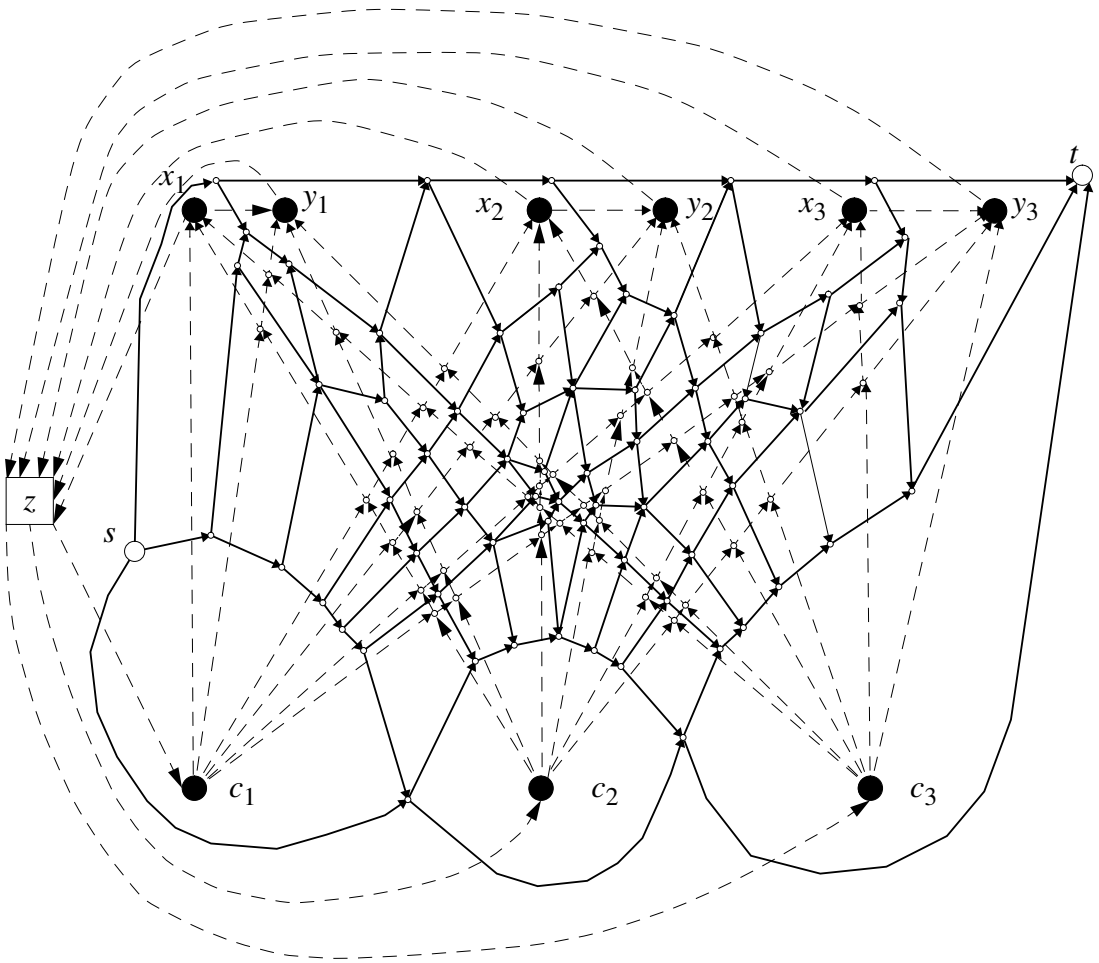

Figure 2.8: Dual digraph $\vec{\mathcal{D}}$ of network $\vec{\mathcal{P}}$ shown in Fig. 2.7.

Since $\mathcal{P}$ is triconnected (see Theorem 1), the planar embedding of $\mathcal{P}$ and the dual graph of $\mathcal{P}$ are unique. We construct the dual digraph $\vec{\mathcal{D}}$ of $\vec{\mathcal{P}}$, by taking the dual graph $\mathcal{D}$ of $\mathcal{P}$ and orienting every dual edge from the face on the left to the face on the right of the primal edge (see Fig. 2.8).

Lemma 8 *The dual digraph $\vec{\mathcal{D}}$ of $\vec{\mathcal{P}}$ is upward planar, triconnected, acyclic and has exactly one source and one sink, denoted with s and t, both of which are on the same face. Also, each face of $\vec{\mathcal{D}}$ has exactly one source and one sink.*

Proof: Because $\vec{\mathcal{P}}$ is planar and triconnected (Theorem 1), so is its dual $\vec{\mathcal{D}}$. By Lemma 7, exactly two faces of $\vec{\mathcal{P}}$ are directed cycles. Hence $\vec{\mathcal{D}}$ has exactly two switches, denoted by s and t (see Fig. 2.8), corresponding to these two switches, s which is a source and t which is a sink. s corresponds to the face of $\vec{\mathcal{P}}$ consisting of the literal vertex x_1, dummy vertex z and the clause vertex c_1. t corresponds to the face of $\vec{\mathcal{P}}$ consisting of the vertices y_n, z and c_m. Both s and t are on the face that is the dual of z. From Lemma 7, each face of $\vec{\mathcal{P}}$ consists of at most two directed paths and hence each vertex of $\vec{\mathcal{D}}$ has the

bimodal property. Therefore $\vec{\mathcal{D}}$ is bimodal. Again from Lemma 7, each vertex of $\vec{\mathcal{P}}$ has the bimodal property and none of them is a source or a sink. Hence each face of $\vec{\mathcal{D}}$ has exactly one source and one sink. Since s and t are the only switches of $\vec{\mathcal{D}}$ and both of them are on the same face, it follows that there is a consistent assignment of labels to the angles of $\vec{\mathcal{D}}$ in which exactly two angles are labeled large, namely the angles at s and t in their common face. Therefore, by Lemma 1, $\vec{\mathcal{D}}$ has an upward embedding and hence is upward planar. □

Starting from digraph $\vec{\mathcal{D}}$ we construct a new digraph $\vec{\mathcal{G}}$ by replacing the edges of $\vec{\mathcal{D}}$ with subgraphs (tendrils or wiggles), as follows (see Fig. 2.9):

- Every edge of $\vec{\mathcal{D}}$ that is the dual of a literal edge, fragment edge, or dummy edge incident on a literal vertex is replaced with tendril T_c, where $[c]$ is the capacity range of the dual edge. Note that c is a multiple of parameter θ.
- Every edge of $\vec{\mathcal{D}}$ that is the dual of a dummy edge incident on a clause vertex is replaced with wiggle W_c, where $[0 \cdots c]$ is the capacity range of the dual edge.

We distinguish two types of vertices in digraph $\vec{\mathcal{G}}$: we call *primary* vertices, the vertices that are also vertices of $\vec{\mathcal{D}}$, and *secondary* vertices the remaining vertices. There is no directed path from the sink the pole to the source pole of a tendril or a wiggle. Therefore $\vec{\mathcal{G}}$ is also acyclic. The following lemma is immediate.

Lemma 9 *Digraph $\vec{\mathcal{G}}$ is planar and acyclic. Also, the only primary source vertex is s and the only primary sink vertex is t. Both s and t are on the same face.*

By Lemma 8 and the construction of digraph $\vec{\mathcal{G}}$, all the embeddings of $\vec{\mathcal{G}}$ are obtained by choosing one of the two possible flips for each tendril. In an embedding of $\vec{\mathcal{G}}$, the face that corresponds to the dummy vertex of $\mathcal{P}$ is called the *dummy face* of the embedding. The other faces are called the *nondummy* faces.

Lemma 10 *Digraph $\vec{\mathcal{G}}$ is upward planar if and only if the tendrils can be flipped and the wiggles can be arranged such that for every nondummy face the total contribution of the tendrils and wiggles is zero.*

Proof:

If: By Lemma 8, $\mathcal{D}$ has an upward embedding. Let A be the assignment of labels to the angles in this embedding. A is a consistent assignment. Therefore if the tendrils can be flipped and the wiggles can be arranged to get an embedding ψ in which, for every face, the total contribution of the tendrils and wiggles is zero, we obtain from ψ an upward embedding of $\vec{\mathcal{G}}$ by assigning to the angles of the primary vertices of $\vec{\mathcal{G}}$ that are not angles of the internal faces of the tendrils to the same label as in assignment A.

Only If: Suppose $\vec{\mathcal{G}}$ has an upward embedding ψ. Let B be the assignment of labels to the angles of ψ. By Lemma 1, B is a consistent assignment, so that, denoting with

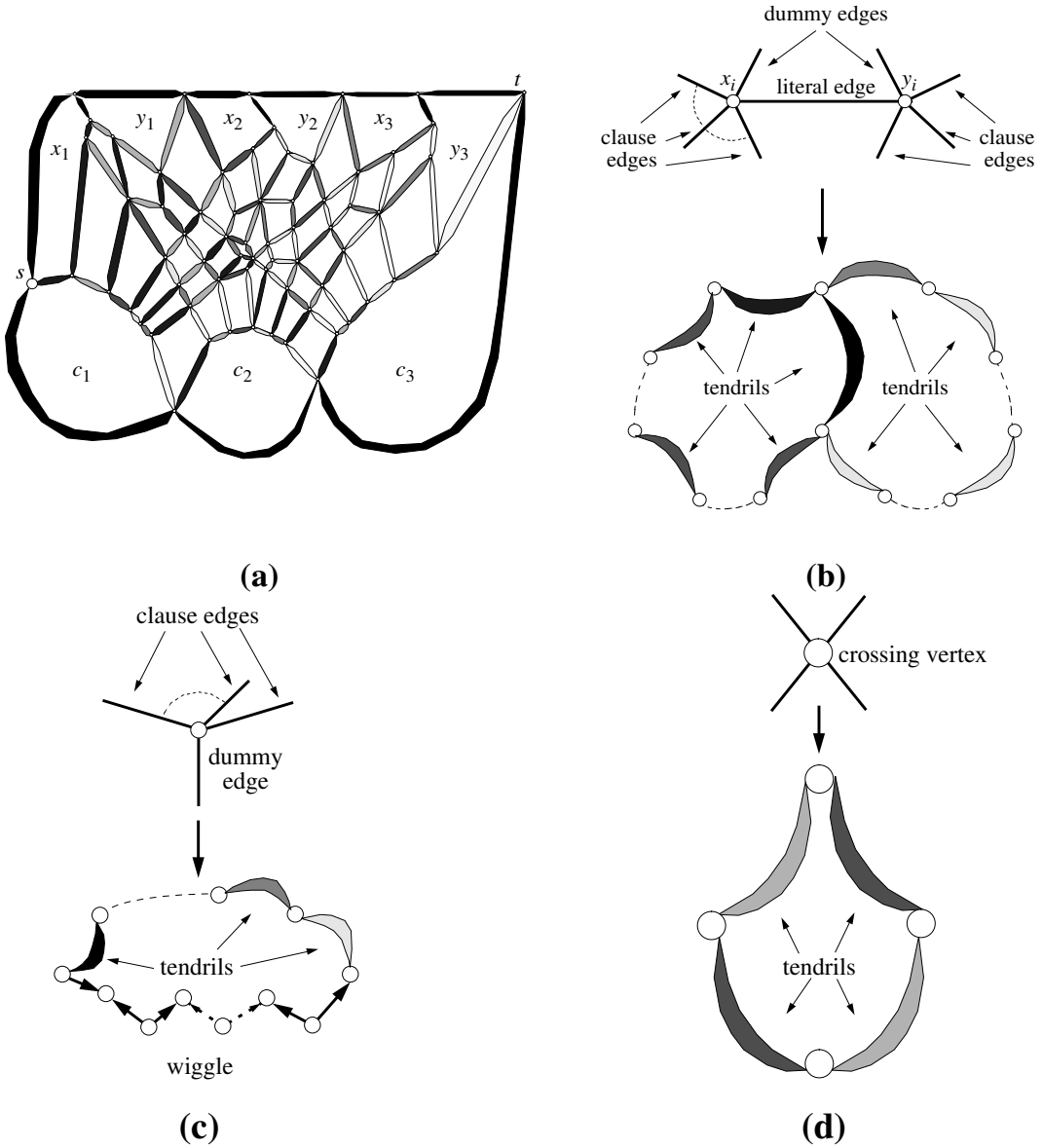

Figure 2.9: (**a**) Schematic illustration of digraph $\vec{\mathcal{G}}$ obtained from $\vec{\mathcal{D}}$ by replacing edges with tendrils and wiggles. (**b**) The two faces of $\vec{\mathcal{G}}$ associated with literal vertices x_i and y_i of $\mathcal{P}$. (**c**) The face of $\vec{\mathcal{G}}$ associated with a clause vertex of $\mathcal{P}$. (**d**) The face of $\vec{\mathcal{G}}$ associated with a crossing vertex of $\mathcal{P}$.

$L(f)$ and $S(f)$, the number of angles of face f with label *large* and *small*, respectively, we have that $|L(f) - S(f)| = 2$ for every face f of ψ.

Let f be a nondummy face of ψ. By Lemma 8, f has exactly one primary source vertex and one primary sink vertex. Let us denote them by u and v. Let $\tau(f)$ be the contribution of the tendrils to f. Recall from Section 2.2.2, that the contribution of a tendril T_k is either $2k$ or $-2k$. Therefore, $\tau(f)$ is a multiple of $2\theta = 8$.

We have two cases:

Case 1: f corresponds to a literal or a crossing vertex of $\mathcal{P}$. Face f has no wiggles and exactly two angles between incoming or outgoing edges that are not angles of the tendrils of f. Thus, $L(f) - S(f) = \tau(f) + t$, with $|t| \leq 2$. By Lemma 1, $|L(f) - S(f)| = 2$. Hence, $|\tau(f) + t| = 2$, with $|t| \leq 2$. Since $\tau(f)$ is a multiple of 8, this implies $\tau(f) = 0$.

Case 2: f corresponds to a clause vertex of $\mathcal{P}$. Face f has three tendrils, one wiggle, and exactly two angles between incoming or outgoing edges that are not angles of the tendrils or wiggle of f. Let $\omega(f)$ be the contribution of the wiggle. We have $L(f) - S(f) = \tau(f) + \omega(f) + t$, with $|t| \leq 2$. By Lemma 1, $|L(f) - S(f)| = 2$. Hence, $|\tau(f) + \omega(f) + t| = 2$, with $|t| \leq 2$.

Let η_j be the sum of the absolute values of the contributions of the tendrils to f. By construction, we have $|\omega(f)| \leq \eta_j - 8$. We claim that $|\tau(f)| \leq \eta_j - 8$. Indeed, since $\tau(f)$ is a multiple of 8, $|\tau(f)| > \eta_j - 8$ implies $|\tau(f)| = \eta_j$ and thus $|\tau(f) + \omega(f)| \geq 8$, a contradiction.

Since the wiggle of f can be rearranged so that $\omega(f)$ takes any even value between $-\eta_j + 8$ and $\eta_j - 8$, ψ can be modified so that $\tau(f) + \omega(f) = 0$ without affecting the other faces of the embedding.

□

We establish the following correspondence between digraph $\vec{\mathcal{G}}$ and network $\mathcal{P}$ (see Fig. 2.9):

- The faces of $\vec{\mathcal{G}}$ correspond to the vertices of $\mathcal{P}$.
- The tendrils and wiggles of $\vec{\mathcal{G}}$ correspond to the the edges of $\mathcal{P}$.
- Flipping a tendril T_k of $\vec{\mathcal{G}}$ corresponds to orienting an edge e of $\mathcal{P}$, where e is the dual of the edge replaced by T_k in constructing $\vec{\mathcal{G}}$ from $\vec{\mathcal{D}}$. Edge e is oriented towards the dual vertex of a face f of $\vec{\mathcal{G}}$ if and only if T_k contributes $2k$ to f.
- The contribution of a tendril or wiggle U of $\vec{\mathcal{G}}$ corresponds to the flow in an edge e of $\mathcal{P}$. e is the dual of the edge which is replaced by U in constructing $\vec{\mathcal{G}}$ from $\vec{\mathcal{D}}$. The contribution of U to a face f is equal to the amount of the flow coming into the dual vertex of f through edge e.

- The balance of the contribution of the tendrils and wiggles to the faces of $\vec{\mathcal{G}}$ corresponds to the conservation of flow at the vertices of $\mathcal{P}$, i.e., the total contribution of the tendrils and wiggles to a face f is zero if and only if there is a conservation of flow at the dual vertex of f.

Theorem 2 *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses and the associated planar switch-flow network $\mathcal{P}$, digraph $\vec{\mathcal{G}}$ associated with $\mathcal{S}$ and $\mathcal{P}$ has $O(n^3m^2)$ vertices and edges, and can be constructed in $O(n^3m^2)$ time. Instance $\mathcal{S}$ is satisfiable and network $\mathcal{P}$ admits a feasible flow if and only if digraph $\vec{\mathcal{G}}$ is upward planar. Also, given an upward planar embedding for $\vec{\mathcal{G}}$, a feasible flow for $\mathcal{P}$ and a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^3m^2)$.*

Proof: The number of vertices and edges in a tendril or a wiggle is $O(n)$. Since $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges, $\mathcal{S}$ has $O(n^3m^2)$ vertices and edges. $\mathcal{P}$ can be constructed from $\mathcal{S}$ in $O(n^2m^2)$ time and $\vec{\mathcal{G}}$ can be constructed from $\mathcal{P}$ in $O(n^3m^2)$ time giving a total time complexity of $O(n^3m^2)$ for the construction of $\vec{\mathcal{G}}$ from $\mathcal{S}$.

From Lemma 10 and the correspondence established above between a feasible flow in $\mathcal{P}$ and the upward planarity of $\vec{\mathcal{G}}$, we have that instance $\mathcal{S}$ is satisfiable and network $\mathcal{P}$ admits a feasible flow if and only if digraph $\vec{\mathcal{G}}$ is upward planar. We also obtain that given an upward planar embedding for $\vec{\mathcal{G}}$, a feasible flow for $\mathcal{P}$ and a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^3m^2)$. □

From Theorems 1 and 2 we conclude:

Corollary 1 *Upward planarity testing is NP-complete.*

2.5 Conclusions

The problem of finding an efficient algorithm for upward planarity testing had been open for many years. In this chapter we have shown that the upward planarity testing problem is NP-complete, and hence a polynomial time algorithm is unlikely to exist.

Chapter 3

Upward Planar Grid Drawings of Trees

An important criterion for a drawing of a graph is that it takes up as little area as possible. This is motivated by the finite resolution of all of our current technologies for rendering a drawing, and also by circuit-area optimization criteria in VLSI layout [12, 75, 109]. In the following, we assume the existence of a *resolution rule* that implies a finite minimum area for the drawing of any graph. A typical resolution rule is to require *grid* drawings, where the vertices and bends of the edges have integer coordinates. Indeed, this consideration recently motivated the re-examination of straight-line drawings of planar directed graphs, because they require exponentially-large area [37], whereas several researchers have recently shown that planar graph drawings require only quadratic area, and that such drawings can be produced in linear time [26, 68, 94]. Moreover, some very nice work by Kant [68] shows that a number of other aesthetic criteria (such as convex faces) can be satisfied for a planar drawing while still keeping the area quadratic.

In this chapter we consider the problem of constructing area-efficient upward planar grid drawings of rooted trees. An *upward* drawing of a tree T is one in which every edge of T is drawn as a vertically monotone chain from the child to the parent, so that the parent u of a node v has y-coordinate greater than or equal to the one of v. This is the natural way in which rooted trees are usually drawn to display their hierarchic structure (e.g., see any undergraduate text in data structures). The difficulty is that most of the known techniques for constructing planar upward grid drawings of trees require $\Omega(N^2)$ area in the worst case [89, 98].

3.0.1 Previous Work

If we relax the upward requirement, however, then, as independently shown by Leiserson [75] and Valiant [109], one can construct an $O(N)$-area planar *orthogonal* grid drawing of an N-node tree T, where the nodes are placed at integer grid points and the edges follow paths of the grid. However, Brent and Kung [16] show that if the leaves of an N-node complete binary tree are constrained to be on the convex hull of the drawing,

then the drawing needs $\Omega(N \log N)$ area.

Thus, a natural question is whether $O(N)$ area is still achievable for planar upward drawings. Crescenzi, Di Battista, and Piperno [21] have recently provided a negative answer to this question for the case of *strictly upward* grid drawings, where the nodes have integer coordinates, and the parent of a node has y-coordinate strictly greater than the ones of its children. Namely, they exhibit a family of binary trees that require $\Omega(N \log N)$ area in any strictly upward planar grid drawing. This lower bound is tight within a constant factor: Shiloach [95] and Crescenzi, Di Battista, and Piperno [21] give linear-time algorithms that construct a strictly upward planar straight-line grid drawing of an N-node rooted tree with $O(N \log N)$ area, $O(N)$ height, and $O(\log N)$ width.

Their result doesn't settle the question for the standard notion of upward drawing, however, which allows a child node to be on the same horizontal line as its parent, as long as it is not above its parent. In addition, it should be noted that producing the exact minimization of the area of the drawing of a tree is NP-hard under several drawing conventions [11, 15, 42]. Nevertheless, Crescenzi, Di Battista, and Piperno [21, 22] give $O(N)$-area planar straight-line upward grid drawings of AVL trees. They do not, however, give a general construction for other types of trees.

An *hv-drawing* of a binary tree has the following properties: (i) nodes have integer coordinates; (ii) an edge between a node v and its parent u is drawn either as a horizontal segment, with u to the left of v, or as a vertical segment, with u above v; (iii) the enclosing rectangles of the drawings of the subtrees rooted at the children of a node are disjoint. Hence, hv-drawings are a special case of upward planar straight-line drawings. Eades, Lin, and Lin [43] show how to construct in $O(N\sqrt{N \log N})$ time a minimum-area hv-drawing of an N-node binary tree. However, they do not provide specific bounds on the area requirement of hv-drawings.

Related results on the area requirement of visibility representations of trees are given in [71].

3.0.2 Our Results

We show that, for any rooted bounded-degree tree T with N nodes, one can construct a planar upward grid drawing of T with $O(N)$ area in $O(N)$ time, and that such drawing can have width $O(N^{\alpha})$, for any prespecified constant α such that $0 < \alpha < 1$. The latter feature provides great flexibility to applications that need to fit the drawing in a prescribed region of the plane. Our algorithm for constructing such drawings uses a two step process. In the first step, it constructs an *upward layering* of T, which is an assignment of the nodes of T to horizontal layers such that a parent is placed either at the same layer as its children or at a higher layer. Notice that the end-points of an edge e may get assigned to non-consecutive layers; we say that e *traverses* the layers that are between the layers to which its end-points are assigned. In the second step, it first introduces dummy nodes in each edge of tree, one for every layer traversed by the edge, and assigns x-coordintates to the nodes and bends (each bend corresponds to a dummy node) by using an inorder-traversal of the tree. The key-part of our algorithm is the way an upward layering is constructed in the first step. This is done by first converting T

into a left-heavy tree T' and then repeatatively adding (nodes contained in the) leftmost paths of the subtrees of T' into an initially empty ordered sequence Λ. The order of nodes in Λ is then used to assign the nodes to layers. Details are given in Section 3.2. We also extend our approach to trees of arbitrary maximum degree d, at a small additional cost when d exceeds $N^{1-\epsilon}$ for any $\epsilon > 0$. Our drawings do not preserve the left-to-right ordering of the children in T, however. But this should not be surprising, for we show that if one requires a planar upward tree drawing to preserve the left-to-right ordering, then the drawing requires at least $\Omega(N \log N)$ area in the worst case, and we show that this is tight to within a constant factor. Our $O(N)$-area drawing is for the *polyline grid* model, where the nodes of T are mapped to integer grid points, and the edges of T are mapped to polygonal chains with bends at grid points. These polygonal chains need not follow along grid edges, however.

If one desires such a drawing, then in Section 3.3, we show that one can construct a planar upward orthogonal grid drawing of an N-node binary tree T with $O(N \log \log N)$ area in $O(N)$ time. Our algorithm uses a recursive-decomposition technique in which the tree is first decomposed into small trees called *blocks*, each with $\Theta(\log n)$ nodes, by recursively deleting the 1/3-2/3 separators (see Chazelle [17]) of the trees obtained during the course of recursion. Each block is then drawn suboptimally with height $O(\log n \log n)$ and width $O(\log n)$ using a simple algorithm. Next, the drawings of the blocks are stacked one above the above, and the separators which were deleted earlier are routed between them, giving us a drawing of T with $O(N \log \log N)$ area. This $\log \log N$ factor in the area may, at first, seem unnatural, but we show that it is not, for we give an N-node binary tree that requires $\Omega(N \log \log N)$ area for any upward orthogonal grid drawing. Thus, we show that there is an intermediate case between the $\Theta(N)$ area achievable for non-upward planar orthogonal grid tree drawings and the $\Theta(N \log N)$ area achievable for strictly-upward planar grid drawings, or for upward planar grid drawings that preserve the left-to-right order. It is also interesting to observe that the upward requirement penalizes the area less than the requirement of placing the leaves on the same horizontal line, for which the $\Omega(N \log N)$ area bound also applies [16]. We summarize the previous and current bounds on planar grid tree drawings in Table 3.1.

Most of the results and techniques presented in this chapter can also be found in [53, 52].

3.1 Preliminaries

In this section we give definitions that will be used throughout the chapter.

A *drawing* Γ of a graph G maps each vertex of G to a distinct point of the plane and each edge (u, v) of G to a simple Jordan curve with endpoints u and v. We say that Γ is a *straight-line* drawing (see Fig. 3.1(a)) if each edge is a straight-line segment. Γ is a *polyline* drawing (see Fig. 3.1(b)) if each edge is a polygonal chain, and we call *bends* the intermediate vertices of the chain that are not vertices of G. Γ is an *orthogonal* drawing (see Fig. 3.1(c)) if each edge is a chain of alternating horizontal and vertical segments. A *grid* drawing is such that the vertices and bends along the edges have integer coordinates. *Planar* drawings, where edges do not intersect, are especially important

Drawing	Tree Type	Previous Bounds	Our Bounds
Upward Straight-Line	general	$O(N \log N)$ [21, 95]	
Upward Straight-Line	AVL, Fibonacci, complete binary	$\Theta(N)$ [21, 22]	
Strictly-Upward Straight-Line	general	$\Theta(N \log N)$ [21]	
Upward Polyline	degree $O(n^{\delta})$, $\delta < 1$		$\Theta(N)$ (§ 3.2.2)
Upward Ordered Polyline	degree $O(1)$		$\Theta(N \log N)$ (§ 3.2.3)
Leaves-on-Hull Orthogonal	binary	$\Theta(N \log N)$ [16]	
Non-Upward Orthogonal	degree $O(1)$	$\Theta(N)$ [75, 109]	
Upward Orthogonal	binary		$\Theta(N \log \log N)$ (§ 3.3)

Table 3.1: Area-requirements for various types of planar grid drawings of a rooted tree with N nodes.

because they improve the readability of the drawing, and, in the context of VLSI layouts, they simplify the design process [12, 75, 109]. An *upward* drawing of a directed graph is such that every edge is a curve monotonically nondecreasing in the vertical direction (when traversed along the direction of the edge).

The *area* of a drawing Γ is the area of the smallest rectangle R with sides parallel to the axes covering the drawing. The *width* and *height* of Γ are the *width* and *height* of R, respectively. We assume the existence of a *resolution rule* that implies a finite minimum area for the drawing of any graph. A typical resolution rule is to require grid drawings. When a resolution rule is given, it is meaningful to consider the problem of finding drawings with minimum area.

An *ordered tree* is a rooted tree with a prespecified left-to-right order of the children of each node. Let T be an ordered tree. We assume that each edge of T is directed from the child to the parent. The ordering of the children of a node v will be referred to as their left-to-right order. Hence, the first and last children of v will be referred to as the leftmost child and rightmost child of v, respectively. The *degree* of a node of T is the number of its children. Tree T is said to be *left-heavy* (see Fig. 3.2(a)) if, for every node v of T, the children of v are ordered by nonincreasing size of their subtrees. A *leftmost path* of T is a maximal path consisting of nodes that are leftmost children, except the last node.

A *binary tree* is defined as a rooted tree such that each node has at most two children. Examples of planar upward drawings of a binary tree are given in Fig. 3.1.

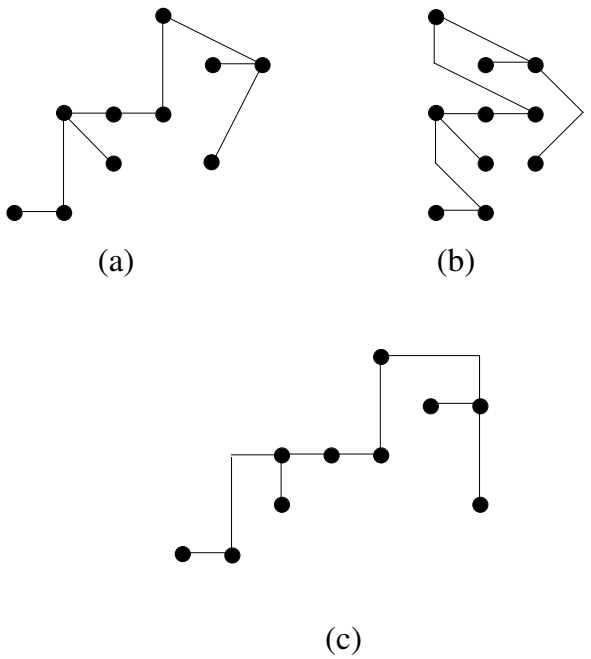

Figure 3.1: Example of planar upward drawings of the same binary tree: **(a)** straight-line; **(b)** polyline; **(c)** orthogonal.

3.2 Polyline Drawings

In this section we investigate polyline drawings. First, we describe a layering technique that will be used to construct the drawings.

3.2.1 Upward Layerings

We define the *inorder visit* of a rooted ordered T as follows:

1. recursively visit the first subtree of T;
2. visit the root of T;
3. recursively visit the other subtrees of T, in left-to-right order.

An *upward layering* of T is a mapping λ of the nodes of T to nonnegative integers that satisfies the following properties (see Fig. 3.2(a)):

1. If w is the leftmost child of v, then $\lambda(v) \leq \lambda(w)$;
2. If w is a child of v but not the leftmost child, then $\lambda(v) < \lambda(w)$.
3. If u is the root of T, then $\lambda(u) = 0$.

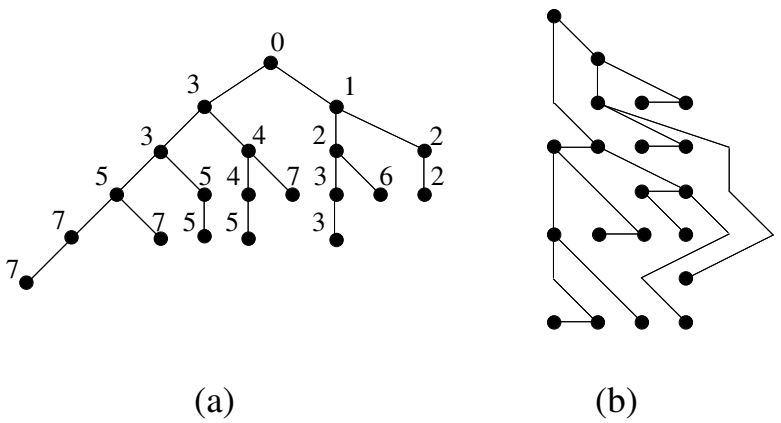

Figure 3.2: **(a)** Example of upward layering λ of a left-heavy tree. **(b)** Drawing associated with λ.

We say that a node v is assigned to layer i if $\lambda(v) = i$. An edge (u, v) is said to *traverse* layer i if $\lambda(v) < i < \lambda(u)$. The *height* of upward layering λ is defined as $\max_{v \in T} \lambda(v)$. The *width* of a layer i is the number of nodes assigned to layer i plus the number of edges that traverse layer i. The *width* of λ is the maximum width of a layer.

The following theorem shows that an upward layering can be extended to a planar polyline upward grid drawing where the nodes are placed along horizontal lines associated with the layers (see Fig. 3.2(b)).

Theorem 3 *Given an upward layering λ with height H and width W of an N-node ordered tree T, a planar polyline upward grid drawing of T with height W and width H can be constructed in $O(H \cdot W)$ time.*

Proof: First, we insert dummy nodes along the edges that traverse layers. Namely, if edge (u, v) traverses layers i through j, we insert $j - i + 1$ dummy nodes along (u, v), and assign them to layers i through j, respectively. Let T' be the resulting tree. For each node v of T', we set $y(v) = -\lambda(v)$, and $x(v)$ equal to the number of nodes of layer $\lambda(v)$ preceding v in the inorder visit. The edges of T' are then drawn as straight-line segments. Clearly, this yields a straight-line upward grid drawing of T' with height H and width W, such that every edge either joins nodes of consecutive layers, or joins a leftmost child to its parent on the same level. We claim that the drawing is also planar. To prove the claim, we observe: (a) a horizontal edge (u, v) on layer i cannot be crossed because all the nodes between u and v in the inorder sequence are assigned to layers below i; (b) if there were a crossing between edges (v', w') and (v'', w''), where w' precedes w'' in layer i and v'' precedes v' in layer $i+1$, then we would have that w' precedes w'' in the inorder sequence but v' follows v'' in the inorder sequence, a contradiction. Finally, we obtain a planar polyline upward grid drawing of T by replacing the dummy nodes of T' with bends. The height and width are not affected. The above construction can be easily carried out in time $O(H \cdot W)$. □

Therefore it is sufficient for us to describe how to construct an upward layering of a tree.

3.2.2 Drawing Algorithm

In this section we describe an algorithm for constructing an upward layering of an N-node rooted tree. We show that if the tree has bounded degree, an upward layering with width $O(N^\alpha)$ and height $O(N^{1-\alpha})$ can be constructed for any constant α such that $0 < \alpha < 1$.

So, let T be a left-heavy ordered rooted tree with N nodes (we will show how to remove this left-heavy restriction later). The following algorithm constructs an upward layering λ of T with width $O(N^\alpha + d \log N)$ and height $O(N^{1-\alpha})$. The algorithm incrementally assembles an ordered sequence Λ of nodes of T, and marks some nodes of T, such that the following invariants are maintained:

1. If u precedes v in Λ, then $\lambda(u) \geq \lambda(v)$; and
2. a node is marked if and only if it is the first node of its layer contained in Λ.

The assembly of Λ is performed by repetitive insertions of leftmost paths. At the end of the computation, the sequence Λ contains all the nodes of T, so that Λ and the marking of the nodes uniquely identify the upward layering λ. Note that in the sequence Λ a child precedes its parent, while in the layering λ, a child is assigned to a layer number greater than or equal to the one of its parent. The algorithm consists of three steps:

1. *Preprocessing:* Initialize Λ as the leftmost path containing the root of T.
2. *Main Loop:*
   for $k = 1, \cdots, \log N$ **do**
   (* execute round k *)
   (a) Select the nodes v of T such that
   - v is a child of a node of Λ;
   - v is not already in Λ; and
   - the subtree rooted at v has size at least $N/2^k$.

   (b) Sort the selected nodes according to the order of their parents in Λ.
   (c) Partition the sorted sequence of selected nodes into blocks of $2^{\alpha k}$ nodes (with the last block possibly having fewer nodes).
   (d) For each block $(v_1, \cdots, v_m)$, let u be the parent of v_1. Mark u and successively insert into Λ the leftmost paths of $v_1, \cdots, v_m$, right before u.

   endfor
3. *Postprocessing:* Scan sequence Λ, and mark a node if it has $N^\alpha - 1$ unmarked predecessors.

Layering λ is finally constructed by assigning node v to layer i if there are i marked nodes following v in Λ.

The algorithm is illustrated in Fig. 3.3. An example of the layering λ produced (for a different tree) is shown in Fig. 3.2.

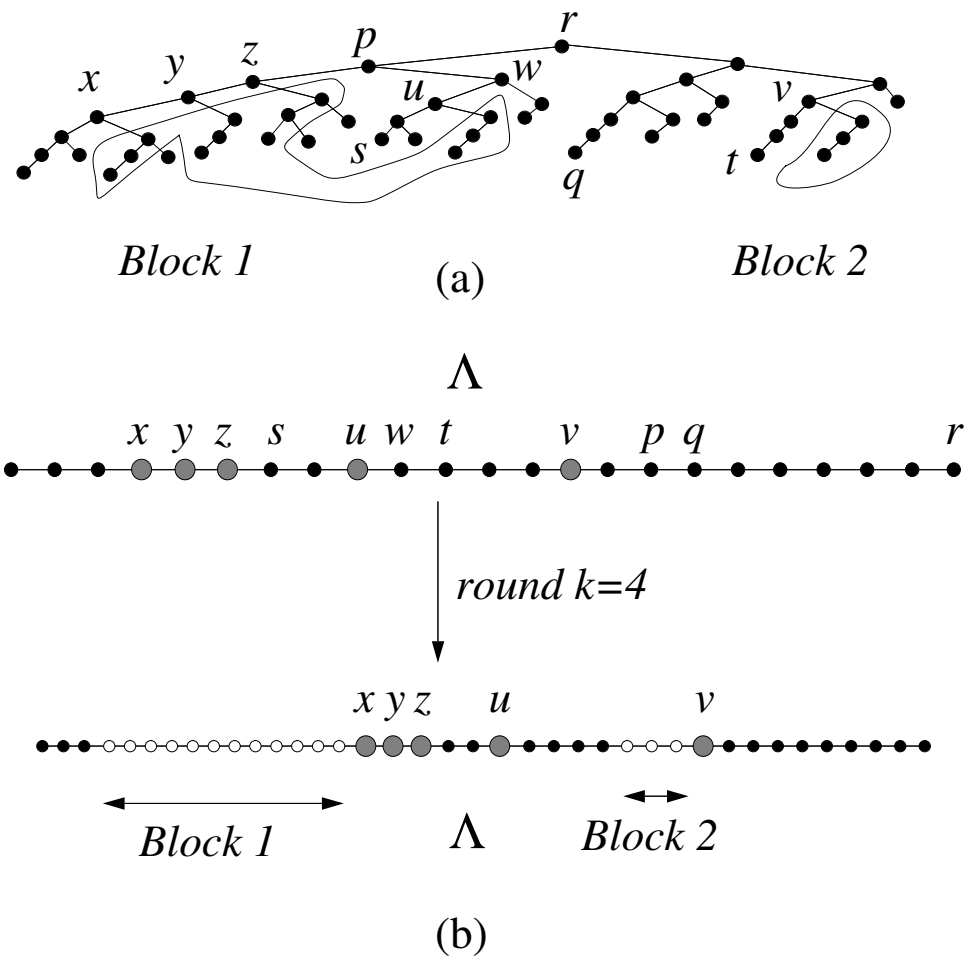

Figure 3.3: Illustration of the execution of round 4 of the upward layering algorithm of Theorem 4. Here, $N = 50$, $\alpha = 1/2$ and $k = 4$. **(a)** Tree. **(b)** Two blocks (of size at most 4) are created in Step 2c, and nodes x and v are marked in Step 2d.

Theorem 4 *Let α be a constant such that $0 < \alpha < 1$. Given an N-node left-heavy rooted ordered tree T with maximum degree d, an upward layering λ of T with height $H = O(N^{1-\alpha})$ and width $O(N/H + d \log N)$ can be constructed in $O(N)$ time.*

Proof: Every node is eventually inserted into Λ. Namely, after round k all the nodes in the leftmost path of a subtree of size $N/2^k$ are inserted into Λ. Hence, every node is assigned to a layer. All the children of a node v precede v in Λ. Also, if u is a child of v but is not the leftmost child of v, then $\lambda(u) > \lambda(v)$ because either v is marked, or there is a marked node between u and v in Λ. Hence, λ is an upward layering.

We say an edge (u, w) of T is *across* a marked node v if u precedes v, and v precedes w in Λ. After round k, the number of edges across a node is increased by at most $2^{\alpha k} + d - 1$. Hence, at the end of the main loop the number of edges across a marked node is at most

$$\sum_{k=1}^{\log N} \left(2^{\alpha k} + d - 1\right).$$

Since, $\sum_{i=1}^{m} a^i = a(a^m - 1)/(a - 1)$, we get

$$\sum_{k=1}^{\log N} \left(2^{\alpha k} + d - 1\right) = \frac{2^\alpha}{2^\alpha - 1}(N^\alpha - 1) + (d - 1)\log N.$$

The postprocessing step does not increase the number of edges across a marked node. After the postprocessing step, at most N^α nodes are assigned to a layer. Also, the number of edges that traverse a layer is less than or equal to the number of edges across the marked node of that layer. We conclude that the width of λ is at most

$$N^\alpha + \frac{2^\alpha}{2^\alpha - 1}(N^\alpha - 1) + (d - 1)\log N.$$

Since there are at most 2^k subtrees of size at least $N/2^k$, the number of blocks created in step 2c in round k is at most $\lceil (2^k/2^{\alpha k}) \rceil \leq 2^{(1-\alpha)k} + 1$. Hence, after round k, the number of marked nodes is increased by at most $2^{(1-\alpha)k} + 1$. Hence the total number of nodes marked in the main loop is at most

$$\sum_{k=1}^{\log N} \left(2^{(1-\alpha)k} + 1\right) = \frac{2^{1-\alpha}}{2^{1-\alpha} - 1}(N^{1-\alpha} - 1) + \log N.$$

The postprocessing step marks at most $N^{1-\alpha}$ nodes. The height of λ is equal to the number of marked nodes, so that it is at most

$$N^{1-\alpha} + \frac{2^{1-\alpha}}{2^{1-\alpha} - 1}(N^{1-\alpha} - 1) + \log N.$$

To achieve linear time complexity, we set up a data structure that allows us to efficiently perform Steps 2a–2b. We say that a node is *active* for round k if it is in Λ and has a child not in Λ whose subtree has size at least $N/2^k$. A node is called *active* if it is active for any round k. We maintain $\log N$ lists such that, before round k, the k-th list contains the nodes active for round k. Within each list, the active nodes are in the same relative order as in Λ. An active node can appear in more than one list, and has pointers to its representatives in the lists.

The nodes selected in round k are children of the nodes in list k, so that they can be accessed and sorted in Steps 2a–2b in $O(1)$ time per node. Every node v that has more than one child and gets inserted into Λ at round k becomes a new active node, and its representatives are inserted into the appropriate lists. The insertion in each such list is carried out in a manner similar to insertion in Λ. This can be done in $O(1)$ time per representative in Step 2d. Also, whenever a node becomes inactive for round k, we remove its representative from the k-th list. Again, this can be done in $O(1)$ time per

representative. Therefore, the total time for maintaining the lists is proportional to the maximum total size of the lists. The k-th list can have at most 2^k nodes. Since a node in list k is also in list k' for $k' > k$, we have that the maximum total size of the lists is

$$\sum_{k=1}^{\log N} 2^k \cdot (\log N - k + 1) = 4N - \log N - 1.$$

The remaining computations can be performed in $O(N)$ time, and we conclude that the time complexity of the algorithm is $O(N)$. □

Theorem 5 *Given a rooted tree T with N nodes and maximum degree d, and a constant α such that $0 < \alpha < 1$, a planar polyline upward grid drawing of T with height $H = O(N^{1-\alpha})$ and width $O(N/H + d \log N)$ can be constructed in $O(N)$ time.*

Proof: First, permute the children of a node so that they are ordered by nonincreasing size. The tree so obtained is left-heavy, so that the result follows from Theorems 3–4. □

Theorem 5 implies the existence of optimal area planar upward polyline grid drawings for trees of maximum degree $O(N^\delta)$, for any constant δ such that $0 < \delta < 1$.

Corollary 2 *Given a bounded-degree rooted tree T with N nodes, and a constant α such that $0 < \alpha < 1$, a planar polyline upward grid drawing of T with area $O(N)$, height $H = O(N^{1-\alpha})$ and width $O(N/H)$ can be constructed in $O(N)$ time.*

Corollary 3 *Let α and δ be constants such that $0 < \delta < \alpha < 1$. Given a rooted tree T with N nodes and maximum degree $O(N^\delta)$, a planar polyline upward grid drawing of T with area $O(N)$, height $H = O(N^{1-\alpha})$ and width $O(N/H)$ can be constructed in $O(N)$ time.*

The algorithm described in this section has been implemented for binary trees. The drawing produced for a complete binary tree with 63 nodes and $\alpha = 1/2$ is shown in Fig. 3.4.

3.2.3 Order Preserving Drawings

The drawings obtained with the above algorithm do not preserve the left-to-right order of the children. This is justified, however, by the area lower bound given in Theorem 6. Our proof of theorem 6 is based upon the following lemma:

Lemma 11 *Any planar upward polyline grid drawing of the complete binary tree with N nodes has $\Omega(\log N)$ width and $\Omega(\log N)$ height.*

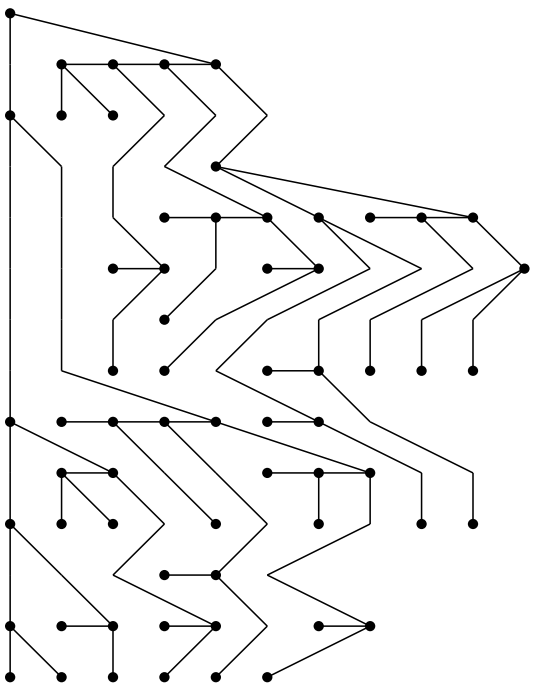

Figure 3.4: Drawing of the complete binary tree with 63 nodes produced by the implementation of the algorithm of Corollary 2 with $\alpha = 1/2$.

Proof: Let us denote by $H(N)$ and $W(N)$, the minimum height and width, respectively, of an upward polyline grid drawing of an N node complete binary tree T.

In any upward polyline grid drawing of T, a node, its children and its grand-children can not all be placed at the same height. Hence we get the recurrence $H(N) \geq H((N-3)/4) + 1$; $H(1) = H(3) = 0$. Therefore $H(N) = \Omega(\log N)$.

The width of any upward polyline grid drawing of T is at least one plus the minimum of the widths of the drawings of the subtrees of T in the drawing. Therefore $W(N) \geq W((N-1)/2) + 1$ with $W(1) = 0$. Hence $W(N) = \Omega(\log N)$. □

Theorem 6 *The N-node ordered binary tree B_N requires $\Omega(N \log N)$ area in any planar upward polyline grid drawing that preserves the order of the children.*

Proof: (For Theorem 6) Let B_N be an ordered binary tree comprising (see Fig. 3.5(a)):

- a chain with $N/3$ nodes, alternating between left and right children;
- $N/3$ leaves attached to each node of the chain, alternating as left and right children; and
- a complete subtree with $N/3$ nodes attached to the bottommost node of the chain.

In any planar upward polyline grid drawing of B_N, because of the order of the children, each pair of consecutive edges of the chain contributes at least one unit to

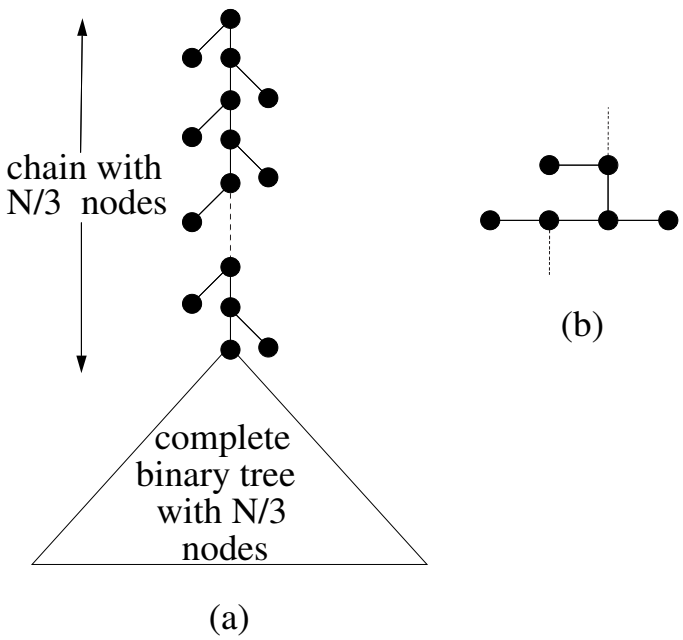

Figure 3.5: Order-preserving drawings: **(a)** Tree for the lower bound of Theorem 6. **(b)** Each pair of consecutive edges of the chain increases the height by at least one unit.

the height of the drawing (see Fig. 3.5(b)) so that the chain requires $\Omega(N)$ height. By Lemma 11, the complete subtree requires $\Omega(\log N)$ width. □

The lower bound of Theorem 6 is tight and can be achieved with the following simple recursive algorithm:

1. Let $T_1, T_2, \ldots, T_m$ be the subtrees of a bounded degree tree T whose root is v (see Fig. 3.6(a), where m is five). Let the root of the subtree T_i be denoted by v_i. Recursively construct the drawings of each T_i. Vertically stack their drawings (see Fig. 3.6(b)) such that the subtree at the bottom has the maximum size among the subtrees (other subtrees can be placed in any order, e.g. in Fig. 3.6(b), they are in the order T_5, T_3, T_2, T_1 from top to bottom). Place the root v above the drawing of the topmost subtree.

2. Now for every T_i, draw edge (v_i, v) as a polyline that uses one vertical track (grid column) either on the left or on the right of each T_j drawn above T_i depending upon whether T_j is to the left or right of T_i in T, and correspondingly switches from left to right or vice-versa (e.g., in Fig. 3.6, T_4 is to the left of T_5 and is to the right of the subtrees T_1, T_2 and T_3. Hence in Fig. 3.6(b), edge (v_4, v) first uses a vertical track on the left of the drawing of T_5 and them switches to a vertical track on the right of the drawing of T_1, T_2 and T_3). This completes the construction of the drawing of the tree T.

Fig. 3.6(e) shows the drawing constructed by this algorithm for the ordered tree of Fig. 3.6(d).

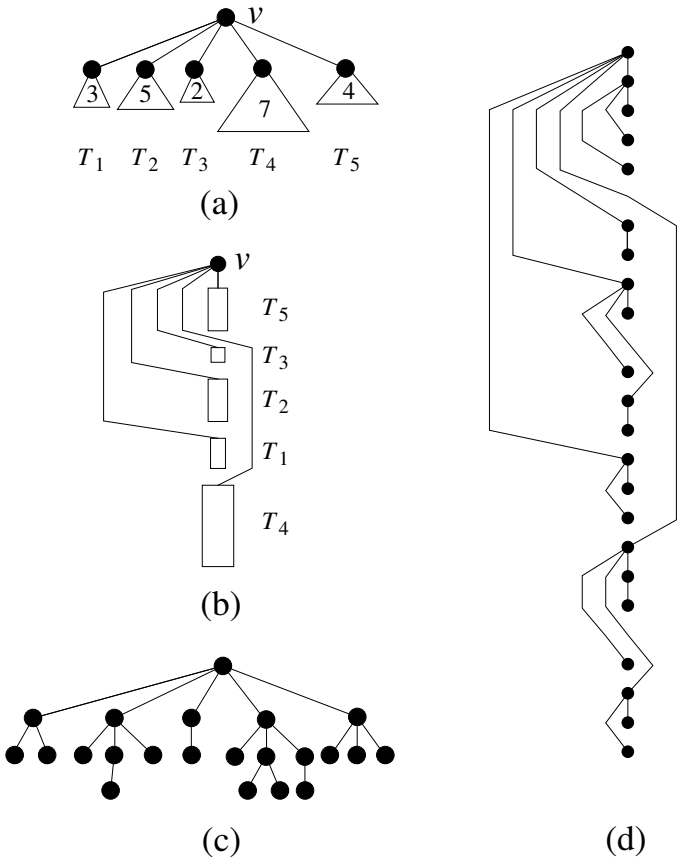

Figure 3.6: Example of construction of order-preserving drawings. Here, $m = 5$. **(a)** Tree T with subtrees $T_1, T_2, \ldots, T_5$. The number inside the triangle representing subtree T_i indicates the number of nodes of T_i. **(b)** Drawing of T, where the rectangles represent the drawings of the T_i's. **(c)** An ordered tree T'. **(d)** Drawing of T' constructed by the algorithm of Theorem 7.

Theorem 7 *Given a bounded-degree ordered tree T with N nodes, a planar polyline upward grid drawing of T with width $O(\log N)$, height $N-1$ and area $O(N \log N)$ that preserves the order of the children can be constructed in $O(N)$ time.*

Proof: Let $T_1, T_2, \ldots, T_m$ be the subtrees of T and v be the root of T. Let the root of the subtree T_i be denoted by v_i.

Since m is a constant, only a constant number (say c) of vertical tracks are needed to route all the edges of the type (v_i, v). Let T_k be the subtree that is drawn bottommost and hence has the maximum size among the subtrees. Therefore if we denote by $W(T)$ the width of the drawing of T, we have:

$$W(T) \leq \max\{W(T_k), \max_{i \neq k}\{W(T_i)\} + c\} \tag{3.1}$$

Let $width(N)$ be the maximum width of the drawing, constructed by the algorithm, of any tree with N nodes. Now we show inductively that $width(N) \leq \alpha \log_2 N$, for some

constant $\alpha \geq c$. This is trivially true for $N = 1$ since $width(1) = 0$.

Now suppose this is true for any tree with less than N nodes. If N is the size of a tree T, then the size of T_k (as defined above) is at most $N - 1$ and since there can be at most two subtrees of sizes both at least $(N-1)/2$, size of any subtree T_i of T, for $i \neq k$ is at most $(N-1)/2$. Therefore from equation 3.1 and our inductive hypothesis, $width(N) \leq \max\{\alpha \log_2(N-1), \alpha \log_2((N-1)/2) + c\}$. For $\alpha \geq c$, we have $width(N) \leq \alpha \log_2 N$.

It is easy to prove by a simple inductive argument that the height of the drawing of T is $N - 1$. Thus the area of the drawing is $O(N \log N)$.

The algorithm can be trivially implemented in linear time. □

3.3 Orthogonal Drawings

In this section we consider planar orthogonal upward grid drawings of binary trees, and provide a tight $\Theta(N \log\log N)$ bound on the area.

3.3.1 Drawing Algorithm

First, we present a simple straight-line drawing algorithm that will be later used as a subroutine. The algorithm is a variation of the ones by Shiloach [95] and by Crescenzi, Di Battista, and Piperno [21]. We say that a node in a drawing is *obstructed* if the vertical line through v intersects the drawing below v.

Lemma 12 *Let T be a binary tree with N nodes. A planar straight-line orthogonal upward grid drawing of T with width at most N and height at most $\log N$, such that every node of degree 0 or 1 is not obstructed, can be constructed in $O(N)$ time.*

Proof: The algorithm first transforms tree T into a left-heavy binary tree T'. Then for every node v, it sets $y(v)$ equal to the minus of the number of nodes on the path in T' from v to the root that are right children. Then, traversing T' in the post order sequence, for every node v, it sets $x(v)$ equal to the x-coordinate of its right child, if it has one otherwise sets $x(v)$ to one plus the x-coordinate of the last node visited before it. Finally it draws each edge as a straight-line between its endpoints. The drawing is clearly a straight-line upward drawing.

Two nodes v and w are assigned same x-coordinates by the algorithm if and only if there is a sequence of nodes $u_0 = v, u_1, u_2, \ldots, u_m = w$ or $u_0 = w, u_1, u_2, \ldots, u_m = v$ such that u_i is the right child of u_{i-1} in T'. Since no node of T with degree 0 or 1 has a right child in T', each node of T with degree 0 or 1 is assigned a unique x-coordinate and hence is not obstructed.

For every edge $e = (u, v)$ of T, either $y(u) - y(v) = 0$ which happens when u is the left child of v in T', or $x(u) - x(v) = 0$ when u is the right child of v in T'. Hence the drawing constructed is orthogonal.

Now we show that the drawing is planar. Suppose for contradiction, there are edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ that cross. Assume without loss of generality that v_1 precedes u_2 in the post order traversal of T'. If e_1 does not belong to a subtree rooted at u_2 then because of the post order assignment of x-coordinates, we have that $x(u_1) \leq x(v_1) < x(u_2) \leq x(v_2)$. Hence e_1 and e_2 can not cross, a contradiction. If e_1 belongs to a subtree rooted at the left child of u_2, then $x(u_1) \leq x(v_1) < x(u_2) \leq x(v_2)$ and if e_1 belongs to a subtree rooted at the right child of u_2, then $y(u_1) \leq y(v_1) < y(u_2) \leq y(v_2)$, and in either case e_1 and e_2 can not cross, again a contradiction. Hence the drawing constructed is planar.

The drawing has width at most N, and has height at most $\log N$ because any path in T' from a leaf to the root consists of at most $\log N$ nodes that are right children of their respective parents.

The algorithm can be easily implemented in linear time. □

An example of a drawing constructed by the algorithm of Lemma 12 is shown in Fig. 3.7(a).

Now, we recall some definitions on separators of binary trees. Let T be a binary tree with N nodes. A *partial tree* of T is a tree which is a subgraph of T. (Note the difference between partial tree and subtree: the subtree of T rooted at node v is the partial tree of T containing all the descendants of v.) A *separator* of a binary tree T is an edge of T whose removal divides T into two partial trees, each with at least $N/3$ nodes and at most $2N/3$ nodes (e.g., see Chazelle [17]). A recursive decomposition of T by separators defines a binary tree S, called *separator tree*, where each leaf of S corresponds to a node of T, and each internal node μ of S corresponds to a partial tree T_μ of T and to the separator $s_\mu = (u, v)$ of T_μ, with the left child of μ being associated to the partial tree of T_μ rooted at u, and the right child associated with the rest of T_μ. Tree S has $2N - 1$ nodes, height at most $\log_{3/2} N$, and can be constructed in $O(N)$ time (e.g., see Guibas *et al.* [60]).

The algorithm for constructing a planar orthogonal upward grid drawing of an N-node binary tree T is outlined below (see Fig. 3.7):

1. Construct the separator tree S of T.

2. Remove from S the nodes associated with partial trees with less than $\log N$ nodes, and let S' be the resulting truncated separator tree, which has $O(N/\log N)$ nodes and $O(\log(N/\log N)) = O(\log N)$ height. (See Fig. 3.7(b–c).)

3. For each leaf μ of S', construct a drawing of the associated partial tree T_μ, called a *block*, using the algorithm of Lemma 12 (See Fig. 3.7(a)). Since T_μ has $\Theta(\log N)$ nodes, its drawing has $O(\log N)$ width and $O(\log \log N)$ height.

4. Place the drawings of the blocks vertically one above the other, sorted from bottom to top according to the inorder sequence of the associated nodes of S'. (See Fig. 3.7(d).)

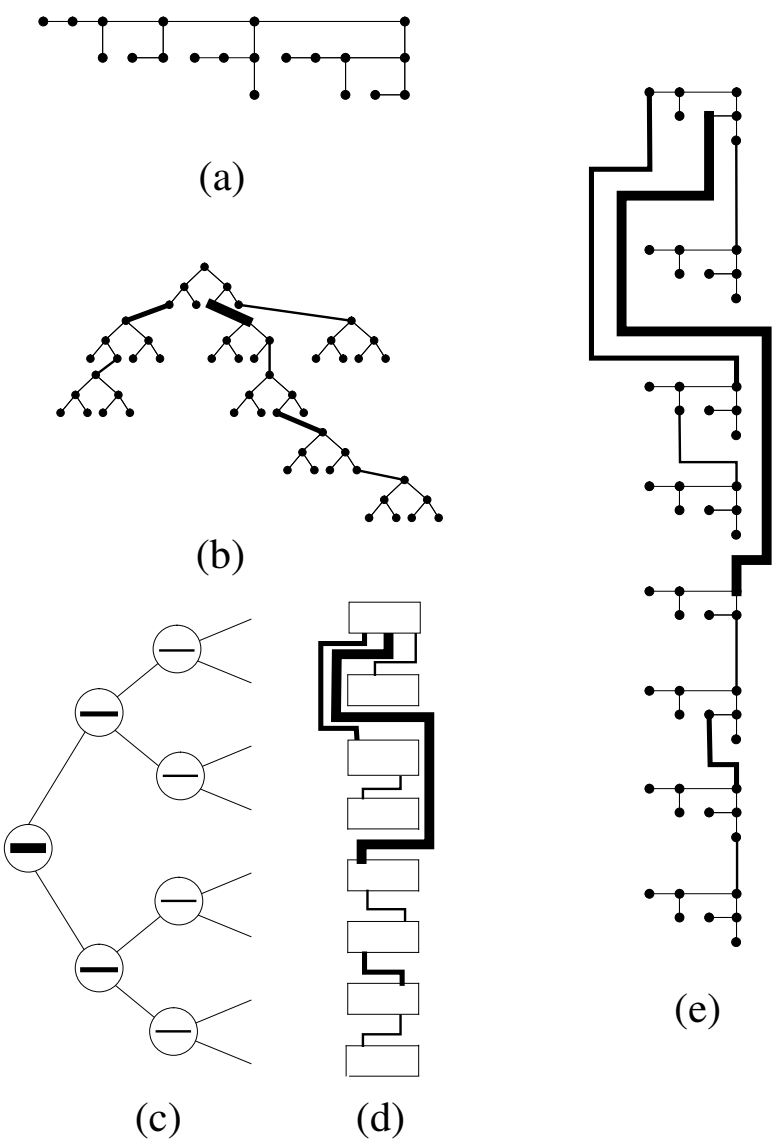

Figure 3.7: Illustration of the algorithm of Theorem 8: **(a)** Example of a drawing produced by the algorithm of Lemma 12. **(b)** A binary tree T and the separators that join blocks, drawn with thick lines. Here, $N = 55$, $\log_2 N = 5$, one block has size 6, and the remaining blocks have size 7. The block size B is obtained for T by using the constraint $\log_2 N \leq B \leq 2 \log_2 N$. **(c)** Truncated separator tree S' of the tree T of part (b). The nodes of S' correspond to the separators of T. **(d)** Stacking of the drawings of the blocks of T and routing of the separators. The rectangles represent the drawings of the blocks. **(e)** Drawing of T constructed by the algorithm of Theorem 8.

5. For each internal node ν of S', route separator $s_\nu = (u, v)$, on the current drawing, creating bends and adding extra tracks (grid rows or columns), whenever needed. (See Fig. 3.7(d).)

Fig. 3.7(e) shows a drawing constructed by this algorithm of the binary tree shown in Fig. 3.7(b).

Theorem 8 *Given a binary tree T with N nodes, a planar orthogonal upward grid drawing of T with $O(N)$ bends, $O(N \log\log N)$ area, $O(\log N)$ width, and $O(N \log\log N / \log N)$ height can be constructed in $O(N)$ time.*

Proof: By Lemma 12, the union of the drawings of the blocks constructed in Step 4 has width $O(\log N)$ and height $O(N \log\log N / \log N)$. We show that all the separators can be routed in Step 5 by adding a total of $O(\log N)$ vertical tracks and $O(N/\log N)$ horizontal tracks. We say that a separator *spans* a block T_μ if its endpoints are one below and the other above the drawing of T_μ. A separator s_ν is routed using one vertical track either on the left or on the right side of the drawing of a block T_μ spanned by s_ν, depending on whether T_μ is to the left or the right of the path from s_ν to the root in modified T (modified because of the conversion of each T_μ into a left-heavy tree in lemma 12).

A *switch* occurs when s_ν changes side between two blocks or enters a block. Each switch needs a distinct horizontal track and two bends. For each basic block T_μ, only the separators associated with ancestors of μ in S can span T_λ, so that $O(\log N)$ extra vertical tracks are sufficient to route all the separators in Step 5. The number of horizontal tracks added in Step 5 is equal to the total number of switches. The number of switches in the routing of separator s_ν is bounded by the height of the subtree rooted at ν in S'. If ν corresponds to a partial subtree of size k, the height of the subtree rooted at ν in S' is $O(\log(k/\log N))$. Thus the total number of switches $s(N)$ is the solution of the recurrence $s(k) = s(k/c) + s(k(1-1/c)) + O(\log(k/\log N)); s(\log N) = 0$, where c lies between 2 and 3. It is easy to see that $s(k)$ is less than or equal to $c_1 k/\log N - c_2 \log(k/\log N) - c_3$ for appropriate constants c_1, c_2 and c_3. Therefore the total number of switches is $O(N/\log N)$. □

The $\log\log N$ factor in the area achieved by the algorithm of Theorem 8 may at first seem unnatural, but it is not, as we show in the next section that an $N \log\log N$ area-bound is tight within a constant factor.

3.3.2 A Superlinear Lower Bound on the Area

We show that the superlinear area-bound of Theorem 8 is tight within a constant factor. Let T_N be the N-node binary tree consisting of (see Fig. 3.8):

- a chain C with $N/3$ nodes, where every $\sqrt{\log N}$-th node of C is called a *joint* of C;
- $N/3\sqrt{\log N}$ complete subtrees, where each subtree has $\sqrt{\log N}$ nodes and is rooted at a child of a joint of C;

- a complete subtree with $N/3$ nodes, which is rooted at a child of the first node of C.

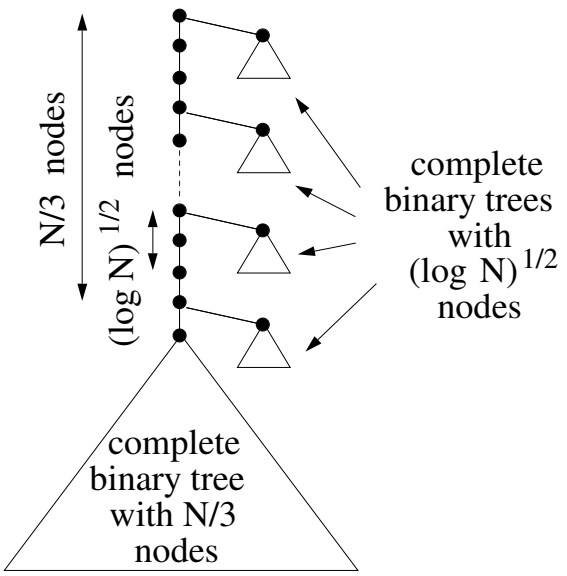

Figure 3.8: Tree for the lower bound of Theorem 9.

Theorem 9 *Any planar upward orthogonal grid drawing of the N-node tree T_N requires area $\Omega(N \log\log N)$.*

Proof: Consider any planar upward orthogonal grid drawing of T_N, and let W and H be the width and height of the drawing, respectively. If W is more than $N/6$, then the area of the drawing is $\Omega(N \log N)$ since, by Lemma 11, $H = \Omega(\log N)$. Now, suppose W is at most $N/6$. Since T_N contains a complete subtree with $N/3$ nodes, by Lemma 11 we have that W is $\Omega(\log N)$. Consider the subdrawing of any subchain S of C with $2W$ nodes. We claim that the height of the drawing of S is $\Omega(\log\log N)$. Since the drawing is upward, the drawings of any two consecutive subchains of C must be vertically stacked. Hence, the claim implies the statement of the theorem.

The proof of the claim is illustrated in Fig. 3.9. For the sake of contradiction, we need only to consider the case when the height of the drawing of S is less than $\log\log N$. Let ℓ' be the horizontal line through the bottommost node of S in the drawing. Since S has $2W$ nodes the drawing of S has width at most W and contains at least W obstructed nodes (recall the definition of "obstructed" from Section 3.3.1). Also, by a simple pigeon-hole argument, there are at least $W/\log\log N$ obstructed nodes of S on the same horizontal line ℓ'', where ℓ'' is above ℓ'. Thus, there are at least $W/(\sqrt{\log N}\log\log N)$ obstructed joints along line ℓ''. Consider the subtrees with $\sqrt{\log N}$ nodes connected to such obstructed joints. Since the drawing is upward, these subtrees are drawn below line ℓ''. If any such subtree is drawn entirely above ℓ', then by Lemma 11 the height of the drawing of S is $\Omega(\log\log N)$, and the claim is verified. Otherwise, every subtree has a leaf v below or on line ℓ', and we consider the path from v to its closest ancestor joint u. Let x' and

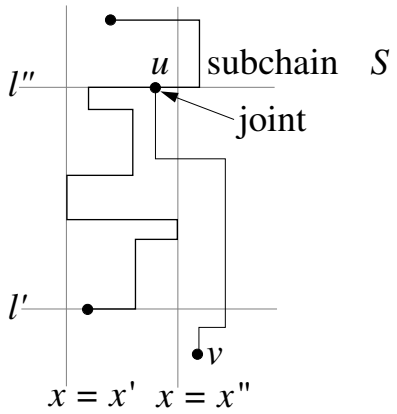

Figure 3.9: Drawing of subchain S in the proof of Theorem 8.

x'' be the minimum and maximum x-coordinates of the drawing of S below line ℓ''. The path between u and v must intersect one of the vertical lines at $x = x' - 1$ or $x = x'' + 1$. Also, since we have a planar upward orthogonal grid drawing, such intersections must occur at distinct grid points of these two vertical lines, and between ℓ' and ℓ''. Since there are at least $W/(\sqrt{\log N} \log\log N)$ such paths, we have that the height of the drawing of S is at least $W/(2\sqrt{\log N} \log\log N)$. Recalling that $W = \Omega(\log N)$, we conclude that the height of the drawing of S is $\Omega(\sqrt{\log N}/(2\log\log N)) = \Omega(\log\log N)$, which contradicts our height assumption about S. This completes the proof of the claim. □

The drawings constructed by the algorithm of Theorem 8 do not preserve the left-to-right order of the children. Note that the lower bound of Theorem 6 applies also to orthogonal drawings.

3.4 Experimental Results

We have implemented the algorithm of Corollary 2 given in section 3.2 for constructing planar upward polyline drawings of trees, on a Sun Sparcstation 10 in language "C". Our implementation takes a binary tree as its input. The implementation places $\lceil 2^{\alpha k} \rceil$ nodes in a block in step 2c of the algorithm. The theoretical upper bound for width and height of the drawing with $\alpha = 1/2$, of a N-node binary tree, produced by our implementation, computed as in the proof of theorem 4 is

$$\begin{aligned} Width &= \lceil (3+\sqrt{2})\sqrt{N} + \log_2 N \rceil \\ Height &= \lceil (3+\sqrt{2})\sqrt{N} + \log_2 N \rceil \end{aligned}$$

The additive term of $\log_2 N$ appears in width because the implementation places $\lceil 2^{\alpha k} \rceil$ nodes in a block in the step 2c as compared to $2^{\alpha k}$ nodes as described in the algorithm. The ratio of theoretical area bound of drawing and number of nodes in input tree therefore lies between 19.48 and 33.34.

The experimental results obtained for complete binary trees and Fibonacci trees, with $\alpha = 1/2$ are presented in Table 3.2. In the table, we have denoted the theoretical

area bound by A_{th} and the experimental area by A_{ex}. Fig. 3.10 and Fig. 3.11 give a comparison of the the ratios $r_{th} = A_{th}/N$ and $r_{ex} = A_{ex}/N$ for complete binary and Fibonacci trees respectively. The value of r_{ex} for both complete binary and Fibonacci trees is less than 5 even up to about 24 million nodes and hence the algorithm is quite area-efficient in practice.

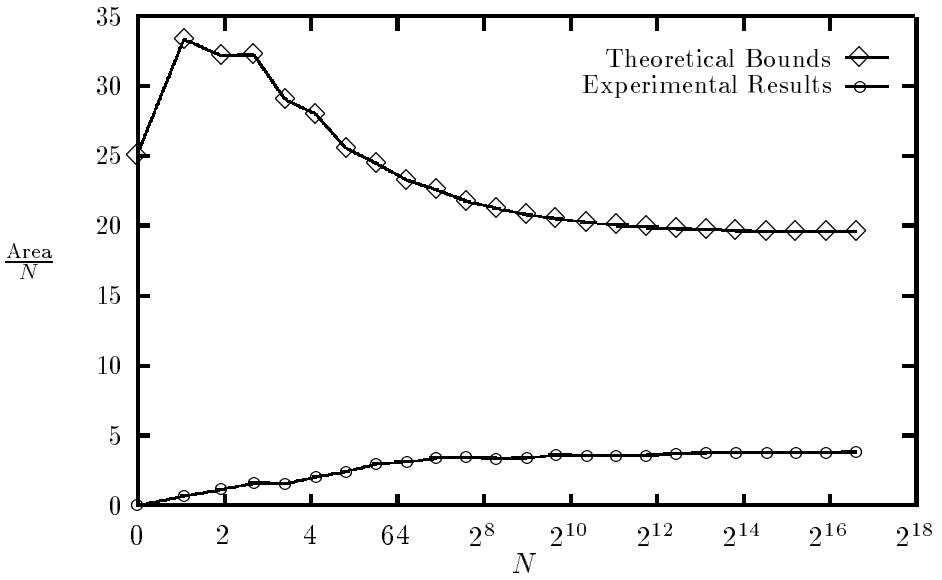

Figure 3.10: Experimental results on the area of the drawings of complete binary trees produced by the algorithm of Corollary 2 with $\alpha = 1/2$, and comparison with the theoretical upper bounds

3.5 Conclusion

In this chapter, we have investigated the area requirement of different types of planar upward drawings of a rooted bounded-degree tree T with N nodes. Our results are summarized as follows: T admits a planar upward polyline grid drawing with area $O(N)$, and with width $O(N^{\alpha})$ for any prespecified constant α such that $0 < \alpha < 1$. If T is a binary tree, then T admits a planar orthogonal upward grid drawing with area $O(N \log\log N)$. If T is ordered, then T admits an $O(N \log N)$-area planar upward grid drawing that preserves the left-to-right ordering of the children of each node. All of the above area bounds are asymptotically optimal in the worst case. Also, there are $O(N)$-time algorithms for constructing each of the above types of drawings of T with asymptotically optimal area.

The drawings obtained with the techniques presented in this chapter indicate that perhaps there is a tradeoff between the aesthetic quality and area bound achieved by tree drawing algorithms. This issue needs to be further explored. A possible direction is to

	# Nodes N	Theoretical Bounds			Experimental Results			
		Width = Height	Area A_{th}	$r_{th} = \frac{A_{th}}{N}$	Width	Height	Area A_{ex}	$r_{ex} = \frac{A_{ex}}{N}$
Complete	15	22	484	32.27	5	6	30	2.00
Binary	31	30	900	29.03	8	7	56	1.81
Tree	63	42	1764	28.00	11	13	143	2.27
	127	57	3249	25.58	18	18	324	2.55
	255	79	6241	24.47	27	29	783	3.07
	511	109	11881	23.25	40	41	1640	3.21
	1023	152	23104	22.58	62	57	3534	3.45
	2047	211	44521	21.75	90	79	7110	3.47
	4095	295	87025	21.25	121	114	13794	3.37
	8191	413	170569	20.82	178	158	28124	3.43
	16383	580	336400	20.53	261	228	59508	3.63
	32767	815	664225	20.27	374	314	117436	3.58
	65535	1147	1315609	20.07	521	451	234971	3.59
	131071	1616	2611456	19.92	746	632	471472	3.60
	262143	2279	5193841	19.81	1071	905	969255	3.70
	524287	3216	10342656	19.73	1542	1284	1979928	3.78
	1048575	4541	20620681	19.67	2182	1802	3931964	3.75
	2097151	6414	41139396	19.62	3074	2553	7847922	3.74
	4194303	9063	82137969	19.58	4344	3616	15707904	3.75
	8388607	12808	164044864	19.56	6192	5108	31628736	3.77
	16777215	18105	327791025	19.54	8902	7229	64352558	3.84
Fibonacci	20	25	625	31.25	6	7	42	2.10
Tree	33	31	961	29.12	10	9	90	2.73
	54	39	1521	28.17	10	11	110	2.04
	88	48	2304	26.18	15	14	210	2.39
	143	60	3600	25.17	24	19	456	3.19
	232	76	5776	24.90	26	23	598	2.58
	376	95	9025	24.00	42	30	1260	3.35
	609	119	14161	23.25	56	40	2240	3.68
	986	149	22201	22.52	64	48	3072	3.12
	1596	187	34969	21.91	86	62	5332	3.34
	2583	236	55696	21.56	130	77	10010	3.88
	4180	298	88804	21.24	194	95	18430	4.41
	6764	376	141376	20.90	193	122	23546	3.48
	10945	476	226576	20.70	276	156	43056	3.93
	17710	602	362404	20.46	384	194	74496	4.21
	28656	763	582169	20.32	402	252	101304	3.54
	46367	967	935089	20.17	577	320	184640	3.98
	75024	1226	1503076	20.03	825	404	333300	4.44
	121392	1555	2418025	19.92	831	521	432951	3.57
	196417	1974	3896676	19.84	1282	659	844838	4.30
	317810	2507	6285049	19.78	1705	832	1418560	4.46
	514228	3185	10144225	19.73	1717	1070	1837190	3.57
	1346268	5143	26450449	19.65	3463	1705	5904415	4.39
	2178308	6537	42732369	19.62	4824	2141	10328184	4.74
	3524577	8309	69039481	19.59	4890	2733	13364370	3.79
	5702886	10564	111598096	19.57	6968	3494	24346192	4.27
	9227464	13433	180445489	19.56	9530	4425	42170250	4.57
	14930351	17081	291760561	19.54	9980	5664	56526720	3.79
	24157816	21721	471801841	19.53	14633	7202	105386866	4.36

Table 3.2: Experimental results on the drawings of complete binary trees and Fibonacci trees produced by the implementation of the algorithm of Corollary 2 with $\alpha = 1/2$, and comparison with the theoretical upper bounds.

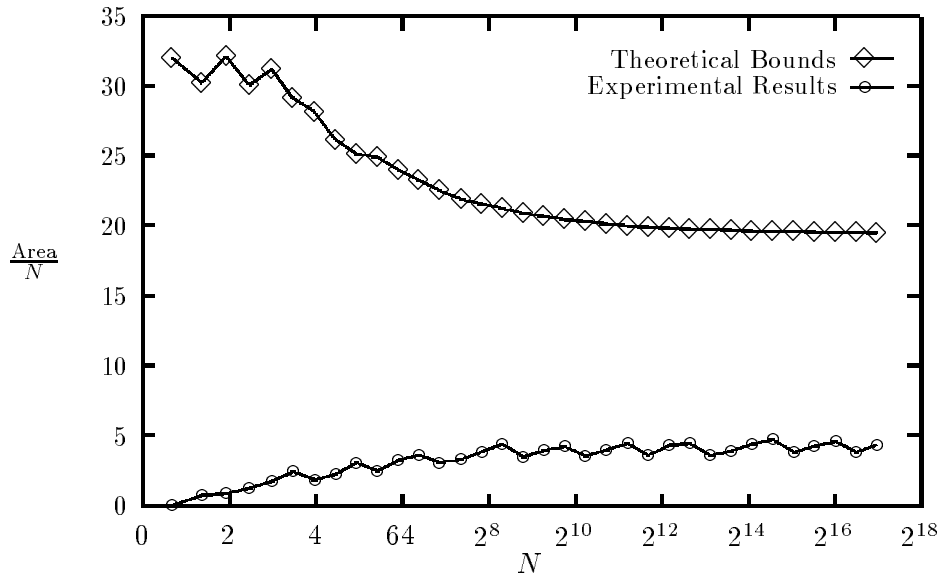

Figure 3.11: Experimental results on the area of the drawings of Fibonacci trees produced by the algorithm of Corollary 2 with $\alpha = 1/2$, and comparison with the theoretical upper bounds

investigate the tradeoff between the maximum number of bends per edge and the area-requirement of a drawing. For example, our algorithm for polyline drawings gives area $O(N)$ but allows $O(N)$ bends per edge, whereas our algorithm for orthogonal drawings gives area $O(N \log \log N)$ and $O(\log N)$ bends per edge.

In view of our results, the main open problem on this subject is determining the area requirement of planar upward *straight-line* drawings of rooted trees. There is still a gap between the trivial $\Omega(N)$ lower bound and the $O(N \log N)$ upper bound [21, 95]. It would also be interesting to determine the total edge-length of our $O(N)$ area polyline drawings and extend our results to unbounded-degree trees. A related open problem is to investigate the area requirement of planar upward straight-line drawings of rooted trees such that the *angular resolution* (i.e., the minimum angle between any two edges incident on the same node) is maximized. Previous results on the angular resolution of (non upward) drawings of graphs appear in [48, 82].

Chapter 4

Angular Resolution

4.1 Introduction

Coping with the finite resolution of display devices and of the human eye is a fundamental problem in graph drawing. Namely, in visualization applications, it is important to construct drawings of graphs that avoid placing vertices and edges too close. In this chapter we investigate the problem of constructing planar straight-line drawings with with large angles between the edges. Namely, we study the angular resolution of straight-line drawings, defined as the smallest angle formed by two incident edges. Besides visualization applications (see, e.g., [29]), constructing drawing with large angular resolution is important in the design of wireless communications networks (see, e.g., [2]).

The study of the angular resolution of drawings has attracted considerable interest in the last years. Formann, Hagerup, Haralambides, Kaufmann, Leighton, Simvonis, Welzl, and Woeginger [49] were the first to study the angular resolution of (generally *nonplanar*) straight-line drawings of various classes of graphs. In particular, they show that a degree-d graph has a straight-line drawing with angular resolution $\Omega(1/d^2)$, and every degree-d planar graph has a straight-line (*nonplanar*) drawing with angular resolution $\Omega(1/d)$. (Note that the above bounds are independent from the number of vertices of the graph.) They also pose the open problem of characterizing the angular resolution of *planar* straight-line drawings.

Malitz and Papakostas [83] make further progress on the above open problem by proving a lower bound on the angular resolution dependent only on the degree of the graph. Namely, they show that a degree-d planar graph admits a planar straight line drawing with angular resolution $\Omega(1/7^d)$, irrespectively of the number of vertices. They also analyze a linear program associated with two necessary conditions on the angles of a planar triangulation (i.e., the sum of the angles around a vertex is 2π, and the sum of the angles inside each triangle is π), and find that such a linear program admits a solution with $\Omega(1/d)$ angles. This leads them to conjecture that the trivial $\Omega(1/d)$ lower bound on the angular resolution may be achievable for planar straight-line drawings.

Kant [68, 69] shows that testing whether a biconnected planar graph admits a planar straight-line drawing with angular resolution greater than or equal to a given constant

is NP-hard, thus providing further motivation to the study of asymptotic bounds. He also considers polyline drawings (where edges are drawn as polygonal chains), and shows that a degree-d triconnected planar graph admits a planar polyline grid drawing with angular resolution $\Omega(1/d)$.

Di Battista and Vismara [38] provide a characterization of the angles in planar straight-line drawings of maximal planar graphs through a nonlinear system of inequalities. As a consequence, they show that testing whether a prescribed assignment of angles to a maximal planar graph is achievable in a planar straight-line drawing can be done in linear time, and that constructing a planar straight-line drawing with maximum angular resolution can be reduced to the solution of a nonlinear optimization problem.

Our results can be summarized as follows:

In Section 4.3, we prove the first nontrivial upper bound on the angular resolution of planar straight-line drawings by exhibiting a family of degree-d planar graphs that require angular resolution $O(\sqrt{\log d/d^3})$ in any planar straight-line drawing. Previously, only the trivial $O(1/d)$ bound was known, and determining whether it is achievable was posed as an open problem in [83, 69].

In Section 4.4, we show a continuous trade-off between the area and the angular resolution of planar straight-line drawings. Namely, there exists a constant $c > 1$ such that, for any $n > d > 6$, with $n-3$ a multiple of $d-4$, there exists a planar graph G_n^d with n vertices and degree d such that any planar straight-line drawing of G_n^d with angular resolution ρ has area $\Omega(c^{\rho n})$. In particular, this implies that there exist bounded-degree planar graphs for which asymptotically optimal resolution can be achieved only at the expense of an exponential increase in the area. Previously, area/trade-off results were proved for planar drawings (see, e.g., [6, 16, 21, 37, 26, 53, 68, 69, 70, 71, 75, 93, 94, 101, 109]), depending on various restrictions on the representation (e.g., upward, straight-line, orthogonal, polyline). Our result is the first "continuous" area/trade-off result for planar drawings.

In Section 4.5, we give linear-time algorithms for constructing planar straight-line drawings with large resolution of various classes of graphs, such as series-parallel graphs, maximal outerplanar graphs, and nested-star graphs (i.e., maximal planar graphs generated from a triangle by repeated insertions of stars). The angular resolution is shown to be $\Omega(1/d^2)$ for series-parallel and nested-star graphs, and $\pi/(d-1)$ for maximal outerplanar graphs. Our algorithms for series-parallel and nested-star graphs exploit the recursive structure of these graphs. In each recursive step of these algorithms, a triangle Δ is allocated to a subgraph H (of the input graph), and H is drawn completely inside Δ. Δ has the property that the interior angle (of Δ) at an apex verticex a of Δ either has a magnitude independent of both the number of vertices in H and the degree of H, or has a magnitude proportional to the degree of the vertex of H that is placed on a, giving us the desired lower bound on the smallest angle. Our algorithm for drawing a maximal outerplanar graph first performs a breadth-first search of the graph, and then places vertices on concentric circles such that vertices visited in the same step of the search are placed on the same circle. Vertices on the same circle are distributed such that the angle between any two edges incident on the same vertex is at least $\pi/(d-1)$. Previously, Malitz and Papakostas [83] showed that angular resolution $\pi/2(d-1)$ can be achieved for

outerplanar graphs. No polynomial bounds in $1/d$ were known for series-parallel graphs and nested-star graphs. We leave it as an open problem either proving or disproving that $\Omega(1/d^2)$ is a tight lower bound on the angular resolution of the series-parallel and nested-star graphs. Note that the family of graphs requiring angular resolution $O(\sqrt{\log d}/d^3)$ in any planar straight-line drawing are actually nested-star graphs.

The significance of our result on nested-star graphs is motivated by the following observation: Malitz and Papakostas [83] show that a disk packing of a planar degree-d graph yields a planar straight-line drawing with angular resolution $\Omega(1/7^d)$, and they exhibit a family of nested-star graphs to prove that this exponential bound is the best that can be obtained with the disk packing method. Our results imply that the disk packing method yields suboptimal results and leave as an open problem whether the angular resolution of planar straight-line drawings can be bounded from below by a polynomial in $1/d$.

Our algorithms for drawing nested-star and series-parallel graphs exploit various geometric properties of the angles in certain triangulations. Since most existing algorithms for drawing planar graphs rely on graph-theoretic properties (e.g., orientation, coloring, flow), the techniques in this chapter appear to be interesting in the way they blend geometric and graph-theoretic methods. We believe that our results will stimulate further research in this direction. Most of the results and techniques presented in this chapter can also be found in [55, 58].

The rest of this chapter is organized as follows. Section 4.2 provides basic definitions. The upper bound on the angular resolution is shown in Section 4.3. Section 4.4 contains the tradeoff between area and angular resolution. The algorithms for constructing straight-line drawings with large angular resolution, of nested-star series-parallel, and maximal outerplanar graphs are described in Section 4.5. Our Conclusions are given in Section 4.6.

4.2 Definitions

First, we review basic graph drawing terminology [29]. Various graphic standards have been proposed for the representation of graphs in the plane. Usually, vertices are represented by points, and each edge (u, v) is represented by a simple open Jordan curve joining the points associated with the vertices u and v. A *straight-line* drawing maps each edge into a straight-line segment. The *angular resolution* of a straight-line drawing is the smallest angle formed by two edges incident on the same vertex. A drawing is *planar* if no two edges intersect. The *area* of a drawing is the area of the smallest convex polygon covering the drawing. Whenever we give lower bounds on the area, we assume that the drawing is constrained by some rule that prevents it from being arbitrarily scaled down (e.g., integer coordinates for the vertices, or a minimum unit distance between any two vertices). The results of Section 4.4 are fully general, since they hold under any rule that implies a finite minimum area for the drawing of a graph.

Other notation and concepts used in the chapter are as follows. Two vertices u and v are *neighbors* if there is an edge (u, v) in the graph. A vertex w is a *common* neighbor

of vertices u and v if there are edges (u, w) and (v, w) in the graph. The *degree* of a vertex is the number of edges incident on it. The *degree* of a graph is the maximum degree of any vertex in the graph. The notation $d_G(u)$ indicates the degree of vertex u in graph (or subgraph) G. Given a line segment with endpoints A and B, AB denotes either the line segment itself or its length. Given two line segments AB and BC, $\angle ABC$ denotes either the angle between AB and BC or its measure. Finally, ΔABC denotes the triangle with vertices A, B and C. The *internal angles* of a triangle ΔABC are the angles inside it at its vertices. Therefore, ΔABC has three internal angles, namely $\angle ABC$, $\angle BCA$, and $\angle CAB$.

In each planar drawing of a planar graph, the edges of a vertex are incident on it in a cyclic order. An *embedding* of a planar graph is the collection of circular permutations of the edges incident on each vertex in a planar drawing of the graph. An *embedded* planar graph is a graph equipped with an embedding. A *cycle* $c = w_0 w_1 w_2 w_3 \ldots w_m w_0)$, where $m \geq 2$, consists of the edges $(w_0, w_1), (w_1, w_2), (w_2, w_3), \ldots, (w_m, w_0)$. A *path* $p = uw_1 w_2 w_2 \ldots v$, where for each i, $w_i \neq u, v$, consists of edges $(u, w_1), (w_1, w_2), (w_2, w_3), \ldots,$ (w_m, v). We also use the shorthand notation $p : u \rightsquigarrow v$, to denote a path p from u to v. All planar drawings of an embedded graph are topologically equivalent. Let G be an embedded planar graph. Each planar drawing of G divides the plane into several regions, which are separated from each other by cycles consisting of the edges and vertices of G; These cycles are called the *faces* of G. Notice that exactly one region is unbounded; The cycle separating it from the other regions is called the *external face* of G. The other faces of G are called its *internal faces*. A vertex v is in the *interior* of a cycle c of G if in any planar drawing of G (and hence in all the planar drawings of G), it is placed strictly inside the region enclosed by the drawing of c. A face f of G is *enclosed* by a cycle c of G if in any planar drawing of G (and hence in all the planar drawings of G), no vertex and edge of f is drawn outside the region enclosed by the drawing of c. A path p is *on* a face f if all its edges are on Face f.

We assume, unless explicitly stated, that a graph does not have any multiple edges, i.e., between any two vertices, there can be at most one edge.

4.3 Upper Bound on the Angular Resolution

In this section we show that there exists an infinite family of degree-d graphs that require angular resolution $O(\sqrt{\log d/d^3})$ in any planar straight-line drawing. Thus, for these graphs the linear program of [83] is guaranteed to give an inconsistent drawing.

First, we prove the following geometric lemma (see Fig. 4.1).

Lemma 13 *Let D be a point inside a triangle ΔABC. Let α_1, α_2, β_1, β_2, γ_1 and γ_2 be the angles defined in Fig. 4.1. If $\alpha \leq \pi/2$ and $\alpha_2/\alpha_1 \geq 1$, then*

$$\min\left(\frac{\beta_2}{\beta_1}, \frac{\gamma_2}{\gamma_1}\right) \leq \frac{\pi^2}{4}\sqrt{\frac{\alpha_1}{\alpha_2}} \tag{4.1}$$

Proof:

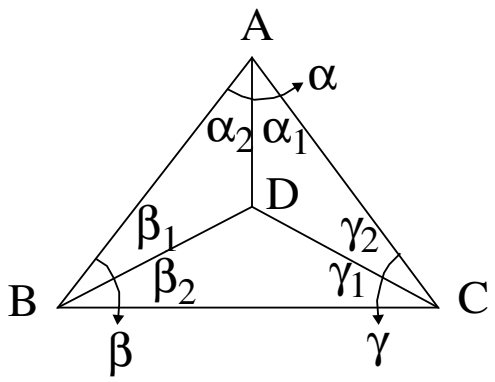

Figure 4.1: Dividing a triangle ΔABC by introducing a vertex D in its interior and joining D with A, B and C.

$$\frac{\sin\alpha_2}{\sin\alpha_1}\frac{\sin\beta_2}{\sin\beta_1}\frac{\sin\gamma_2}{\sin\gamma_1} = 1 \tag{4.2}$$

We shall use the following variants of Eq. 4.2:

$$(\sin\beta_2/\sin\beta_1)(\sin\gamma_2/\sin\gamma_1) = \sin\alpha_1/\sin\alpha_2 \tag{4.3}$$
$$(\sin\beta_1/\sin\beta_2)(\sin\gamma_1/\sin\gamma_2) = \sin\alpha_2/\sin\alpha_1 \tag{4.4}$$

We know that at least two angles of a triangle have value not exceeding $\pi/2$. α is one of them (see the lemma statement). Let us assume without loss of generality that γ be the other angle not exceeding $\pi/2$. Since $\alpha \leq \pi/2$, both α_1 and α_2 are at most $\pi/2$. There are two cases:

$\beta_2 \geq \beta_1$: Therefore $\beta_1 < \pi/2$. Since $\beta_1 = \beta - \beta_2 < \pi - \beta_2$, $\sin\beta_2 \geq \sin\beta_1$. Since $\sin\theta \geq (2/\pi)\theta$ for $\theta \leq \pi/2$ and $\theta \geq \sin\theta$, from Eq. 4.3 we get

$$\frac{\sin\beta_2}{\sin\beta_1}\frac{(2/\pi)\gamma_2}{\gamma_1} \leq \frac{\alpha_1}{(2/\pi)\alpha_2} \tag{4.5}$$

Since $\sin\beta_2 \geq \sin\beta_1$, $\gamma_2/\gamma_1 \leq (\pi^2/4)(\alpha_1/\alpha_2) \leq (\pi^2/4)\sqrt{\alpha_1/\alpha_2}$ because $\alpha_2/\alpha_1 \geq 1$. Hence, we obtain Eq. 4.1.

$\beta_2 < \beta_1$: Therefore $\beta_2 \leq \pi/2$ because otherwise $\beta_2 > \pi$. Again, since $\sin\theta \geq (2/\pi)\theta$ for $\theta \leq \pi/2$ and $\theta \geq \sin\theta$, from Eq. 4.4 we get

$$\frac{\beta_1}{(2/\pi)\beta_2}\frac{\gamma_1}{(2/\pi)\gamma_2} \geq \frac{(2/\pi)\alpha_2}{\alpha_1}$$

Therefore, we have $(\beta_2/\beta_1)(\gamma_2/\gamma_1) \leq \pi^3\alpha_1/(8\alpha_2)$, which implies Eq. 4.1.

□

We are now ready to present the main theorem of this section.

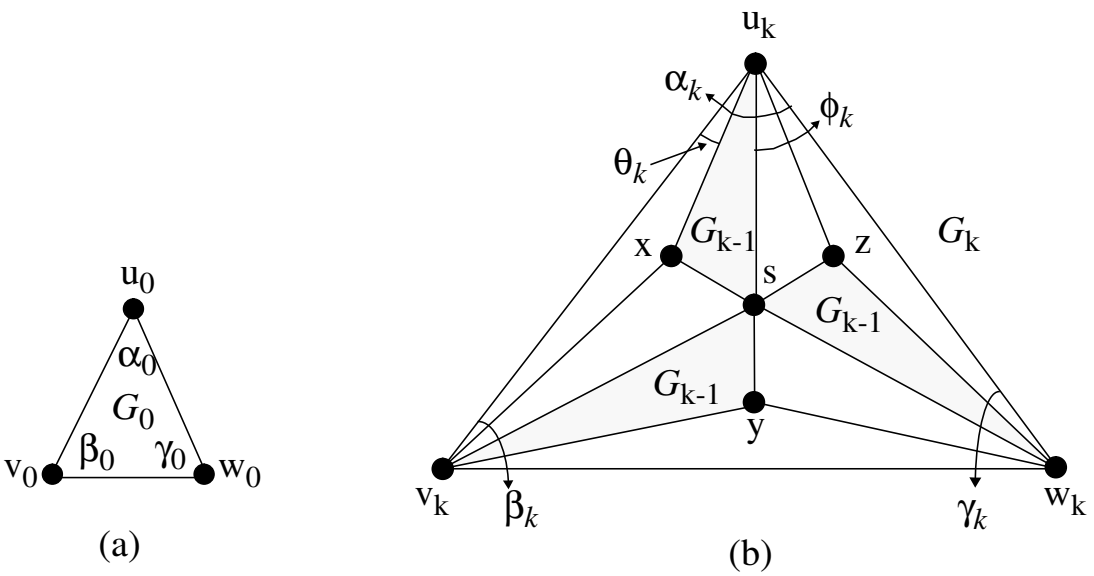

Figure 4.2: (*a*) Graph G_0; (*b*) Graph G_k;

Theorem 10 *There exists an infinite family of planar graphs with degree d and $\Theta(3^{d/9})$ vertices that require angular resolution $O(\sqrt{(\log d)/d^3})$ in any planar straight-line drawing.*

Proof: Consider graph G_k, which is recursively defined as follows: G_0 is a triangle with vertices u_0, v_0 and w_0 (see Fig. 4.2(a)).Graph G_k is constructed from three copies of G_{k-1}, as shown in (see Fig. 4.2(b)). The number of vertices $n(k)$ of G_k is given by the recurrence $n(k) = 3n(k-1) - 2; n(0) = 3$. Hence, $n(k) = \Theta(3^k)$. Let $d_k(a)$ denote the number of neighbors in G_k of a vertex a. $d_k(u_k) = d_k(v_k) = d_k(w_k) = d_{k-1}(w_{k-1}) + 3$. Since $d_k(w_k) = d_{k-1}(w_{k-1}) + 3; d_0(w_0) = 2$, we have $d_k(w_k) = 3k + 2$. $d_k(s) = d_{k-1}(u_{k-1}) + d_{k-1}(v_{k-1}) + d_{k-1}(w_{k-1}) = 3d_{k-1}(w_{k-1}) = 9k - 3$. Since s is the vertex in G_k with the highest degree, G_k has degree $9k - 3$.

Given a straight line drawing of G_k with $u_k v_k w_k$ on the external face, we define the following angles (see Fig. 4.2(*b*)): $\alpha_k = \angle w_k u_k v_k$, $\beta_k = \angle u_k v_k w_k$, $\gamma_k = \angle v_k w_k u_k$, $\phi_k = \angle w_k u_k s$ and $\theta_k = \angle v_k u_k x$.

We now claim that the angular resolution of any planar straight-line drawing of G_m with $\Delta u_m v_m w_m$ as the external face is $O(\sqrt{(\log m)/m^3})$. Assume without any loss of generality that $\alpha_m \leq \pi/2$ because in any triangle at least one internal angle has value not exceeding $\pi/2$. If α_0 is $O(\sqrt{(\log m)/m^3})$ the claim holds. So suppose $\alpha_0 = \Omega(1/m^{3/2})$. We shall find another angle in the drawing that is $O(\sqrt{(\log m)/m^3})$. Namely, for any $k \leq m$, let $r_k = \alpha_{k-1}/(\phi_k + \theta_k)$. Thus $\alpha_{k-1} = \alpha_k\, r_k/(r_k + 1)$. We have $\alpha_0 = \alpha_m \prod_1^k r_k/(r_k + 1)$. Since $\alpha_m < \pi$ and $\alpha_0 = \Omega(1/m^{3/2})$, we obtain

$$\prod_{k=1}^{m} r_k/(r_k + 1) = \Omega(1/m^{3/2}) \tag{4.6}$$

We now prove that there exists an $i \geq m/2$ such that $r_i = \Omega(m/\log m)$. For any k,

from Eq. 4.6 we get that $\prod_{j=k}^{m} r_j/(r_j+1) = \Omega(1/m^{3/2})$ because $r_q/(r_q+1) < 1$ for any q. If $r(k) = \max_{j=k}^{m}(r_j)$, we get $\{r(k)/(r(k)+1)\}^{m-k} = \Omega(1/m^{3/2})$. Consequently

$$\left(1+\frac{1}{r(k)}\right)^{m-k} = O(m^{3/2}) \tag{4.7}$$

It is easy to see from Eq. 4.7 that for sufficiently large m, $r(m/2) > 1$. From Eq. 4.7,

$$\begin{aligned}(m/2)(\log(1+1/r(m/2))) &= O(\log m)\\ m/r(m/2) &= O(\log m)\\ r(m/2) &= \Omega(m/\log m)\end{aligned}$$

Thus there is an index $i \geq m/2$ such that $r_i = \Omega(m/\log m)$. By Lemma 13, either $\beta_{i-1} \leq \beta_i\pi^2/4(\sqrt{r_i}+\pi^2/4)$ or $\gamma_{i-1} \leq \gamma_i\pi^2/4(\sqrt{r_i}+\pi^2/4)$. Since $d_{i-1}(v_{i-1}) = d_{i-1}(w_{i-1}) = 3i-1$, both β_{i-1} and γ_{i-1} include $3i-2 \geq 3m/2-2$ angles in their interior. Hence for sufficiently large m, there is an angle inside either β_{i-1} or γ_{i-1} that is $O(\sqrt{(\log m)/m^3})$. This proves our claim.

It can be readily seen that a planar straight-line drawing of G_m where $\Delta u_m v_m w_m$ is not the external face contains as a subdrawing a drawing of G_{m-1} with external face $\Delta u_{m-1} v_{m-1} w_{m-1}$. Hence, any planar straight-line drawing of G_m has angular resolution $O(\sqrt{(\log(m-1))/(m-1)^3}) = O(\sqrt{(\log m)/m^3})$. We conclude by recalling that graph G_m has degree $\Theta(m)$ and $\Theta(3^m)$ vertices.

□

4.4 Tradeoff between Area and Angular Resolution

In this section we show that there exists a continuous tradeoff between the area requirement and the angular resolution of a planar straight-line drawing. Namely, for every $n > d > 0$, we show that there exists a planar graph G_n^d with n vertices and degree d such that any planar straight-line drawing of G_n^d with resolution ρ requires area $\Omega(c^{\rho n})$, where c is a constant greater than 1.

For a fixed d, we recursively define graph G_k as follows. Graph G_0 is a triangle consisting of three vertices A_0, B_0, and C_0, as shown in Fig. 4.3(a). Graph G_k is obtained from G_{k-1} by adding $d-4$ new vertices and $3(d-4)+3$ new edges, as shown in Fig. 4.3(b). For $k \geq 2$, G_k has $k(d-4)+3$ vertices and degree d. Graph G_n^d is defined as $G_{\frac{n-3}{d-4}}$, whenever $n-3$ is a multiple of $d-4$.

First, we give a technical lemma, which uses the notation of Fig. 4.3.

Lemma 14 *Given a planar straight line drawing of G_k, let Δ_i^k be the area of $\Delta A_k B_i C_i$, with $1 \leq i \leq (d-4)/2$, then in at least one of the following pairs of edges, both edges have lengths greater than $\sqrt{2\Delta_i^k}$: (a_i, a_{i+1}), (f_i, f_{i+1}), (d_i, e_{i+1}), and (d_i, b_{i+1}).*

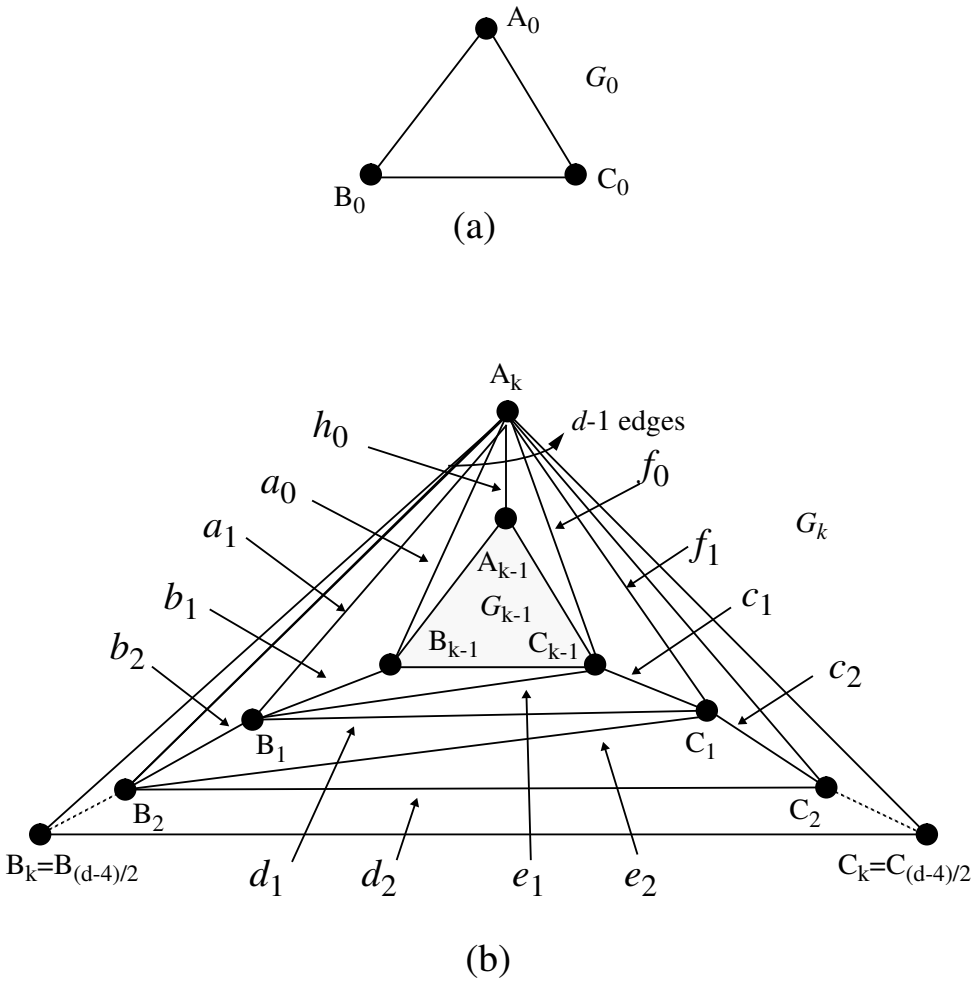

Figure 4.3: Graph G_n^d: (a) G_0; (b) Constructing G_k from G_{k-1}. Notice that G_n^d is defined as $G_{\frac{n-3}{d-4}}$, whenever $n-3$ is a multiple of $d-4$.

Proof: We know from elementary geometry that in a triangle with area Δ, at least two sides have lengths at least $\sqrt{2\Delta}$. Hence, at least one of a_i and d_i has length at least $\sqrt{2\Delta_i^k}$. Let β_i be the internal angle at B_i of triangle ΔAB_iC_i. We distinguish two cases.

$\beta_i < 90°$ (see Fig. 4.4(a)): Let B_il be a line parallel to d_i. Let B_im be a line parallel to a_i. Because of planarity, vertex B_{i+1} has to be placed inside the shaded region bounded by the lines B_il and B_im. Because $\angle A_kB_il > 90$, a_{i+1} is longer than a_i. Because $\angle C_iB_im > 90$, e_{i+1} is longer than d_i. Hence, the lemma is true for (a_i, a_{i+1}) or (d_i, e_{i+1}).

$\beta_i \geq 90°$ (see Fig. 4.4(b,c)): Edge f_i has length at least $\sqrt{2\Delta_i^k}$ because it is opposite to the largest angle (β_i) of Triangle ΔAB_iC_i. We consider two subcases:

Edge d_i has length at least $\sqrt{2\Delta_i^k}$ (see Fig. 4.4(b)): Let lC_ism and urv be lines perpendicular to d_i. Let C_it and wB_i be lines parallel to f_i. Let B_isz be

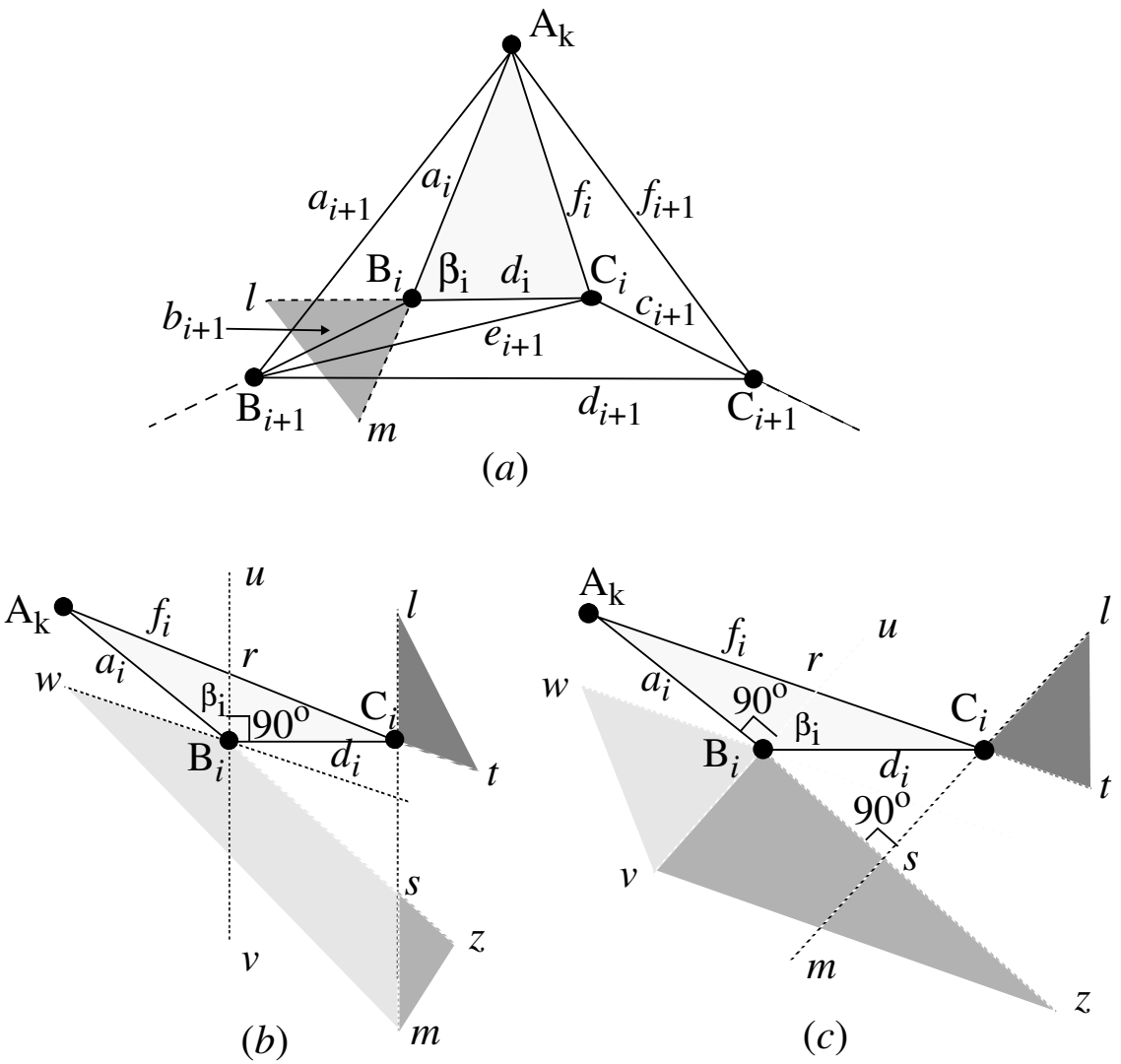

Figure 4.4: Let β_i be the internal angle $\angle A_k B_i C_i$ of the triangle $\Delta A_k B_i C_i$. (a) Case 1: $\beta_i < 90°$; (b) Case 2: $\beta_i \geq 90°$ and $d_i > \sqrt{\Delta_i}$; (c) Case 2: $\beta_i \geq 90°$ and $a_i > \sqrt{\Delta_i}$

a line parallel to a_i. If B_{i+1} is placed in the shaded region wB_ism then, because of planarity, C_{i+1} has to be placed in the shaded region lC_it. Since $f_{i+1} \geq rC_i > d_i$, we have that f_{i+1} is longer than d_i. Thus the lemma is true for the pair (f_{i+1}, f_i). Else, if B_{i+1} is placed in the shaded region zsm, then b_{i+1} is longer than d_i. Thus the lemma is true for the pair (b_{i+1}, d_i).

Edge a_i has length at least $\sqrt{2\Delta_i^k}$ (see Fig. 4.4(c)): Let lC_ism and urv be lines perpendicular to a_i. Let C_it and wB_i be lines parallel to f_i. Let B_isz be a line parallel to a_i. If B_{i+1} is placed in the shaded region wB_iv, then C_{i+1} has to be placed in the shaded region lC_it to avoid crossings. Because $\angle A_kC_il \geq 90°$, f_{i+1} is longer than f_i, and the lemma is true for the pair (f_{i+1}, f_i). Else, if B_{i+1} is placed in the shaded region vB_iz, then because $\angle A_kB_iv = 90°$, a_{i+1} is longer than a_i, and the lemma is true for the pair (a_{i+1}, a_i).

□

We are now ready to state the main result of this section:

Theorem 11 *There exists a constant $c > 1$ such that, for any $n > d > 6$, with $n-3$ a multiple of $d-4$, there exists a planar graph G_n^d with n vertices and degree d such that any planar straight-line drawing of G_n^d with angular resolution ρ has area $\Omega(c^{\rho n})$.*

Proof: Let n and d be two integers such that $n > d > 6$, and $n-3$ is a multiple of $d-4$. Let $k = (n-3)/(d-4)$. Consider a planar straight-line drawing D of G_k with $A_kB_kC_k$ as the external face. We denote with Δ_k the area of D, and with Δ_i^k the area of Triangle $\Delta A_kB_iC_i$, where $1 \leq i \leq (d-4)/2+1$, in Drawing D. From Lemma 14, we have

$$\Delta_{i+1}^k > \Delta_i^k + \frac{1}{2}\sqrt{2\Delta_i^k}\sqrt{2\Delta_i^k}\sin\rho > \Delta_i^k(1+\sin\rho). \tag{4.8}$$

Since $\Delta_k = \Delta^k_{\frac{d-4}{2}+1}$ and $\Delta_1^k = \Delta_{k-1}$, we obtain the recurrence

$$\Delta_k > \Delta_{k-1}(1+\sin\rho)^{\frac{d-4}{2}}. \tag{4.9}$$

Recalling the recursive construction of G_k, we get

$$\Delta_k > \Delta_0(1+\sin\rho)^{\frac{d-4}{2}k} = \Delta_0(1+\sin\rho)^{\frac{d-4}{2}\frac{n-3}{d-4}} = \Delta_0(1+\sin\rho)^{\frac{n-3}{2}} \tag{4.10}$$

Since $\sin\rho \geq \frac{1}{2}\rho$, we have $\Delta_k > \Delta_0(1+\rho/2)^{\frac{n-3}{2}}$. This gives,

$$\log\Delta_k > \log\Delta_0 + \frac{n-3}{2}\log(1+\rho/2) \tag{4.11}$$

Because the smallest angle in a triangle can be at most $\pi/3$, we have that $\rho \leq \pi/3 < 1$. Using the expansion, $\log(1+y) = y - y^2/2 + y^3/3 - y^4/4 + \ldots$, for $y \leq 1$, we have that $\log\Delta_k > \log\Delta_0 + ((n-3)/2)\rho(1/2 - \pi/24)$. Therefore, $\Delta_k > r^{\rho n}$ for some constant $r > 1$. Finally, we observe that in any planar drawing of G_k with $A_kB_kC_k$ not being the external face, there is a subgraph of G_k isomorphic to $G_{k/2}$, that is drawn with $A_{k/2}B_{k/2}C_{k/2}$ as the external face. We conclude that any planar straight line drawing of G_k has area $\Omega(c^{\rho n})$ for some constant $c > 1$. □

Note that Theorem 11 is fully general, since it holds under any rule that implies a finite minimum area for the drawing of a graph.

The following corollary is immediate (this corollary is also proved independently in [83]).

Corollary 4 *There exists an infinite family of bounded-degree graphs that require exponential area in any planar straight-line drawing with bounded (from below) angular resolution.*

4.5 Constructing Drawings with Large Angular Resolution

In this section we give linear-time algorithms for constructing planar straight-line drawings with large angular resolution for various classes of planar graphs. We show that every nested-star graph and series-parallel graph admits a planar straight-line drawing with angular resolution $\Omega(1/d^2)$, and that every maximal outerplanar graph admits a planar straight-line drawing with angular resolution $\pi/(d-1)$, where d denotes the degree of a graph.

4.5.1 A Geometric Lemma

In this section we give a geometric lemma that is used in our algorithms for drawing nested-star and series-parallel graphs.

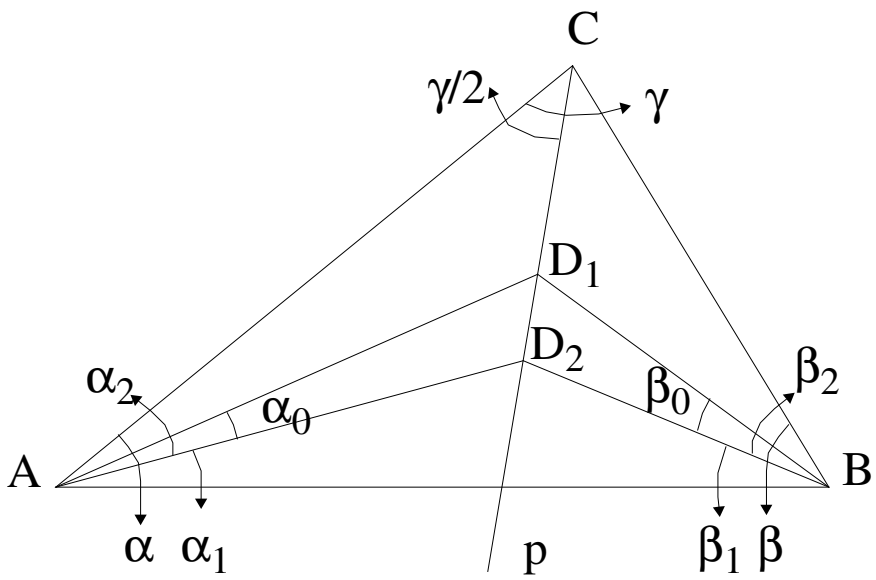

Figure 4.5: Triangle ΔABC in Lemma 15

Lemma 15 *Let ΔABC be a triangle with internal angles $\alpha = \angle CAB$, $\beta = \angle ABC$ and $\gamma = \angle BCA$ such that $\alpha \leq \beta \leq \pi/2$, as shown in Fig. 4.5. Let p be the bisector of angle γ, and D_1 and D_2 be any two points on p inside ΔABC. Let $\alpha_0 = \angle D_1AD_2$ and $\beta_0 = \angle D_1BD_2$. Then*

$$\frac{\beta_0}{\beta} \geq \frac{1}{4\pi^2}\frac{\alpha_0}{\alpha}$$

Proof: Consider Fig. 4.5. Let $\alpha_2 = \angle D_2AC$, $\alpha_1 = \angle D_2AB$, $\beta_2 = \angle D_2BC$, and $\beta_1 = \angle D_2BA$. Since $\alpha \leq \pi/2$ and $\beta \leq \pi/2$, angles α_0, α_1, α_2, β_0, β_1, and β_2 are all at most $\pi/2$. Since p is the bisector of γ, using Eq. 4.2 for D_2 we get

$$\sin\alpha_2 \sin\beta_1 = \sin\beta_2 \sin\alpha_1 \tag{4.12}$$

$\beta_2 \geq \alpha_2$ since otherwise using Eq. 4.12, we can show that $\beta_1 < \alpha_1$ which in turn implies that $\beta = \beta_2 + \beta_1 < \alpha_2 + \alpha_1 = \alpha$. Again using Eq. 4.2, this time for D_1 we get,

$$\frac{\sin(\alpha_2 - \alpha_0)}{\sin(\alpha_1 + \alpha_0)} = \frac{\sin(\beta_2 - \beta_0)}{\sin(\beta_1 + \beta_0)}$$

This gives

$$\frac{\sin\alpha_2}{\sin\alpha_1}\frac{1 - \cot\alpha_2 \tan\alpha_0}{1 + \cot\alpha_1 \tan\alpha_0} = \frac{\sin\beta_2}{\sin\beta_1}\frac{1 - \cot\beta_2 \tan\beta_0}{1 + \cot\beta_1 \tan\beta_0}$$

Let $\rho_0 = \tan\beta_0 / \tan\alpha_0$. After using Eq. 4.12, substituting $\tan\beta_0$ by $\rho_0 \tan\alpha_0$ and then collecting the terms containing ρ_0, we get

$$\begin{aligned} \rho_0 &= \frac{\cot\alpha_2 + \cot\alpha_1}{\cot\beta_2 + \cot\beta_1 + (\cot\alpha_1 \cot\beta_2 - \cot\alpha_2 \cot\beta_1)\tan\alpha_0} \\ &\geq \frac{\cot\alpha_2 + \cot\alpha_1}{\cot\beta_2 + \cot\beta_1 + \cot\alpha_1 \cot\beta_2 \tan\alpha_0} \end{aligned} \tag{4.13}$$

The following can be easily shown by elementary trigonometry

$$\begin{aligned} \cot\alpha_2 + \cot\alpha_1 &= \frac{\sin(\alpha_1 + \alpha_2)}{\sin\alpha_2 \sin\alpha_1} = \frac{\sin\alpha}{\sin\alpha_2 \sin\alpha_1} \\ \cot\beta_2 + \cot\beta_1 &= \frac{\sin(\beta_1 + \beta_2)}{\sin\beta_2 \sin\beta_1} = \frac{\sin\beta}{\sin\beta_2 \sin\beta_1} \end{aligned} \tag{4.14}$$

Since $\alpha_0 \leq \alpha_2 \leq \pi/2$, $\tan\alpha_0 \leq \tan\alpha_2$. Therefore $\cot\alpha_1 \cot\beta_2 \tan\alpha_0 \leq \cot\alpha_1 \cot\beta_2 \tan\alpha_2$. Using Eq. 4.12 we get,

$$\begin{aligned} \cot\alpha_1 \cot\beta_2 \tan\alpha_0 &\leq \cot\alpha_1 \cot\beta_2 \tan\alpha_2 = \frac{\cos\alpha_1 \cos\beta_2}{\sin\alpha_1 \sin\beta_2}\frac{\sin\alpha_2}{\cos\alpha_2} \\ &= \frac{\cos\alpha_1 \cos\beta_2}{\sin\beta_1 \sin\alpha_2}\frac{\sin\alpha_2}{\cos\alpha_2} = \frac{\cos\alpha_1}{\sin\beta_1}\frac{\cos\beta_2}{\cos\alpha_2} \end{aligned}$$

We have show earlier that $\alpha_2 \leq \beta_2 \leq \pi/2$. Therefore $\cos\beta_2 \leq \cos\alpha_2$. Hence

$$\cot\alpha_1 \cot\beta_2 \tan\alpha_0 \leq \frac{\cos\alpha_1}{\sin\beta_1} \leq \frac{1}{\sin\beta_1} \tag{4.15}$$

From Eqs 4.13, 4.14, and 4.15 we get,

$$\rho_0 \geq \frac{\frac{\sin\alpha}{\sin\alpha_2\sin\alpha_1}}{\frac{\sin\beta}{\sin\beta_2\sin\beta_1}+\frac{1}{\sin\beta_1}} = \frac{\frac{\sin\alpha}{\sin\alpha_2\sin\alpha_1}}{(\frac{\sin\beta}{\sin\beta_2}+1)\frac{1}{\sin\beta_1}}$$

Since $\sin\beta/\sin\beta_2 \geq 1$,

$$\rho_0 \geq \frac{\sin\alpha/(\sin\alpha_2\sin\alpha_1)}{2\sin\beta/(\sin\beta_2\sin\beta_1)} = \frac{\sin\alpha}{2\sin\beta}\frac{\sin\beta_2\sin\beta_1}{\sin\alpha_2\sin\alpha_1}$$

Using Eq. 4.12, $\sin\beta_2\sin\beta_1/(\sin\alpha_2\sin\alpha_1) = \sin^2\beta_2/\sin^2\alpha_2 = \sin^2\beta_1/\sin^2\alpha_1$. If $j \in \{1,2\}$ be such that $\beta_j = \max(\beta_2,\beta_1)$, then

$$\rho_0 \geq \frac{\sin\alpha\sin^2\beta_j}{2\sin\beta\sin^2\alpha_j}$$

Since $\beta/2 \leq \beta_j \leq \pi/2$ and $\alpha_j \leq \alpha \leq \pi/2$, $\sin\beta/2 \leq \sin\beta_j$ and $\sin\alpha_j \leq \sin\alpha$. Therefore

$$\frac{\tan\beta_0}{\tan\alpha_0} = \rho_0 \geq \frac{\sin^2(\beta/2)}{2\sin\beta\sin\alpha} = \frac{\sin(\beta/2)}{4\cos(\beta/2)\sin\alpha} \geq \frac{\sin(\beta/2)}{4\sin\alpha}$$

$\beta \leq \pi/2$ and $\alpha \leq \pi/2$ as assumed in the lemma. Since $\tan\theta \geq \sin\theta \geq (2/\pi)\theta$ if $\theta \leq \pi/2$, and $\sin\theta \leq \theta$,

$$\tan\beta_0 \geq \frac{1}{4}\frac{(2/\pi)\beta/2}{\alpha}(2/\pi)\alpha_0 = \frac{1}{2\pi^2}\frac{\beta}{\alpha}\alpha_0 \tag{4.16}$$

If $\beta_0 > \pi/3$ then since $\beta \leq \pi/2$ (as assumed in the lemma),

$$\frac{\beta_0}{\beta} > \frac{2}{3} > \frac{1}{4\pi^2} \geq \frac{1}{4\pi^2}\frac{\alpha_0}{\alpha}$$

If $\beta_0 \leq \pi/3$ then $2\beta_0 \geq \tan\beta_0$. Therefore from Eq. 4.16, we get $\beta_0/\alpha_0 \geq (1/4\pi^2)(\beta/\alpha)$ and hence $\beta_0/\beta \geq (1/4\pi^2)\alpha_0/\alpha$. □

4.5.2 Nested-Star Graphs

A nested-star graph G is defined recursively as follows: G has three vertices called its *apex* vertices (denoted by u, v and w in Fig. 4.6). G either consists of a single triangle (see Fig. 4.6(a)) or is a *composition* of three nested-star graphs G_1, G_2 and G_3, and is constructed from them as shown in Fig. 4.6(b). Vertices u, w and z are also the apex vertices of G_1, vertices u, v and z are also the apex vertices of G_2 and vertices v, w and z are also the apex vertices of G_3. Graphs G_1, G_2 and G_3 are also called *constituents* of G.

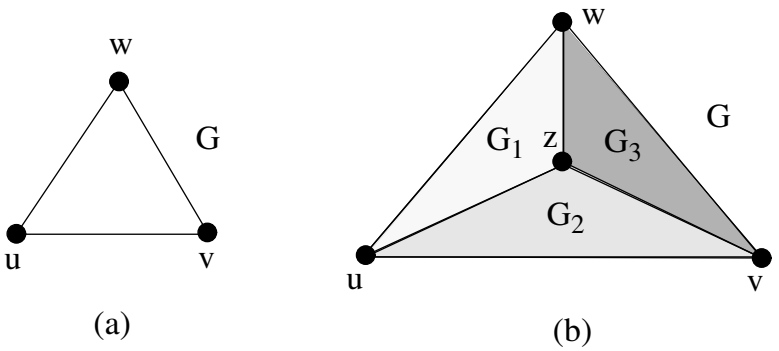

Figure 4.6: A nested-star graph G: (a) G consists of a single triangle; (b) G is a composition of three nested-star graphs G_1, G_2 and G_3;

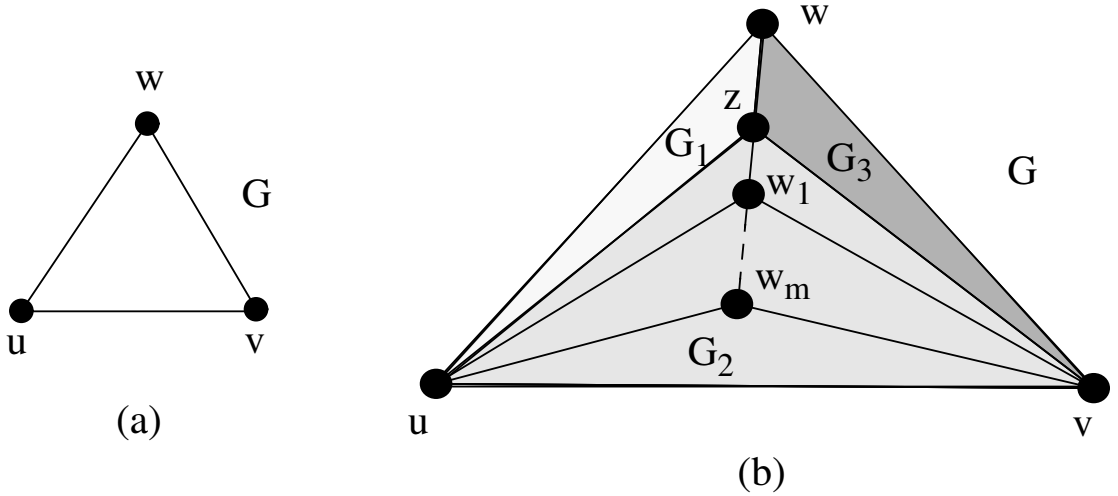

Figure 4.7: Proof of Lemma 16 for a nested-star graph G: (a) G consists of a single triangle; (b) G is a composition of three nested-star graphs G_1, G_2 and G_3;

A *nested-star subgraph* of G is either G itself or a nested-star subgraph of a constituent of G.

The following lemma states a property of nested-star graphs that is used by our drawing algorithm.

Lemma 16 *Let G be a nested-star graph. Let u and v be two apex vertices of G. Then, there exists a path $p = w_1 w_2 \ldots w_m$ in G, consisting of edges $w_i w_{i+1}$, where $1 \leq i \leq m-1$, such that (a) each w_i is a common neighbor of u and v, and (a) every common neighbor x of u and v belongs to p, i.e., there exists an i, where $1 \leq i \leq m$, such that $x = w_i$.*

Proof: We prove this by induction on the number of vertices in G. When G has only three vertices, u, v and w (see Fig. 4.7(a)), then the lemma is true for the path p consisting of only one vertex, namely w, because w is the only common neighbor of u and v. Now suppose G is a composition of three nested-star graphs G_1, G_2 and G_3 (see Fig. 4.7(b)). Let u, v and w be the apex vertices of G. Since G_2 has at least one

vertex less than G, the lemma is true for the common neighbors of u and v in G_2. Let $p = zw_1w_2 \ldots w_m$ be the path consisting of all the common neighbors of u and v in G_2. u and v have only one more common neighbor in G, namely w. Because there is an edge (w, z) in G, there is a path $p' = wzw_1w_2 \ldots w_m$ in G that consists of all and only the common neighbors of u and v in G. □

The Drawing Algorithm

We now describe a recursive algorithm that constructs a planar straight-line drawing with angular resolution at least $1/(24\pi d^2)$, of a nested-star graph with degree d. This algorithm runs in linear time.

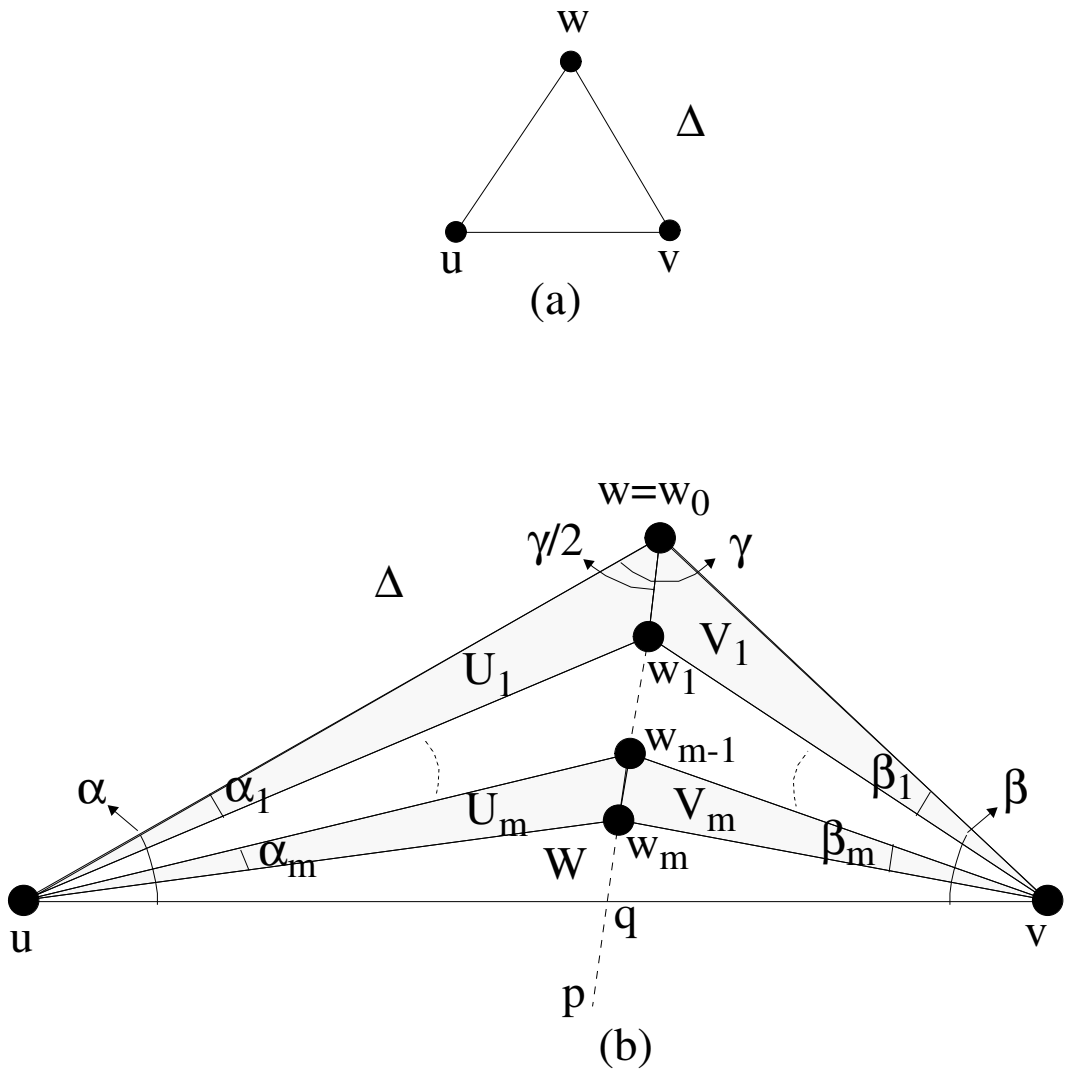

Figure 4.8: Constructing Drawing $D(G, \Delta)$, when the nested-star graph G consist of a: (c) A single triangle; (d) nested-star graphs $U_1, U_2, \ldots U_m$ and $V_1, V_2, \ldots V_m$

Let u, v and w be the apex vertices of G. Given a triangle Δ, which is said to be

allocated to G, we show how to construct a drawing $D(G,\Delta)$ of G such that in $D(G,\Delta)$, (a) each vertex and edge of G is drawn entirely inside Δ, and (b) the apex vertices of G are placed at the vertices of Δ (see Fig. 4.8(a,b)). For constructing $D(G,\Delta)$, we consider two cases:

Case 1: G consists of three and only three vertices: Simply draw G such that its apex vertices coincide with Δ (see Fig. 4.8(a)).

Case 2: G consists of at least four vertices: G is drawn as shown in Fig. 4.8(b). Let α, β and γ be the three internal angles of Δ. Assume without loss of generality that $\gamma \geq \beta \geq \alpha$. From Lemma 16, there is a path $p = w_0(= w)w_1, w_2 \ldots w_m$ consisting of all and only the common neighbors of u and v. Let U_i be the nested-star subgraph of G with w_{i-1}, u and w_i as its apex vertices. Let V_i be the nested-star subgraph of G with w_{i-1}, v and w_i as its apex vertices. Notice that uw_mv is a face of G. Let W be the nested-star subgraph of G which consists only of the triangle uw_mv. Let wp be a line that bisects angle γ. $D(G,\Delta)$ is constructed as follows:

1. For each $i \geq 1$, place w_i on line p in the interior of Δ such that $\angle w_{i-1}uw_i$ is $(d_{U_i}(u) - 1)\alpha/(d_G(u) - 1)$. (Recall that $d_G(u)$ denotes the degree of vertex u in graph G.)

2. (Recursively) construct Drawings $D(U_i, \Delta w_{i-1}uw_i)$ and $D(V_i, \Delta w_{i-1}vw_i)$.

Theorem 12 *A nested-star graph with degree d and n vertices admits a planar straight-line drawing with angular resolution at least $1/(24\pi d^2)$ that can be constructed in $O(n)$-time.*

Proof: Let H be a nested-star graph with degree d and n vertices. Let Δ_E be an equilateral triangle. Construct Drawing $D(H,\Delta_E)$ using the recursive algorithm described above. It is easy to see that the algorithm executes in $O(n)$-time.

We prove now that $D(H,\Delta)$ has angular resolution at least $1/(24\pi d^2)$. For this we show that the algorithm maintains Invariant 1 given below. It is easy to see that if Invariant 1 is maintained, then because each vertex u of H has degree at most d, $D(H,\Delta_E)$ has angular resolution at least $1/(24\pi d^2)$.

Invariant 1: Let G be a nested-star subgraph of H. Let Δ be the triangle allocated to G by the algorithm. Let u be the apex vertex of G, called its *critical vertex*, such that $\angle vuw \leq \angle uvw$ and $\angle vuw \leq \angle uwv$ in Drawing $D(G,\Delta)$, where v and w are the other two apex vertices of G. Then:

1. $(d_G(u) - 1)/(24\pi d^2) \leq \angle vuw \leq \pi/6$, and
2. if G has at least four vertices then, $\angle uvw \geq \pi/6$ and $\angle uwv \geq \pi/6$, otherwise, $\angle uvw \geq 1/(24\pi d^2)$ and $\angle uwv \geq 1/(24\pi d^2)$.

We now prove that Invariant 1 holds for every nested-star subgraph G of H. We prove this by reverse induction on the number of vertices in G.

Clearly, when $G = H$ Invariant 1 holds for G because Δ_E is an equilateral triangle, and each apex vertex of G has degree at most d.

Now suppose G has at least four vertices, and that Invariant 1 holds for G. Let Δ be the triangle allocated to G. Drawing $D(G, \Delta)$ is constructed as shown in Fig. 4.8(b) and as described in Case 2 of the drawing algorithm given earlier in this section. The rest of this proof uses the notation of Fig. 4.8(b). Recall that $\gamma \geq \beta \geq \alpha$. Since Invariant 1 holds for G, $\beta \geq \pi/6$, and $(d_G(u)-1)/(24\pi d^2) \leq \alpha \leq \pi/3$. Since γ is the largest internal angle of Δ, we have that $\gamma \geq \pi/3$ and $\beta \leq \pi/2$. Also recall that Line wp bisects Angle γ. Let q be the intersection of Line wp with Line uv. Each U_i and V_i, as well as the nested-star graph W has at least one vertex less than G. We now show that Invariant 1 holds for W, as well as for each U_i and V_i.

We first show that Invariant 1 holds for U_i, and u is the critical vertex of U_i. Let $\alpha_i = \angle w_{i-1}uw_i$. $\angle uw_{i-1}w_i \geq \angle uww_1 = \gamma/2 \geq \pi/6$. Because $\alpha \leq \pi/6$, we have that $\angle uw_iw_{i-1} \geq \angle uqw = \pi - \gamma/2 - \alpha > \pi/6$. $\alpha_i = (d_{U_i}(u) - 1)\alpha/(d_G(u) - 1)$. Because Invariant 1 holds for G, $\alpha \geq (d_G(u) - 1)/24\pi d^2$. Hence $\alpha_i \geq d_{U_i}(u)/24\pi d^2$. Therefore Invariant 1 holds for U_i.

Now we show that Invariant 1 holds for V_i, and v is the critical vertex of V_i. Let $\beta_i = \angle w_{i-1}vw_i$. $\angle vw_{i-1}w_i \geq \angle vww_1 = \gamma/2 \geq \pi/6$. Because $\beta \leq \pi/2$, we have that $\angle vw_iw_{i-1} \geq \angle vqw = \pi - \gamma/2 - \beta = \pi - (\gamma + \beta)/2 - \beta/2 \geq \pi/2 - \beta/2 > \pi/6$. From Lemma 15, $\beta_i \geq (\alpha_i/4\pi^2\alpha)\beta$. Since $\beta \geq \pi/6$ and $\alpha_i = (d_{U_i}(u) - 1)\alpha/(d_G(u) - 1)$,

$$\beta_i \geq \frac{1}{4\pi^2}\frac{d_{U_i}(u) - 1}{d_G(u) - 1}\frac{\pi}{6}$$

since $d_G(u) \leq d$, $d_{U_i}(u) \geq 2$ and $d_{V_i}(v) \leq d$,

$$\beta_i \geq \frac{1}{24\pi d} \geq \frac{d_{V_i}(v)}{24\pi d^2}$$

Therefore Invariant 1 holds for V_i.

Now we show that Invariant 1 holds for W. Notice that W consists of three vertices only, and $d_W(u) = d_W(v) = 2$. $\angle vuw_m = \alpha - \sum_i (d_{U_i}(u) - 1)\alpha/(d_G(u) - 1) = \alpha - (d_G(u) - 2)\alpha/(d_G(u) - 1) = \alpha/(d_G(u) - 1) = (d_W(u) - 1)\alpha/(d_G(u) - 1)$. Because Invariant 1 holds for G, $\alpha \geq (d_G(u) - 1)/24\pi d^2$. Therefore, $\angle vuw_m \geq (d_W(u) - 1)/24\pi d^2$. From Lemma 15, $\angle uvw_m \geq (\angle vuw_m/4\pi^2\alpha)\beta$. Since $\beta \geq \pi/6$ and $\angle vuw_m \geq \alpha/(d_G(u) - 1) > \alpha/d$, we have that $\angle uvw_m > 1/24\pi d^2 = (d_W(v) - 1)/24\pi d^2$. $\angle uw_mv \geq \angle uwv = \gamma \geq \pi/3$. Hence Invariant 1 holds for W also, where either u or v is the critical vertex of W. □

Fig. 4.9(b) shows a drawing, with angular resolution $60°/(7 - 1) = 10°$, of the nested-star graph G of Fig. 4.9(a), constructed by our algorithm. Notice that G has degree 7.

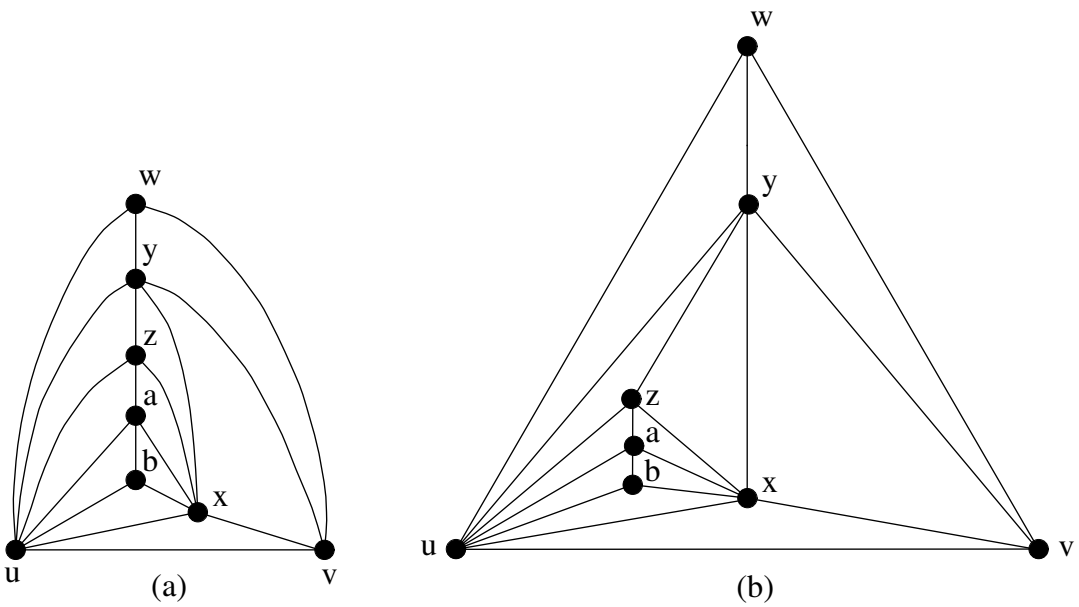

Figure 4.9: (a) A nested-star graph G with degree 7; (b) A drawing of G, with angular resolution $60°/(7-1) = 10°$, constructed by the algorithm of Section 4.5.2.

4.5.3 Series-Parallel Graphs

In this section, we give a linear time algorithm for constructing a planar straight-line drawing with angular resolution at least $1/48\pi d^2$, of a series-parallel graph with degree d.

A *series-parallel* graph G with *poles* s and t is defined recursively as follows: G is either a single edge (s,t) (see Fig 4.10(a)), or a *series composition* (see Fig 4.10(b)), or a *parallel composition* (see Fig 4.10(c)) of two series-parallel graphs G_1 and G_2. Let s_1 and t_1 be the poles of G_1. Let s_2 and t_2 be the poles of G_2. A *series composition* of G_1 and G_2 is done by identifying vertices s_2 and t_1 (see Fig 4.10(b)); For G in this case, $s = s_1$ and $t = t_2$. A *parallel composition* of G_1 and G_2 is done by identifying vertices t_2 and t_1, and the vertices s_2 and s_1(see Fig 4.10(c)); For G in this case, $s = s_1 = s_2$ and $t = t_1 = t_2$. A *series-parallel subgraph* of G is either G itself, or a series-parallel subgraph of G_1 or G_2. Associated with each series-parallel graph G is a binary tree called the *SPQ-Tree* [6] (also known as a *decomposition tree* or a *parse tree*) of G, denoted by $SPQ(G)$. $SPQ(G)$ has three type of nodes– S, P and Q. There is a one-to-one correspondence between the series-parallel subgraphs of G and the nodes of $SPQ(G)$. The root of $SPQ(G)$ corresponds to G. Tree $SPQ(G)$ is defined recursively as follows: If G is a single edge then $SPQ(G)$ consists of a single Q node (Fig. 4.10(d)). If G is a series composition of two series parallel graphs G_1 and G_2 then the root of $SPQ(G)$ is a S node, as shown in Fig. 4.10(e) (Fig. 4.10(f)). If G is a parallel composition of two series parallel graphs G_1 and G_2 then the root of $SPQ(G)$ is a P node, as shown in Fig. 4.10(e) (Fig. 4.10(f)). The two subtrees of $SPQ(G)$ are $SPQ(G_1)$ and $SPQ(G_2)$ (the order of the subtrees is arbitrary). Thus a S node represents a series composition of two series parallel graphs and a P node represents a parallel composition of two series parallel graphs. Q nodes

are the leaves of $SPQ(G)$. Fig. 4.11(b) shows a SPQ-tree that is associated with the series-parallel graph shown in Fig. 4.11(a). Notice that if G has no multiple edges, at most one child of a P node is a Q node. Given G, $SPQ(G)$ can be constructed in linear time using the algorithm given in [108].

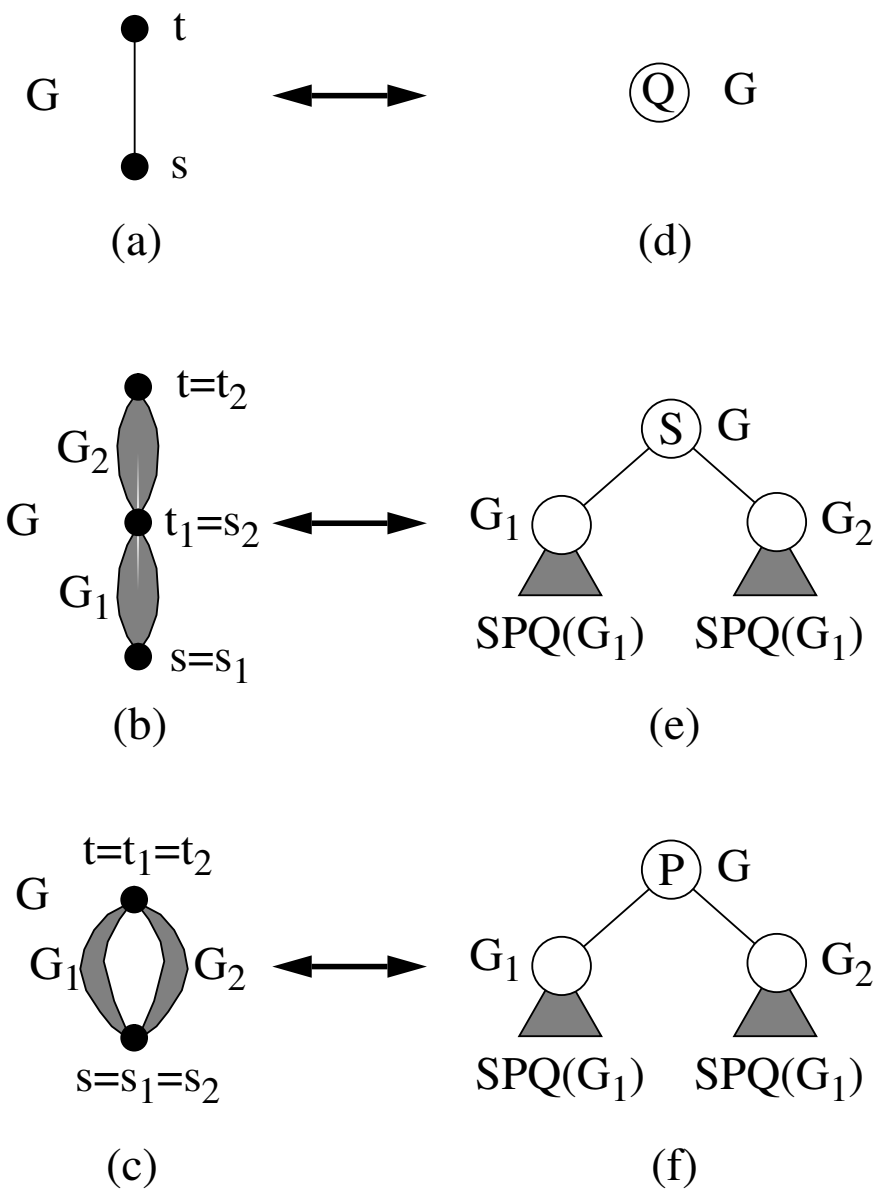

Figure 4.10: A series-parallel graph G when it is: (a) a single edge (s,t), (b) a series-composition of series-parallel graphs G_1 and G_2, and (c) a parallel-composition of series-parallel graphs G_1 and G_2. (d) Corresponding Q node when G is a single edge; (e) Corresponding S node when G is a series-composition of G_1 and G_2; (f) Corresponding P node when G is a parallel-composition of G_1 and G_2.

A *compact* SPQ-tree is a tree whose vertices are of three types, S, P and Q, such that there are no two consecutive P nodes on any path from leaf to root. Each SPQ-tree

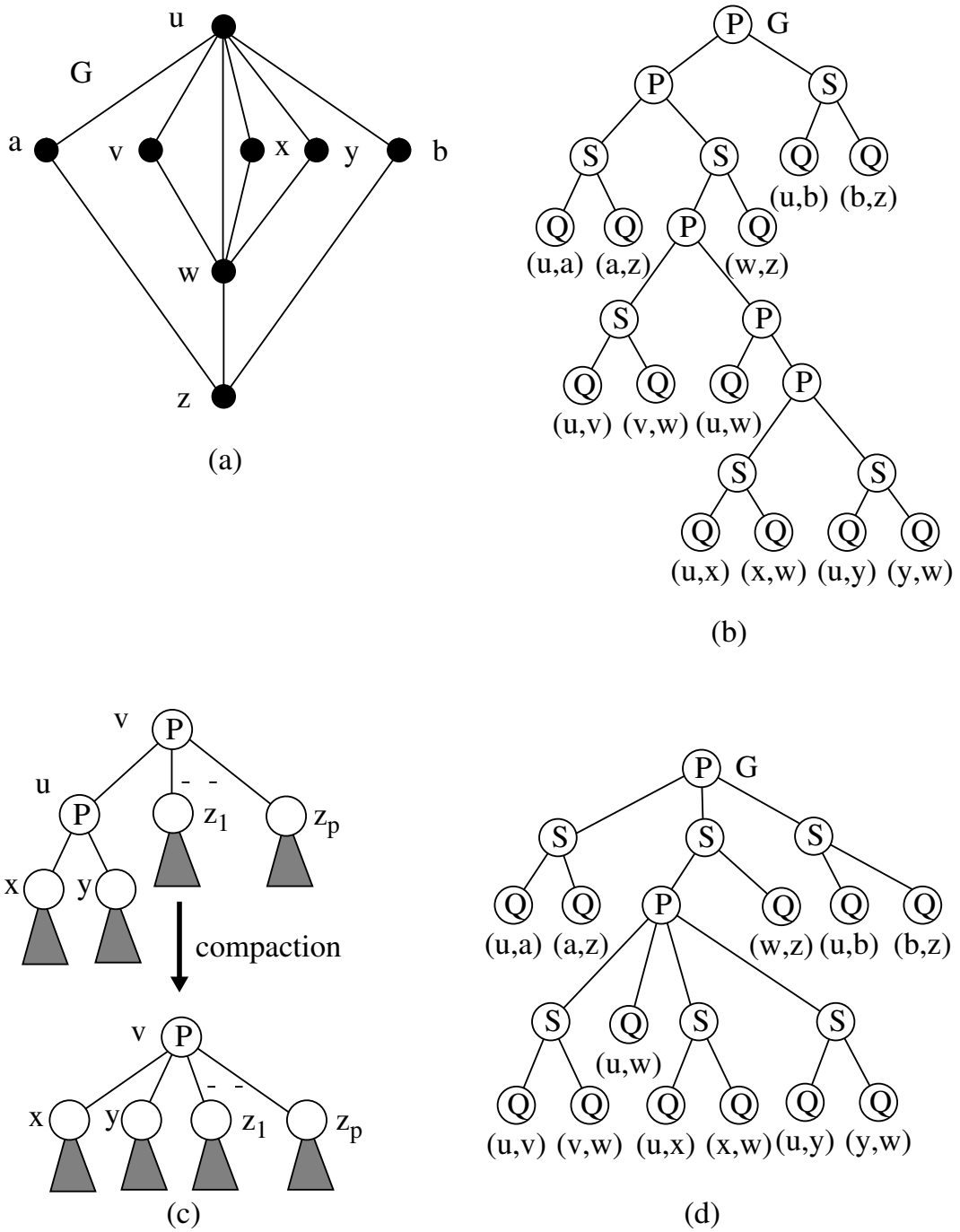

Figure 4.11: Example of a SPQ-tree $SPQ(G)$ and its equivalent compact SPQ-tree $T(G)$: (a) A series-parallel graph G; (b) SPQ-tree $SPQ(G)$; (c) Compacting vertex u into vertex v; (d) The compact SPQ-tree $T(G)$.

can be converted into an equivalent compact SPQ-tree in linear time by a process called *compaction* (see Fig. 4.11(c))in which we recursively merge a P node u into its parent v if v is also a P node so that the children of u become the children of v. Fig. 4.11(d) shows a compact SPQ-tree equivalent to the SPQ-tree of Fig. 4.11(b).

As in a SPQ-tree, each node of a compact SPQ-tree corresponds to a series-parallel graph. The root corresponds to the series-parallel graph associated with the equivalent SPQ-tree. A P node represents a parallel composition of two or more series-parallel graphs and a S node represents a series composition of two series-parallel graphs. Every series-parallel graph G has a compact SPQ-tree associated with it.

Let G be a series-parallel graph with degree d, poles s and t. Let $T(G)$ be a compact SPQ-tree associated with G. Given a triangle Δ, which is said to be *allocated* to G (see Fig. 4.12), we show how to construct a drawing $D(G, \Delta)$ of G such that in $D(G, \Delta)$, (a) each vertex and edge of G is drawn entirely inside Δ, and (b) s and t are placed at two vertices of Δ. The third vertex of Δ is called its *free* vertex. Let r be the free vertex of Δ. For constructing $D(G, \Delta)$, we consider three cases:

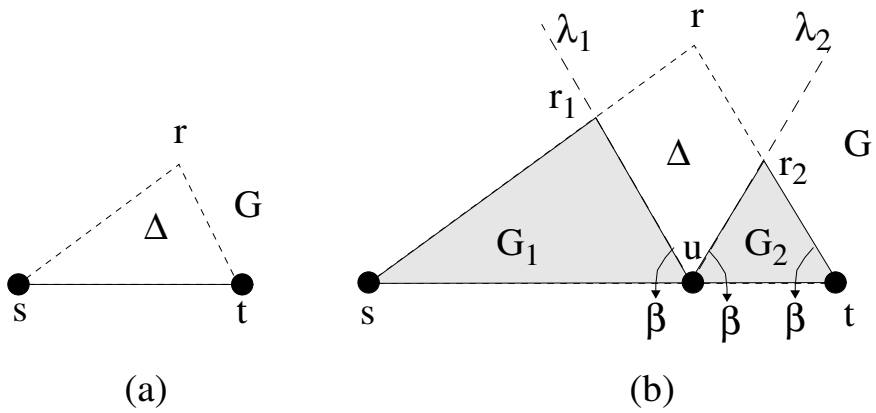

Figure 4.12: Drawing a series-parallel graph G: (a) Drawing G when it is a single edge; (b) Drawing G when it is a series composition of series-parallel graphs G_1 and G_2;

1. *Case Q*: Root of $T(G)$ is a Q node. Recall that in this case, G is a single edge (s, t). We draw G as shown in Fig 4.12(a).

2. *Case S*: Root of $T(G)$ is a S node. In this case, G is a series composition of two series-parallel graphs G_1 and G_2, where s is also a pole of G_1 and t is also a pole of G_2 (see Fig. 4.10(b)). Assume without loss of generality that $\angle rst \leq \angle rts$. We draw G as shown in Fig. 4.12(b). Let r_2 be the midpoint of the line-segment rt. Let λ_2 be the line passing through r_2 making an angle β with the line st. Let q be the intersection of λ_2 and st. Let λ_1 be the line parallel to the line rt passing through q. r_1 is the intersection of λ_1 and the line sr. We allocate Triangle Δsqr_1 to G_1 and Triangle Δtqr_2 to G_2, and (recursively) construct Drawings $D(G_1, \Delta sqr_1)$ and $D(G_2, \Delta sqr_2)$, such that r_1 and r_2 are the free vertices of Δsqr_1 and Δtqr_2 respectively,

3. *Case P*: Root u of $T(G)$ is a P node. Let G be a parallel composition of the

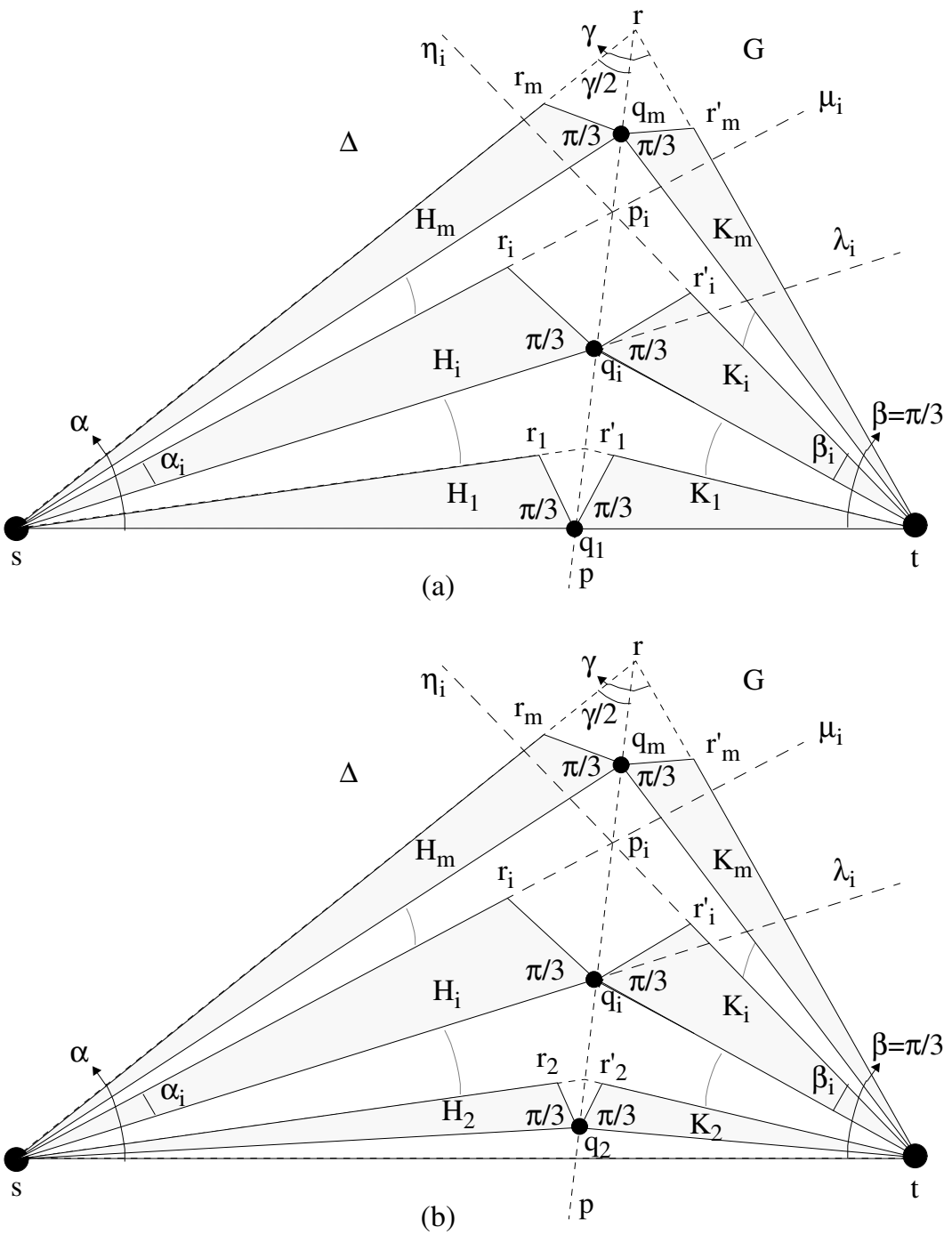

Figure 4.13: Drawing a series-parallel graph G when it is a parallel composition of series-parallel graphs $G_1, G_2, \ldots, G_m$: (a) Case **i**; (b) Case **ii**.

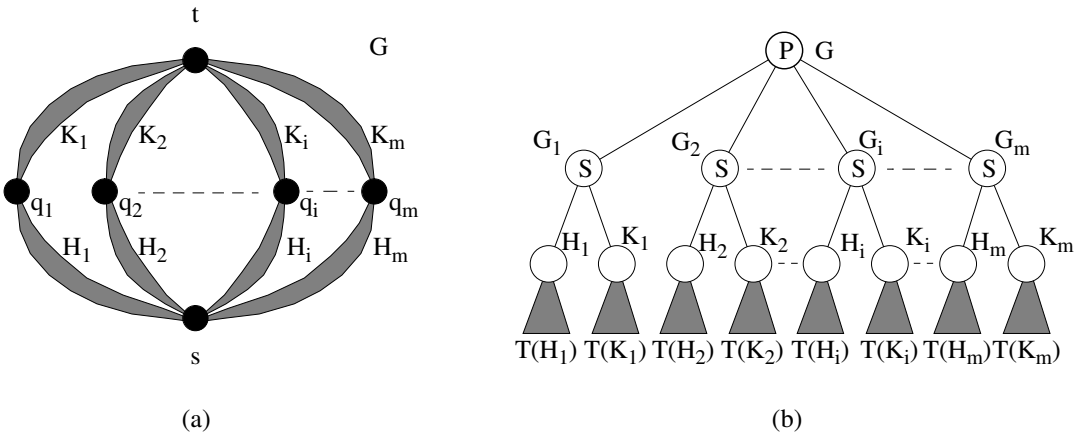

Figure 4.14: Case **i**. Series-parallel graph G is a parallel composition of series-parallel graphs $G_1, G_2, \ldots, G_m$, where each G_i consists of at least two edges: (a) Composition of G, (b) Compact SPQ-tree $T(G)$ associated with G.

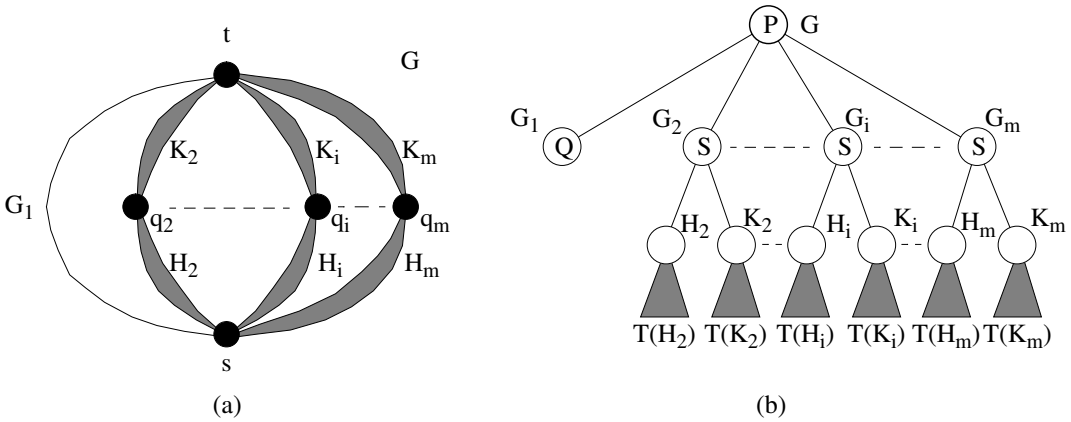

Figure 4.15: Case **i**. Series-parallel graph G is a parallel composition of series-parallel graphs $G_1, G_2, \ldots, G_m$, where G_1 consists of only one edge: (a) Composition of G, (b) Compact SPQ-tree $T(G)$ associated with G.

series-parallel graphs $G_1, G_2, \ldots, G_m$. Let u_i be the child of u that corresponds to G_i. We consider two cases: (**i**) No u_i is a Q node (see Fig. 4.14), or (**ii**) one and only one u_i is a Q node (see Fig. 4.15) and without any loss of generality, we assume that u_1 is that node. In both the cases, (**i**) and (**ii**), if u_i is not a Q node, then it is an S node and has two children: One child corresponds to a series-parallel graph, which we denote by H_i, whose one pole is s and the other child corresponds to a series-parallel graph, which we denote by K_i, whose one pole is t. Let q_i be the common pole of H_i and K_i. Fig. 4.13(a) shows the drawing of G in Case (**i**) and Fig. 4.13(b) shows the drawing of G in Case (**ii**). In both the cases, if u_i is a S node, we allocate Triangle Δsr_iq_i to H_i and Triangle $\Delta sr'_iq_i$ to K_i, and (recursively) construct Drawings $D(H_i, \Delta sr_iq_i)$ and $D(K_i, \Delta sr'_iq_i)$, such that r_i and r'_i are the free vertices of Δsr_iq_i and $\Delta sr'_iq_i$ respectively. In addition, in Case **ii**, we allocated Triangle Δsq_2t to G_1 (recall tha G_1 consists of a single edge),

and construct Drawing $D(K_i, \Delta sq_2t)$, such that q_2 is the free vertex of Δsq_2t.

The position of each r_i, q_i and r'_i is determined as follows: Assume without any loss of generality that $\angle rst \leq \angle rts$. Let $\alpha = \angle rst$. Let p be the bisector of angle $\angle srt$. For each H_i, draw lines λ_i and μ_i from s such that (a) the angle (α_i) between λ_i and μ_i is equal to $(2d_{H_i}(s) - 1)\alpha/(2d_G(s) - 1)$, and (b) the angle between μ_i and λ_{i+1} is equal to $\alpha/(2d_G(s) - 1)$. (Recall that we denote by $d_G(u)$, the degree of vertex u in G.) q_i is the intersection of λ_i with p. r_i is a point on μ_i such that $\angle r_iq_is$ is $\pi/3$. Let p_i be the intersection of p and μ_i. Let η_i be the line passing through t and p_i. r'_i is a point on η_i such that $\angle r'_iq_it$ is $\pi/3$.

Theorem 13 *A series-parallel graph with degree d and n vertices admits a planar straight-line drawing with angular resolution at least $1/(48\pi d^2)$ that can be constructed in $O(n)$-time.*

Proof:

Let G' be a series-parallel graph with degree d and n vertices. Construct a compact SPQ-tree $T(G)$ associated with G', by first constructing $SPQ(G')$ using the $O(n)$-time algorithm of [108], and then compacting $SPQ(G')$ in $O(n)$ time using the compacting procedure described earlier. Let $Delta_E$ be an equilateral triangle. Construct Drawing $D(G', Delta_E)$ using the recursive algorithm given above. It is easy to see that the algorithm executes in $O(n)$-time, giving a total time-complexity of $O(n)$.

We prove now that $D(G', \Delta_E)$ has angular resolution at least $1/(48\pi d^2)$. For this we show that the algorithm maintains Invariant 1 given below. It is easy to see that if Invariant 1 is maintained, then because each vertex of G' has degree at most d, $D(G', \Delta_E)$ has angular resolution at least $1/(48\pi d^2)$.

Invariant 2: Let G be a series-parallel subgraph of G' with poles s and t. Let Δ be the triangle allocated to G by the algorithm. Let r be the free vertex of Δ in the drawing $D(G, \Delta)$. Let u be the pole of G, such that in $D(G, \Delta)$, $\angle ruv \leq \angle rvu$, where v is the other pole of G. u is called the *critical vertex* of G (therefore, either s or t is the critical vertex of G). Then:

1. $(2d_G(u) - 1)/(48\pi d^2) \leq \angle ruv \leq \pi/3$, and
2. if G has at least three vertices then, $\angle rvu = \pi/3$ and $\angle urv \geq \pi/3$, otherwise, $\angle rvu \geq 1/(48\pi d^2)$ and $\angle urv \geq \pi/3$.

We now prove that Invariant 2 holds for every series-parallel subgraph G of G'. We prove this by reverse induction on the number of vertices in G.

Clearly, when $G = G'$ Invariant 1 holds for G because Δ_E is an equilateral triangle, and both s and t have degrees at most d.

Drawing $D(G, \Delta)$ is constructed as shown in Fig. 4.8(b) and as described in Case 2 of the drawing algorithm given earlier in this section. The rest of this proof uses the notation of Fig. 4.8(b).

Now suppose G has at least three vertices, and that Invariant 2 holds for G. Let Δ be the triangle allocated to G. Assume without loss of generality that s is the critical vertex of G. Let r be the free vertex of Δ. Let $\alpha = \angle tsr$, $\beta = \angle str$ and $\gamma = \angle srt$. Since Invariant 2 holds for G, $\alpha \geq (2d_G(s) - 1)/48\pi d^2$ and $\beta = \pi/3$. Therefore $\gamma < \pi - \beta = 2\pi/3$. We have two cases:

- *Case S*: G is a series-composition of two series-parallel graphs G_1 and G_2. Let u be the common pole of G_1 and G_2. Drawing $D(G, \Delta)$ is constructed as shown in Fig. 4.12(b), and as described in Case S of the drawing algorithm given earlier in this section. The rest of the proof for Case S uses the notation of Fig. 4.12(b). We designate s and u as the critical vertices of G_1 and G_2 respectively. $d_{G_1}(s) = d_G(s)$ and $\angle r_1us = \pi/3$. Therefore Invariant 2 holds for G_1, with s as its critical vertex. Also, Invariant 2 holds for G_2, with u as its critical vertex, because $\angle r_2tu = \pi/3$ and since $d_{G_2}(u) \leq d$, $\angle r_2ut = \pi/3 > (2d_{G_2}(u) - 1)/48\pi d^2$.

- *Case P*: G is a parallel composition of the series-parallel graphs $G_1, G_2, \ldots, G_m$ (see Figures 4.14 and 4.15). Drawing $D(G, \Delta)$ is constructed as shown in Fig. 4.13, and as described in Case P of the drawing algorithm given earlier in this section. The rest of the proof for Case S uses the notation of Fig. 4.13.

  We first show that Invariant 2 holds for each H_i, with s as its critical vertex. Let $\alpha_i = \angle r_isq_i$. $\alpha_i = (2d_{H_i}(s) - 1)\alpha/(2d_G(s) - 1) \geq (2d_{H_i}(s) - 1)/48\pi d^2$ because $\alpha \geq (2d_G(s) - 1)/48\pi d^2$. From basic geometry we know that $\angle p_iq_is \geq \pi - (\angle srq_i + \alpha)$. Because $\gamma < 2\pi/3$ as proved earlier, $\angle srq_1 = \gamma/2 < \pi/3$. Because $\alpha \leq \pi/3$, we have that $\angle p_iq_is \geq \angle rq_1s = \pi - (\alpha + \angle srp) > \pi - (\pi/3 + \pi/3) = \pi/3$. Therefore there is a point r_i on the line segment sp_i such that $\angle r_iq_is = \pi/3$. Because $\alpha_i \leq \pi/3$, we have that $\angle sr_iq_i = \pi - (\alpha_i + \angle r_iq_is) \geq \pi - (\pi/3 + \pi/3) = \pi/3$. Therefore Invariant 2 holds for H_i.

  We now show that Invariant 2 holds for each K_i, with t as its critical vertex. Because $\angle q_irt = \gamma/2 < \pi/3$ and $\beta = \pi/3$, we have that $\angle p_iq_it \geq \angle rq_1t = \pi - (\beta + \angle srp) > \pi - (\pi/3 + \pi/3) = \pi/3$. Hence, there is a point r_i' on the line segment p_it such that $\angle r_i'q_it = \pi/3$. Let $\beta_i = \angle r_i'tq_i$. From Lemma 15, $\beta_i \geq (\alpha_i/4\pi^2\alpha)\beta$. Since $\beta = \pi/3$ and $\alpha_i = (2d_{H_i}(s) - 1)\alpha/(2d_G(s) - 1)$,

$$\beta_i \geq \frac{1}{4\pi^2} \frac{2d_{H_i}(s) - 1}{2d_G(s) - 1} \frac{\pi}{3}$$

  Since $d_G(s) \leq d$, $d_{H_i}(s) \geq 1$ and $d_{K_i}(t) \leq d$,

$$\beta_i \geq \frac{1}{24\pi d} \geq \frac{2d_{K_i}(t) - 1}{48\pi d^2}$$

  Because $\beta_i \leq \beta = \pi/3$, we have that $\angle tr_iq_i = \pi - (\beta_i + \angle r_iq_it) \geq \pi - (\pi/3 + \pi/3) = \pi/3$. Therefore Invariant 2 holds for K_i as well.

  We now show that Invariant 2 holds for G_1, when G_1 consists of a single edge only (see Figures 4.15 and 4.13(b)). Notice that $d_{G_1}(s) = d_{G_1}(t) = 1$.

$$\angle q_2st \;=\; \alpha - \sum_{2 \leq i \leq m} (2d_{H_i}(u) - 1)\alpha/(2d_G(u) - 1) + \sum_{2 \leq i \leq m-1} \alpha/(2d_G(u) - 1)$$

$$
\begin{aligned}
&= \alpha - (2d_G(u) - 2d_{G_1}(s) - (m-1))\alpha/(2d_G(u)-1) + (m-2)\alpha/(2d_G(u)-1) \\
&= \alpha - (2d_G(u) - 2d_{G_1}(s) - 1)\alpha/(2d_G(u)-1) \\
&= 2d_{G_1}(u)\alpha/(2d_G(u)-1) \\
&> (2d_{G_1}(u) - 1)\alpha/(2d_G(u)-1)
\end{aligned}
$$

From Lemma 15, $\angle q_2ts \geq (\angle q_2st/4\pi^2\alpha)\beta$. Since $\beta = \pi/3$ and $\angle q_2st > \alpha/(2d_G(u) - 1) > \alpha/2d$, we have that $\angle q_2ts > 1/24\pi d^2 > (2d_{G_1}(t) - 1)/48\pi d^2$. $\angle sq_2t \geq \angle srt = \gamma \geq \pi/3$. Hence Invariant 2 holds for G_1 also, where either s or t is the critical vertex of G_1.

□

Fig. 4.16 shows a drawing, with angular resolution $2 \cdot 60°/(2 \cdot 6 - 1) = 11°$, of the series-parallel graph G of Fig. 4.11(a), constructed by our algorithm. Notice that G has degree 6.

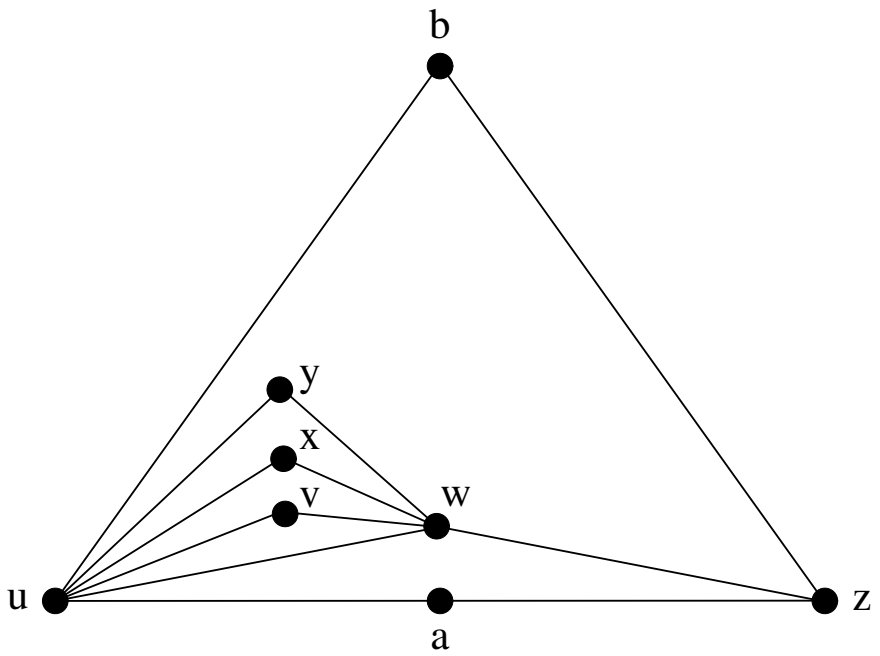

Figure 4.16: A drawing, with angular resolution $2 \cdot 60°/(2 \cdot 6 - 1) = 11°$, of the series-parallel graph G of Fig. 4.11(a), constructed by the algorithm of Section 4.5.3. Notice that G has degree 6.

4.5.4 Outerplanar Graphs

In this section we show that each maximal outerplanar graph with degree d admits a planar straight-line drawing with angular resolution $\pi/(d-1)$. Moreover, such a drawing can be constructed in linear time. This is stated formally in Theorem 14. Because the

proof of Theorem 14 is constructive, it yields a simple algorithm for constructing such a drawing.

An *outerplanar* graph is one that has an embedding in which each vertex is on a single face, called its *outer face* (see Fig. 4.22(a)). A *maximal* outerplanar graph is one to which no edge can be added without making the graph non-outerplanar (for an example, see Fig. 4.22(a)). Let G be an embedded maximal outer planar graph. Let f be the outer face of G. It is easy to see that all the faces of G, except f, are triangles, i.e., consist of only three edges each. Also notice that G is also biconnected, i.e., there is no vertex, called a *cut-vertex*, whose removal divides G into two or more graphs that do not have any vertex in common.

The proof of Theorem 14 uses certain properties of breadth-first search. Before giving the proof, we discuss these properties. Let G be an embedded maximal outerplanar graph, i.e., along with G we are also provided with an embedding of G. Assume that the outer face of G is its external face in the embedding. Recall that in a breadth-first search done starting at a vertex r, we maintain a queue Q of *ready* vertices. Q is initialized by inserting r in it. In each iterative step of the search, we *visit* each and every vertex present in Q, and insert their not-already-visited neighbors into Q (thereby making them ready), so that they are visited in the next iterative step. Thus for each edge (u, v) of the graph, u and v are visited either in the same iterative step or in consecutive iterative steps. A *layering* $\mathrm{L}(G, r)$ of G is a mapping l of the vertices of G to non-negative integers, such that $l(u) = i$ if and only if u is visited in the i^{th} iterative step of a breadth first search of G done starting at Vertex r. r is called the *root* of G in Layering $\mathcal{L}(G, r)$. u *belongs to* (or is *on*) layer i if $l(u) = i$. Notice that $l(r) = 0$. An edge (u, v) *belongs to* (or is *on*) layer i if both u and v belong to layer i. A path p *belongs to* (or is *on*) layer i if all its edges belong to layer i. A path $p : uw_1w_2 \ldots w_mv$ belonging to layer i is a *maximal path* of layer i if the only edge incident on u that belongs to layer i is (u, w_1), and the only edge incident on v that belongs to layer i is (v, w_m). v is an *immediate successor* of u (and conversely, u is the *immediate predecessor* of v) if $l(v) = l(u) + 1$. v is a *common* immediate successor of vertices u and w if v is an immediate successor of both u and w.

Suppose u and v be two vertices of G, belonging to the same layer i in $\mathcal{L}(G, r)$. u is to the *left* (to the *right*, otherwise) v if u precedes v in the clockwise order of vertices on the outer-face, assuming that r is the first vertex in the order. We say that u and v are *consecutive* vertices if there is no vertex between them in the left to right order of the vertices on Layer i. u is *immediately left* (and conversely v is *immediately right*) of v if u is to the left of v, and u and v are consecutive vertices.

Lemma 17 *Let G be an embedded graph maximal outerplanar graph with at least 3 vertices, such that its outer face is its external face in the embedding. Let r be a vertex of G. Let $L(G, r)$ be a layering of G with root r. Each layer i of $L(G, r)$ has the following properties:*

Property P1: *Let u and v be two vertices on layer i. If u and v are neighbors in G, or have a common immediate successor, then u and v are consecutive vertices.*

Property P2: *Let u and v be two vertices on layer i. Then u and v have at most one common neighbor.*

Property P3: *Each vertex on layer i is an immediate successor of at most two vertices.*

Property P4: *If two vertices u and v on layer i have a common immediate successor then they are neighbors in G. Notice that because of Property* **P1**, *u and v are also consecutive.*

Property P5: *Let p be a maximal path belonging to layer i. If $i \geq 2$, then p consists of the immediate successors of two and only two vertices, say u and v, and p also contains the common immediate successor of u and v.*

Proof: Let f be the outer face of G. Denote by Ψ, the embedding of G. Properties **P1**, **P2**, **P3** and **P4** are true for layer 0. Now assume that $i \geq 1$. We prove one by one, that each property holds for layer i. Our proof for each property, except **P4**, has the following flavor: Because G is an outerplanar graph, each vertex v of G is on f. f is the external face of G in Ψ. Hence, there is no cycle c in G, such that v is in the interior of c. For each property P, we show that if for contradiction, we assume that P does not hold for layer i, then there is a vertex v and a cycle c, such that v is in the interior of c. Therefore, Property P holds for layer i.

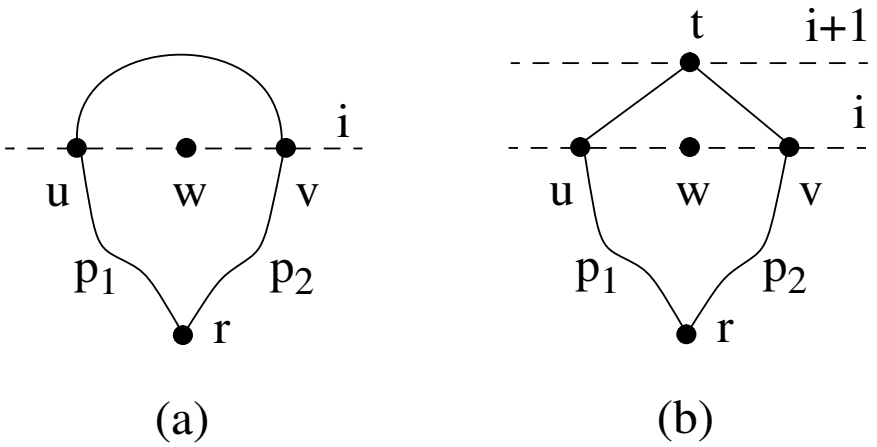

Figure 4.17: Proof of Property **P1**: (a) If there is an edge (u, v); (b) If u and v have a common immediate neighbor t

Property P1: Let u and v be two non-consecutive vertices on layer i. Let w be a vertex on layer i which is to the right of u and to the left of v (see Fig. 4.17). Because G is outerplanar, each vertex of G is on Face f. There are paths $p_1 : r \rightsquigarrow u$, $p_2 : r \rightsquigarrow v$, and $p_3 : r \rightsquigarrow w$ consisting of vertices belonging to levels 0 to $i - 1$, in addition to u, v and w respectively.

Now, suppose for contradiction that u and v are neighbors, i.e., there is an edge (u, v) in G. Then, as shown in Fig. 4.17(a), w is in the interior of cycle $r \rightsquigarrow uv \rightsquigarrow r$, a contradiction. Hence u and v are consecutive vertices.

Again, suppose for contradiction that u and v have a common immediate successor t. Then, as shown in Fig. 4.17(b), vertex w is in the interior of the cycle $r \rightsquigarrow utv \rightsquigarrow r$, and hence u, v and w, a contradiction. Hence u and v are consecutive vertices.

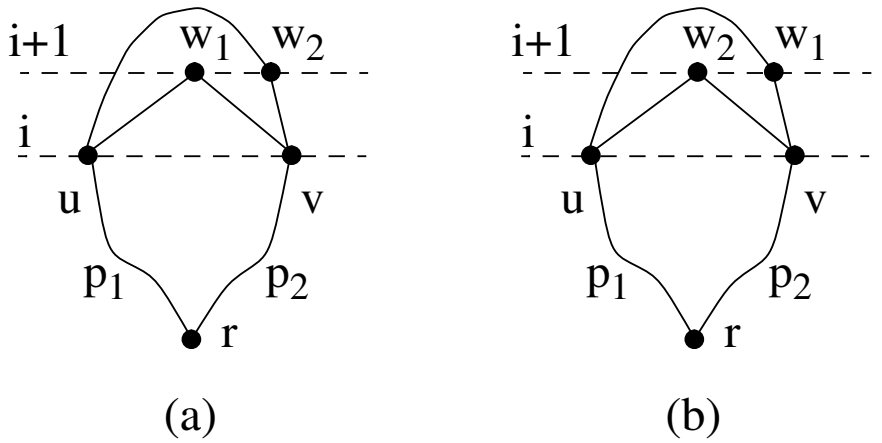

Figure 4.18: Proof of Property **P2**: (a) w_1 is in the interior of the cycle $r \rightsquigarrow uw_2v \rightsquigarrow r$ (b) w_2 is in the interior of the cycle $r \rightsquigarrow uw_1v \rightsquigarrow r$

Property P2: Let u and v be two vertices on layer i. There are paths $p_1 : r \rightsquigarrow u$ and $p_2 : r \rightsquigarrow v$, consisting of vertices belonging to levels 0 to $i-1$, in addition to u and v respectively. Suppose for contradiction, u and v have two or more common immediate successors $w_1, w_2, \ldots, w_m$, where $m \geq 2$. Then, either w_1 is in the interior the cycle $r \rightsquigarrow uw_2v \rightsquigarrow r$ (see Fig. 4.18(a)), a contradiction or w_2 is in the interior of the cycle $r \rightsquigarrow uw_1v \rightsquigarrow r$(see Fig. 4.18($b$)), again a contradiction. Hence, u and v have at most one common immediate successor.

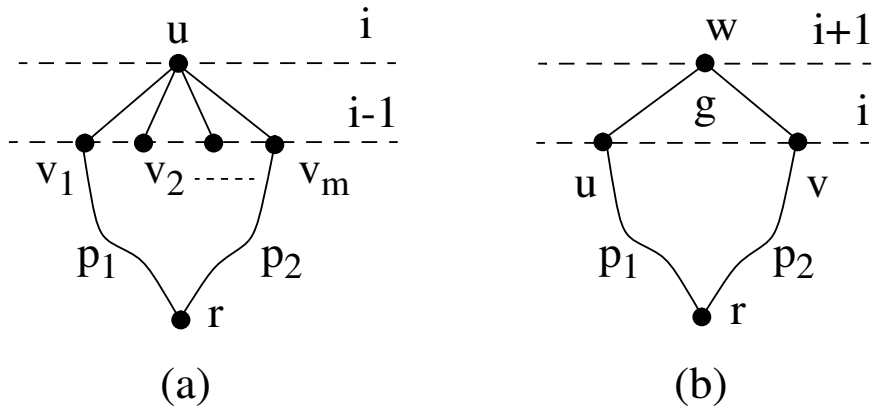

Figure 4.19: Proof of (a) Property **P3**; (b) Property **P4**

Property P3: let u be a vertex on layer i. Suppose for contradiction, it is an immediate successor of three or more vertices, $v_1, v_2, v_3, \ldots, v_m$, where $m \geq 3$ and v_k is to the left of v_{k+1}. There are paths $p_1 : r \rightsquigarrow v_1$ and $p_2 : r \rightsquigarrow v_m$ on Face f, consisting of vertices belonging to levels 0 to $i-2$, in addition to v_1 and v_m respectively. v_2 is in the interior of the cycle $r \rightsquigarrow v_1 u v_m \rightsquigarrow r$ (see Fig. 4.19(a)), a contradiction. Therefore, u has at most two immediate predecessors.

Property P4: Let u and v be two vertices on layer i. Let w be a common immediate successor of u and v. There are paths $p_1 : r \rightsquigarrow u$ and $p_2 : r \rightsquigarrow v$ on Face f, consisting of vertices belonging to levels 0 to $i-1$, in addition to u and v respectively. There exists an internal face g of G such that u, v and w are on g, and the cycle $r \rightsquigarrow uwv \rightsquigarrow r$ encloses g (see Fig. 4.19(b)). Because G is a maximal graph, each face of G, other than f, is a triangle. Hence, g is also a triangle. Therefore, there is an edge (u, v) in G. Hence, u and v are neighbors in G.

Property P5: Let $i \geq 2$. Let $p : a_1 a_2 \ldots a_m$ be a maximal path on layer i, where a_k is to the left of a_{k+1} (see Fig. 4.20).

First, suppose for contradiction that all the a_j's are immediate successors of a single vertex w. We claim that there is only one internal face of G that contains both w and a_1, namely the face that consists of the vertices w, a_1 and a_2. Suppose for contradiction, there is another internal face g that contains both w and a_1. Because G is maximal outerplanar, g is a triangle. But, because for each edge (u, v) of G, u and v, belong to either same layers or adjacent layers, g contains an edge $e \neq (a_1, a_2)$ incident on a_1, that belongs to layer i. This contradicts the assumption that p is a maximal path. Therefore, there is only one internal face of G that contains both w and a_1. From a similar reasoning, there is only one internal face that contains both w and a_m. Therefore, w is a cut-vertex of G (whose removal divides G into two or more graphs, one containing r, and the others containing the a_j's), which contradicts the biconnectivity of G. Hence all the a_j's can not be immediate successors of a single vertex.

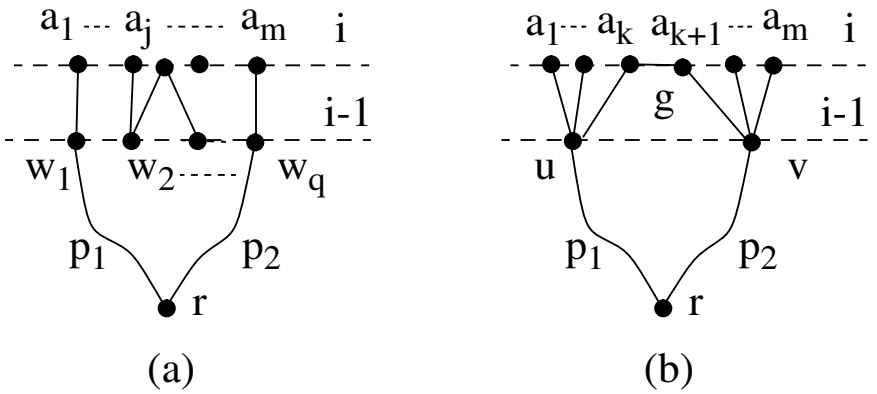

Figure 4.20: Proof of Property **P5**: (a) Vertices of path $p : a_1 a_2 \ldots a_m$ are immediate successors of q vertices $w_1 w_2 \ldots w_q$, where $q \geq 3$; (b) p does not contain the common immediate successor of u and v.

Now, suppose for contradiction that the vertices of p are immediate successors of q vertices $w_1 w_2 \ldots w_q$ on layer $i-1$, where $q \geq 3$. There are paths $p_1 : r \rightsquigarrow w_1$ and $p_2 : r \rightsquigarrow w_q$, consisting of vertices belonging to levels 0 to $i-2$,, in addition to w_1 and w_q respectively. w_2 is in the interior of the cycle $r \rightsquigarrow w_1 a_1 a_2 a_m w_q \rightsquigarrow r$ (see Fig. 4.20(a)), a contradiction.

Therefore, p consists of the immediate successors of two and only two vertices, say u and v. Now we show that p also contains the common immediate successor of u and v. Suppose for contradiction, this is not true. Assume without loss of generality that u is to the left of v. Let vertices $a_1 a_2 \ldots a_k$, where $1 \leq k \leq m-1$ be immediate successors of u only. Vertex a_{k+1} is immediate successor of v only. As shown in Fig. 4.20(b), there is an internal face g of G, that consists of at least four vertices, namely, u, a_k, a_{k+1}, and v, and hence is not a triangle, a contradiction. Hence, u and v have a common immediate successor w, and p contains w. Notice that from Properties **P1** and **P4**, u and v are consecutive as well as neighbors.

□

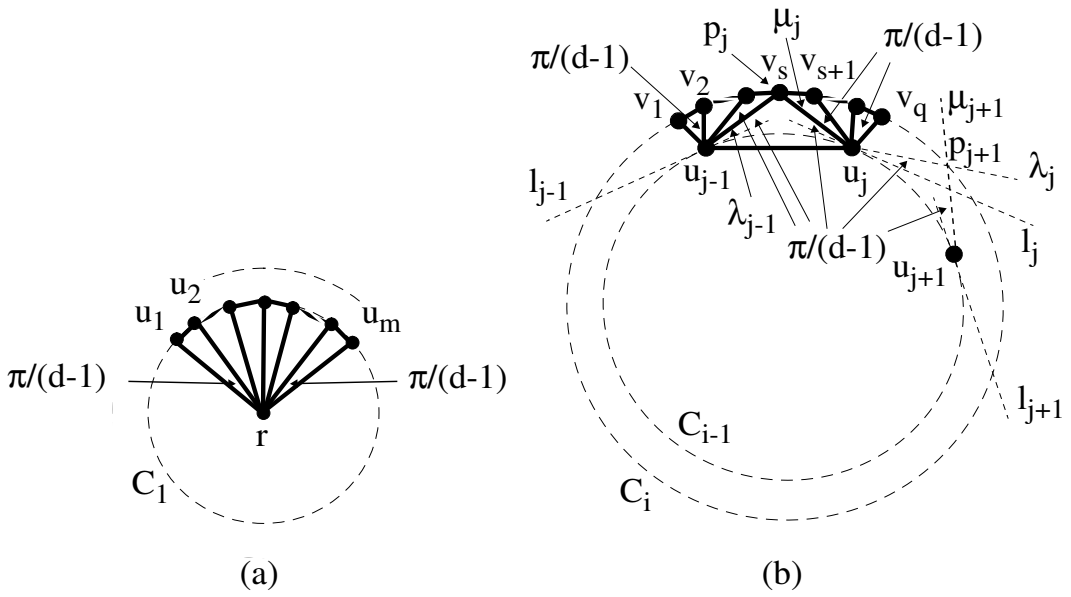

Figure 4.21: Drawing an outerplanar graph G: (a) Placing on Circle C_1, the vertices belonging to layer 1; (b) Placing on Circle C_i, the vertices of a maximal path $p : v_1 v_2 \ldots v_q$ belonging to layer i

Theorem 14 *A maximal outerplanar graph with degree d and n vertices admits a planar straight-line drawing with angular resolution $\pi/(d-1)$ that can be constructed in $O(n)$-time.*

Proof: Let G be a maximal outerplanar graph with degree d and n vertices. Assume without loss of generality that G is equipped with an embedding in which its outer face is its external face. Let r be a vertex of G. Let $\mathrm{L}(G, r)$ be a layering of G with root r.

A planar straight-line drawing D of G, with angular resolution $\pi/(d-1)$ can be constructed by the simple procedure `Drawoutpl` given below, that places the vertices of G on a set of concentric circle such that vertices belonging to the same layer gets placed on the same circle. The correctness of Procedure `Drawoutpl` is based on the fact that it maintains the following invariant while placing the vertices:

Invariant 2: For each layer i, all the edges belonging to layer i have equal lengths.

`Drawoutpl(G)`

Let C_1 be a circle, with non-zero radius. Place r at the center of C_1. Invariant 2 holds trivially for layer 0 (which contains only one vertex, namely r). Let $u_1 u_2 \ldots u_m$ be the left-to-right order of the vertices on layer 1. Notice that each u_j is a neighbor of r. Place vertices $u_1, u_2, \ldots, u_m$ at equidistant positions on C_1 such that for each u_j, angle $\angle u_{j-1} r u_j$ is $\pi/(d-1)$ (see Fig. 4.21(a)). From Propery P1, it is easy to see that Invariant 2 holds for layer 1.

Now, suppose we have already placed the vertices from layers 1 to $i-1$, where $i \geq 2$, on concentric circles $C_1, C_2, \ldots, C_{i-1}$. Let $u_1, u_2, \ldots, u_m$ be the left-to-right order of the vertices on C_{i-1} (see Fig. 4.21(b)). Let l_j be the tangent of Circle C_{i-1} at u_j. Let λ_j and μ_j be two lines passing through u_j, that make angles $\pi/(d-1)$ and $-\pi/(d-1)$ respectively with l_j. Let p_j be the point of intersection of lines λ_{j-1} and μ_j. Let E be the set of edges belonging to layer $i-1$. Because layer i has at least one vertex, and hence at least one maximal path, From Property **5**, layer $i-1$ has at least two vertices a and b which have a common immediate successor. From Property **4**, there is an edge (a, b) in G. Hence, E is non-empty. Also, from Property **1**, for each edge $e = (a, b)$ in E, a and b are consecutive vertices. Let J be a set defined as $J = \{j | (u_{j-1}, u_j) \in E\}$. Because of Invariant 2, there is a unique circle C_i concentric to C_{i-1} that passes through each p_j, for every $j \in J$. Place the vertices belonging to layer i on Circle C_i as follows: For each u_j, if u_j and u_{j-1} have a common immediate successor w(which if exists, is unique from Property **P2**), then place w at the point p_j (see Fig. 4.21(b), where $w = v_s$). From Property **4**, $j \in J$. Hence, w gets placed on Circle C_i. It follows from Property **P1** that, if a vertex w on layer i has two immediate predecessors a and b, then a and b are consecutive vertices. Hence, from Property **P3**, the vertices of layer i that remain to be placed are the ones with only one immediate predecessor. Let $p = v_1, v_2, \ldots, v_q$ be a maximal path on layer i such that v_k is to the right of v_{k-1} (see Fig. 4.21(b)). Let u_{j-1} and u_j be the immediate predecessors of the vertices in p. From Property **P5**, u_{j-1} and u_j have a common immediate successor w, and w is on Path p. Let $w = v_s$, where $1 \leq s \leq q$. Therefore, vertices $v_1, v_2, \ldots, v_s$ are immediate successors of u_{j-1} and vertices $v_s, v_{s+1}, \ldots, v_q$ are immediate successors of u_j. Notice that we have already placed w at point p_j. As shown in Fig. 4.21(b), place each

v_k such that $\angle v_{k-1}u_{j-1}v_k = \pi/(d-1)$, for $k \leq s$, and $\angle v_k u_j v_{k+1} = \pi/(d-1)$, for $k > s$.

It is easy to see that the angular resolution of the drawing constructed by Procedure `Drawoutpl` is $\pi/(d-1)$. As for the time-complexity, $\mathrm{L}(G, r)$ can be easily constructed in $O(n)$-time. Procedure `Drawoutpl` executes in $O(n)$-time. Therefore, the total time-complexity is $O(n)$. □

Fig. 4.22(b) shows a drawing D, with angular resolution $180°/(5-1) = 45°$, of the maximal outerplanar graph G of Fig. 4.22(a), constructed by our algorithm. Notice that G has degree 5. In D, vertices a, h, i and k are placed on the same circle, and vertices b, c, e and j are placed on the same circle.

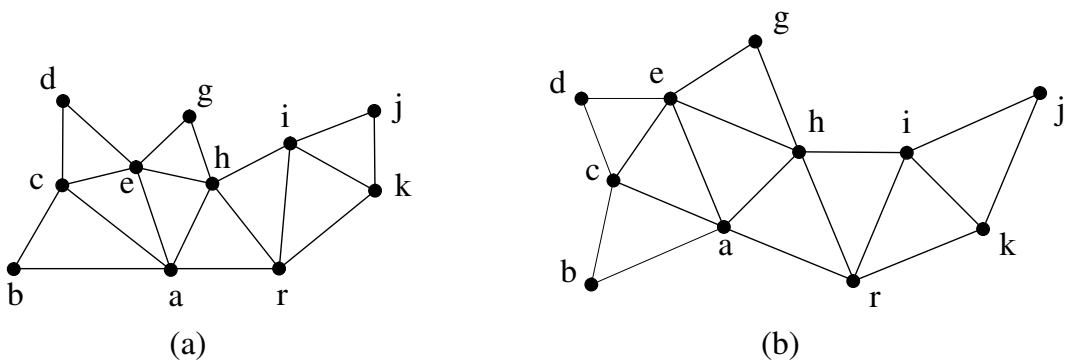

Figure 4.22: (a) A maximal outerplanar graph G with degree 5; (b) A drawing of G, with angular resolution $180°/(5-1) = 45°$, constructed by the algorithm of Theorem 14.

4.6 Conclusion

We have studied the problem of angular resolution and presented several new results. We have proved the first non-trivial upper bound on the angular resolution of planar straight-line drawings. We have shown a continuous tradeoff between the area, and the angular resolution of planar straight-line drawings. We have given linear-time algorithms for constructing planar straight-line drawings with high angular resolution of various classes of graphs, such as series-parallel graphs, maximal outerplanar graphs, and nested-star graphs. Our results are obtained by new techniques that make extensive use of geometrical constructions.

Several interesting problems are still open:

- Find a tighter upper bound on the angular resolution of planar graphs.
- Notice that the graphs that have been shown to have angular resolution $O(\sqrt{\log d/d^3})$ are nested-star graphs, and hence can be drawn with angular resolution $\Omega(1/d^2)$ using the algorithm of Section 4.5.2. Find matching upper and lower bounds for these graphs.

- Prove or disprove whether every planar graph has angular resolution that is a polynomial in its degree. Notice that the result of [83] shows that every planar graph has angular resolution that is exponential in its degree.
- Find tighter lower bound for the area-requirement of straight-line drawings, with large angular resolution, of a planar graph. Also find matching upper and lower bounds for the area-requirement.
- Devise algorithms that are parameterized for area and angular resolution, i.e., they construct drawings with large or small area, depending upon the desired angular resolution.
- Notice that the area can be sometimes be reduced by allowing some edges to have bends. Study the trade-off between area, angular resolution and number of bends.

Chapter 5

Angle Graphs

5.1 Introduction

An *angle graph* is a graph with a fixed cyclic order of edges around each vertex and an angle (between 0° and 360°) specified for every pair of consecutive edges incident on each vertex such that the sum of angles around every vertex is 360°. Fig. 5.1(a, b) show two angle graphs, with angles 15°, 30°, 45°, 120° and 300° between consecutive edges incident on a vertex. A *rectilinear* angle graph is an angle graph in which each angle is a multiple of 90°.

Many well known graph drawing algorithms, while drawing a graph, use rectilinear angle graphs in an intermediate stage. For example, Tamassia's bend-minimization algorithm [99] constructs a planar drawing of a planar orthogonal graph (each vertex has degree at most four) by first generating a rectilinear angle graph referred to as an *orthogonal representation*, and then drawing it. Similarly the algorithm by Tamassia and Tollis [101] that converts a visibility representation of a graph into a rectilinear planar drawing uses an orthogonal representation at an intermediate stage. Drawings of rectilinear graphs also find application in a variety of areas such as VLSI design [75, 109], architectural floor plan layout [77] and data base system design [4]. In fact a recent experimental study [62, 61] has shown that Tamassia's bend-minimization algorithm gives a more aesthetic drawing in practice then some other well known algorithms [18, 27, 19, 113].

The success of rectilinear angle graphs in drawing graphs raises the natural question whether it is possible to use general angle graphs for constructing high quality planar drawings. Given a planar graph, Tamassia's bend minimization algorithm can be used directly to construct angle graphs whose angles are multiples of a preassigned angle. Hence a good characterization of planar angle graphs can allow us to use the bend minimization algorithm to construct aesthetically pleasing drawings of any planar graph, not just orthogonal planar graphs.

A recent trend in the area of graph drawing is towards developing systems which allow the user to specify arbitrary constraints on the positioning of vertices and edges in the drawings. See for example, [23, 24, 41, 65]. Since the user may specify angle constraints, a study of angle graphs is important for such systems.

In a straight-line drawing (edges drawn as straight lines) of a graph, it is difficult to distinguish between two edges incident on the same vertex, if the angle between them is very small. Hence, it is very important in a straight-line drawing of a graph, that the angles between the consecutive edges incident on each vertex be large. Maximizing angles is also important in other applications, such as optical communication where it is very difficult to transmit and receive signal at a very tight angle [48]. The notion of large angles between consecutive edges incident on a vertex is formalized by defining *angular resolution* of a graph [48, 58, 83]. The *angular resolution* of a straight line drawing of a graph is the smallest angle in the drawing between any two consecutive edges incident on the same vertex. For example, the angular resolution of the drawing shown in Fig. 5.1(b) is 30°. *Angular resolution* of a graph is the maximum angular resolution over all the straight-line drawings of the graph.

Studying angle graphs can provide important clues in constructing drawings with large angular resolution. For example we use our result that testing an angle graph for planarity is NP-hard, to show that, given a triconnected planar graph G and an angle α, determining whether G admits a planar straight line drawing with angular resolution at least α is NP-hard. Also see [72] for related results on drawing graphs on a hexagonal grid. Angles have been found to be useful in characterizing planar graphs. In a recent paper [38], Dibattista and Vismara characterize many important constructions such as Delauny drawing, disc packing etc. of a triangulated graph using its angles. The study of angle graphs is also of theoretical interest in itself because they require an extensive use of plane geometry in their analysis. Most of the known graph drawing algorithms use graph theoretic concepts such as orientation, coloring, flow etc. Recent results [58] however shows that plane geometry is a useful tool by successfully applying geometric techniques in deriving a number of results on planar drawings with large angular resolution. A study of angle graphs can provide an invaluable insight into the use of plane geometry for graph drawing algorithms and stimulate research in this relatively unexplored area. Most of the results and techniques presented in this chapter can also be found in [51].

5.1.1 Some Definitions

Let A be an angle graph. The graph isomorphic to A if we drop the angle constraints is called the *underlying graph* of A. Let H be the underlying graph of A. Two edges incident on a vertex v are called *neighboring* edges if they appear consecutively in the cyclic order of edges incident on v. A is called a a *rectilinear angle graph* if each vertex of A has at most four neighbors and each angle is a multiple of 90°. A is an *angle-cycle* (*outerplanar* angle graph, *series-parallel* angle graph, respectively) if H is a simple cycle (outerplanar graph, series-parallel graph, respectively). We *break* an edge (u, v) of A by introducing a new vertex w and replacing (u, v) by two edges (u, w) and (w, v). The angle between the edges incident on w is 180°. A *subdivision* of A is an angle graph that we get by breaking some edges of A.

A *drawing* of A is a mapping of its vertices to points in the plane and its edges to straight-lines (of non-zero lengths) joining their end-points so that the angles in the drawing between any two neighboring edges of A is equal to the angle specified for that pair in A. A *planar* drawing of A is a drawing without any edge crossings. It is possible

that an angle graph may not admit a drawing at all. Fig 5.1(a) shows such an angle graph. Fig 5.1(b), on the other hand, shows an angle graph that admit a drawing. A is called a *consistent* angle graph if it admits a drawing. A is called a *planar* angle graph if it admits a planar drawing.

Let D be a drawing of A. We denote the x- and y- coordinates of a vertex v of A in D by $x(v)$ and $y(v)$ respectively. Let u and v be two vertices of A. We denote by $d(u, v)$, the (metric) distance between two vertices u and v in D. The *horizontal distance* between u and v is equal to $|x(u) - x(v)|$. The *vertical distance* between u and v is equal to $|y(u) - y(v)|$. AB denotes the line-segment joining A and B, and also its magnitude. $\vec{AB}$ denotes the directed line-segment from A to B. $\angle ABC$ denotes the angle between the line-segments AB and BC, and also its magnitude. In this chapter, in each figure showing an angle graph, we label each angle by its magnitude. Sometimes in a figure, to reduce visual complexity, we do not label all the angles. The unlabeled angles can be deduced easily either from the context or from the labeled angles by using the fact that the sum of angles around each vertex is 360°.

Let G be a graph. A vertex u of G is a *articulation vertex* if removing u and its incident edges divides G into two or more graphs G_1, G_2,...G_m, that do not have any vertex in common. Each G_i is a *component* of G. G is *biconnected* if it does not have any articulation vertex. Two vertices u and v of G form a *separating pair* if removing u and v and their incident edges divides G into two or more graphs that do not have any vertex in common. G is *triconnected* if it does not have any separating pair.

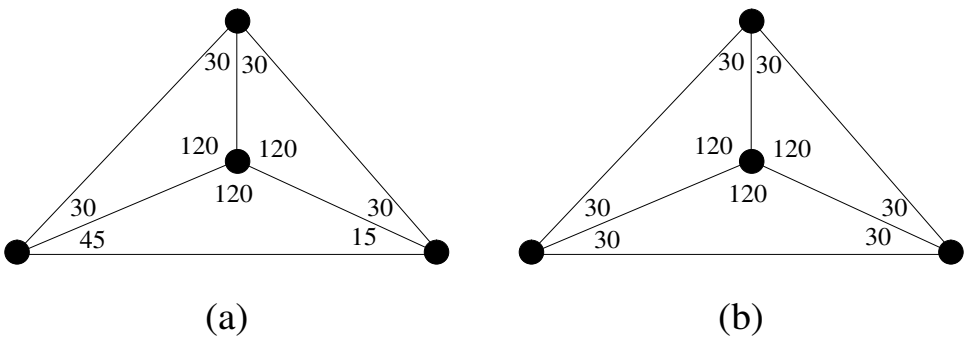

Figure 5.1: An angle graph (a) that is not consistent; (b) that is consistent.

5.1.2 Previous Work

The pioneering work in the area of angle graphs was done by Vijayan in [110]. Vijayan gave a characterization of consistent angle graphs, using a set of linear equalities and inequalities. The ensuing linear program gives a high degree polynomial time algorithm for constructing drawing of an angle graph. Some necessary conditions for an angle graph to be planar, as well as necessary and sufficient conditions for some special angle graphs such as angle cycles, angle graphs with convex faces, outer planar angle graphs and rectilinear angle graphs have also been described in [110]. Vijayan also gave some conjectures that relate the planarity of an angle graphs with the planarity of its biconnected components (see Conjectures 1 and 2 of Section 5.2). A characterization of planar angle

graphs with triangular faces by a set of non-linear equalities is given in [38]. [111] gives an $O(n)$ time algorithm for testing a rectilinear angle graph for planarity and $O(n^2)$ time algorithm for constructing a planar drawing if it is planar. Given a planar graph, [99] shows how to construct angle graphs whose angles are multiples of a preassigned angle, such that any planar drawing of this angle graph gives a planar drawing with minimum number of bends of the input graph.

5.1.3 Our Results

We give several new results concerning drawing of angle graphs. We first study the planarity of angle graphs. We disprove the conjectures of [110] by providing counter examples to them. We then show that the problem of testing a consistent angle graph for planarity is NP-hard and hence unlikely to have an efficient algorithm. Our proof uses a reduction from the well-known 3-SAT problem [50] to the planarity testing problem, i.e., we show that given an instance T of the 3-SAT problem, we can construct in polynomial time an angle graph A(T) such that T admits a satisfying truth-assignment if and only if A(T) is planar. A(T) is constructed using a gadget based approach in which we assemble smaller graphs to construct a bigger graph.

Using our NP-hardness result, we then show that given a *triconnected* planar graph and an angle α, determining whether it admits a planar straight-line drawing with angular resolution at least α is NP-hard. Again, we use a reduction from the 3-SAT problem. Namely, given an instance T of the 3-SAT problem, we first construct the aforementioned angle graph A(T), and then by adding some special graphs called *fans* to A(T) and dropping the angle constraints of A(T), we construct a new graph G(T) such that G(T) admits a planar straight-line drawing with angular resolution 5° if and only if T admits a satisfying truth-assignment. Our result strengthens a previous result of Kant [68] in which he has shown that determining whether a *biconnected* planar graph has a planar straight-line drawing with angular resolution at least α is NP-hard.

We study the problem of testing whether an angle graph whose underlying graph is a series-parallel graph is consistent or not, and provide a linear time testing-algorithm. A series-parallel graph G can either be a series or a parallel composition of two series-parallel graphs G_1 and G_2 called the *constituents* of G. Our algorithm is recursive. It first determines whether G_1 and G_2 are consistent or not, and then determines whether their drawings can be combined together to give a drawing for G. Notice that this requires maintaining information about the drawings that G_1 and G_2 admit. Since there can be infinitely many drawings, our algorithm uses an efficient encoding scheme to represent all possible drawings of a series-parallel graph H. Namely, it uses a 4-tuple called *sector(H)*, that represents the smallest geometric sector that covers all the drawings of H (assuming that the source vertex of H is always placed at the origin of the coordinate system). *Sector*(G) is computed from *sector*(G_1) and *sector*(G_2) by considering whether G is a series or a parallel composition of G_1 and G_2.

The study of area requirements of drawings of graphs has received a lot of attention (see, e.g., [6, 37, 53, 58]) and is motivated by the finite resolution of the current graph drawing technologies and circuit-area optimization criteria of VLSI layouts [12, 75, 109].

Typical resolution rules are requirement of minimum unit distance between vertices, their placement on grid points of a unit-grid etc. We study the area requirement of angle graphs and show that for every $m \geq 1$, we can construct an angle graph with $3m$ vertices that requires area $\Omega(4^m)$ for any drawing, under any resolution rule. Therefore, such an angle graph can not be drawn compactly.

Finally we consider the *multiplanarity* problem of angle graphs. A *multilayered* angle graph is an angle graph in which each edge is assigned to one of several *layers*. A *multiplanar* angle graph is a multilayered angle graph if it has a drawing in which edges assigned to the same layer do not cross. We show that very surprisingly even if we are given a bilayered rectilinear angle graph (edges assigned to only two layers and all the angles are multiples of 90°), it is NP-hard to test its biplanarity. Again, we use a reduction from the 3-SAT problem, using a gadget based approach. Our result is in sharp contrast to the linear time complexity of testing a single-layered rectilinear graph for planarity [111]. Also interesting is the fact that the corresponding problem for general graphs is very simple: A graph is multiplanar if and only if each subgraph induced by the edges assigned to the same layer is planar. In a VLSI circuit even though typically many layers are available, some are preferable to others because of better electric properties [39]. Our NP-hardness result is important in this regard.

5.2 Planarity Testing of Angle Graphs

We start this section by disproving the conjectures given in [110] by providing counter-examples to them. Some definitions first, most of which are from [110]. Let A be an angle graph whose underlying graph G has an articulation vertex v. v is also an *articulation* vertex of A. A *biconnected component* of A is a biconnected component of G with angles specified. Suppose B_1 and B_2 are two biconnected components of A with a common vertex u (u is an articulation vertex of A). We say that B_1 and B_2 *interlace* at u if the edges joining u to the vertices of B_1 and B_2 alternate between B_1 and B_2 at least four times (Fig 5.2(a) shows an example given in [110]). We say that B_1 is *directly forced inside* B_2 if all the edge joining u to the vertices of B_1 are in the interior of the face of B_2 containing u (Fig 5.2(b) shows an example given in [110]-B_2 is also directly forced inside B_1 in this figure). We say that B_1 is *forced inside* B_2 if either B_1 is directly forced inside B_2 or there is another biconnected component B_3 of A such that B_3 is forced inside B_2 and B_1 is attached to B_3 (Fig 5.2(c) shows an example given in [110]). Notice that from the definition of a subdivision (see Sec 5.1.1), the subdivision of a triangle is a polygon whose all but three angles are 180°.

Vijayan has made the following conjectures in [110]:

Conjecture 1 (Vijayan [110]) *An angle graph A that does not contain any angle-cycle which is a subdivision of a triangle is planar if and only if*

1. *A is consistent,*
2. *the faces of each biconnected component of A are consistent angle cycles,*

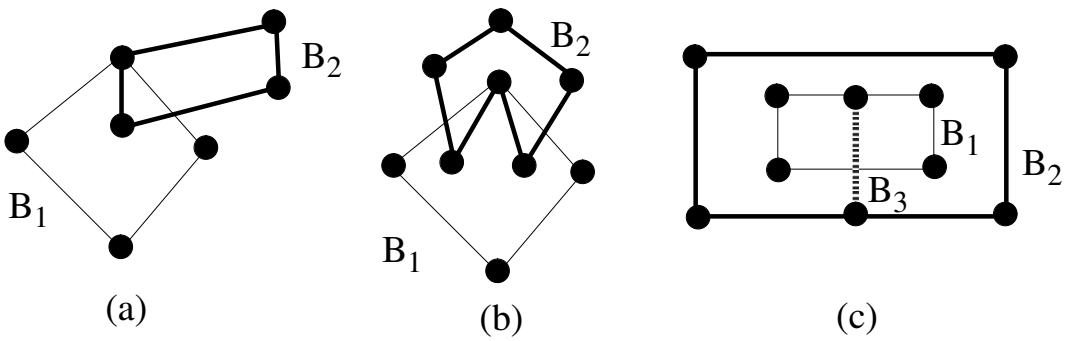

Figure 5.2: (a) Interlacing biconnected components B_1 and B_2, (b) B_1 directly forced inside B_2, and B_2 directly forced inside B_1, (c) B_1 is forced inside B_3. These figures are taken from [110].

3. *no two biconnected components of A interlace at an articulation vertex of A, and*

4. *the forced-inside relation among the biconnected components of A is a partial order.*

Conjecture 2 (Vijayan [110]) *The conditions stated in the Conjecture 1 are necessary and sufficient for angle graphs that do not contain subdivisions of triangles and whose biconnected components have convex interior faces.*

We give a Theorem 15 disproves Conjecture 2 by giving a counter example to it. This in turn disproves Conjecture 1.

Theorem 15 *There exists an angle graph A such that,*

1. *A is consistent,*

2. *A does not contain any angle-cycle that is a subdivision of a triangle,*

3. *the faces of each biconnected component of A are consistent angle cycles,*

4. *no two biconnected components of A interlace at an articulation vertex of A,*

5. *the forced-inside relation among the biconnected components of A is a partial order,*

6. *the biconnected components of A have convex interior faces, and*

7. *A is not planar.*

Proof:

Consider the angle graph A shown in Fig. 5.3(a). u is an articulation vertex of A. A has two biconnected components B_1 and B_2, with u as the articulation vertex. The angles of A are as shown in the figure. Both B_1 and B_2 have convex interior face. B_1 and B_2 do not interlace at u. B_1 is directly forced inside B_2. A does not contain any angle-cycle that is a subdivision of a triangle.

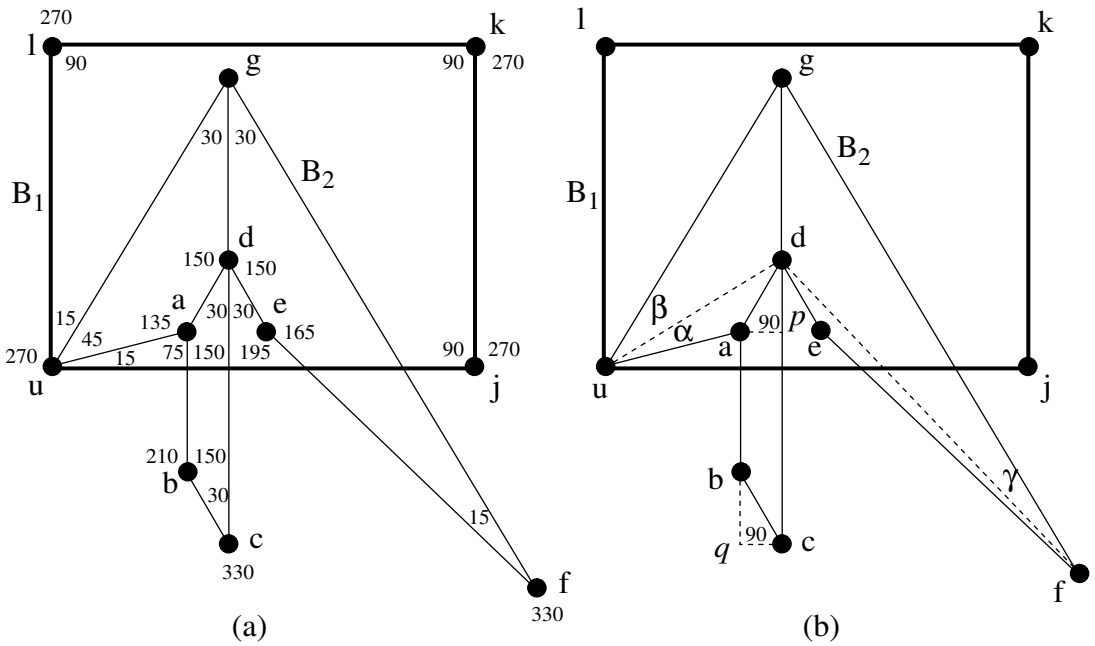

Figure 5.3: Proof of Theorem 15: (a) Angle graph A with biconnected components B_1 and B_2, and articulation vertex u. The edges of B_1 are shown as thick lines, and the edges of B_2 are shown as thin lines. (b) A drawing D of A. The dotted lines are not part of D, and are used for the proof.

We show that A does not admit any planar drawing. The intuition behind our proof is the following: We show that in a drawing of A, if the line-segment ad has a "small" length, then the line-segment fg crosses the line-segment uj otherwise if ad has a "more than small" length, then the line-segment cd crosses the line-segment uj.

We now present our proof formally. We prove that each and every drawing of A is non-planar. Let D be a drawing of A (see Fig. 5.3(b)). Let $\alpha = \angle dua$, $\beta = \angle gud$, and $\gamma = \angle dfg$. Our proof uses the following rule, which is used extensively in geometry:

Sine-rule: In a triangle ΔABC, $AB/\sin(\angle ACB) = BC/\sin(\angle BAC) = AC/\sin(\angle ABC)$.

From Δudg, and from the sine-rule, we have $dg/\sin\beta = ug/sin(\angle udg)$. Again, from Δudg, $\angle udg = 180° - \angle ugd - \beta$. Since $\angle ugd = 30°$, we have

$$dg/\sin\beta = ug/\sin(150 - \beta) \tag{5.1}$$

From Δdfg, and from the sine-rule, we have $dg/\sin\gamma = fg/sin(\angle fdg)$. Again, from Δfdg, $\angle fdg = 180° - \angle fgd - \gamma$. Since $\angle fgd = 30°$, we have

$$dg/\sin\gamma = fg/\sin(150 - \gamma) \tag{5.2}$$

From Eqs 5.1 and 5.2, we have

$$ug/fg = (\sin\gamma/\sin(150 - \gamma))(\sin(150 - \beta)/\sin\beta) \tag{5.3}$$

Because polygons $uadg$ and $defg$ are convex, we have that $\beta \leq 45°$ and $\gamma \leq 15°$. Therefore, $150 - \beta > 90°$ and $150 - \gamma > 90°$. Since $\sin\theta$ is a monotonically increasing function of θ for $0° \leq \theta \leq 90°$, and is a monotonically decreasing function of θ for $90° < \theta \leq 180°$, from Eq. 5.3, it follows that ug is greater than or equal to gf if and only if $\gamma \geq \beta$.

If $ug < gf$, then because gd is perpendicular to uj, and $\angle ugd = \angle dgf$, it follows from basic geometry that fg crosses uj, and therefore, D is non-planar.

Now assume that $ug \geq gf$. Hence, as shown earlier, $\gamma \geq \beta$. Since $\gamma \leq 15°$, we have that $\beta \leq 15°$. Therefore, $\alpha = 45° - \beta \geq 30°$. Because $\angle uad = 135°$, from Δuad we have that $\angle uda = 180° - \angle uad - \alpha = 45° - \alpha \leq 15°$. Therefore, $\angle uda < \alpha$. Since, in a triangle, the side opposite to the larger angle has a larger length, from Δuad we have that $ad > ua$.

Now consider the trapezoid $abcd$. Let p be a point on cd such that $\angle apd = 90°$. Because $\angle adc = 30°$, $pd = ad\cos 30°$. Let q be a point such that $\angle bqc = 90°$. Because polygon $abcd$ is a trapezoid, $aq = pc \geq pd = ad\cos 30°$. Since $ad > ua$, we have $aq > ua\cos 30° > ua\sin 15°$. Because aq is perpendicular to uj, and $\angle auj = 15°$, it follows from basic geometry that aq, and therefore dc, crosses uj. Hence, D is non-planar.

Therefore, A does not admit any planar drawing. □

5.2.1 NP-Hardness of Planarity Testing

In this section, we show that the problem of testing whether a given consistent angle graph admits a planar drawing or not, is NP-hard.

We reduce a well-known NP-hard problem, namely, the 3-SAT problem [50] given below, to the planarity testing problem:

> given a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables and a set $C = \{c_1, c_2, \ldots, c_m\}$ of clauses over X such that every clause has exactly three literals, is there a satisfying truth assignment for C?

Given an instance T of the 3-SAT problem, with n variables and m clauses, we construct a consistent angle graph A(T), called the *associated angle graph* of T, such that T admits a satisfying truth assignment if and only if A(T) is planar. In Section 5.2.1, we describe some gadgets that are used for constructing A(T). In Sections 5.2.1, 5.2.1 and 5.2.1 respectively, we describe three important subgraphs of A(T), namely, the *variable* subgraph, the *connection* subgraph and the *clause* subgraph. We give the full description of A(T) in Section 5.2.1.

Some Gadgets

We need the following gadgets (which are angle graphs) for our construction.

- **Horse shoe:** We use the horse shoe to represent the variables. A *horse shoe* H with *size* $r_1 \times r_2$, is shown in Fig. 5.4(a). The vertices $\{x_0, x_1, \ldots, x_{r_1}\}$ and $\{y_1, y_2, \ldots, y_{r_2+1}\}$ are called the *output* vertices of the faces F_x and F_y respectively. Vertices x_i and x_{i+1} (y_i and y_{i+1}) for odd values of i are *siblings* of each other. Edges e_x, e_y and e_t are called the *left attachment*, *right attachment* and *top edge* respectively of the horse shoe. Because each triangle of H is an equilateral triangle and F_x and F_y are parallelograms, in any planar drawing D of H,

  Observation 1: the attachments of H have same length as its top edge.

  Observation 2: if the top edge of H has unit length then at most one of $d(x_0, x_{r_1})$ and $d(y_1, y_{r_2+1})$ has length one unit or more. H is *left heavy* in D if $d(x_0, x_{r_1}) \geq 1$ (Fig. 5.4(b)) and is *right heavy* in D if $d(y_1, y_{r_2+1}) \geq 1$ (Fig. 5.4(c)), and is *balanced* otherwise.

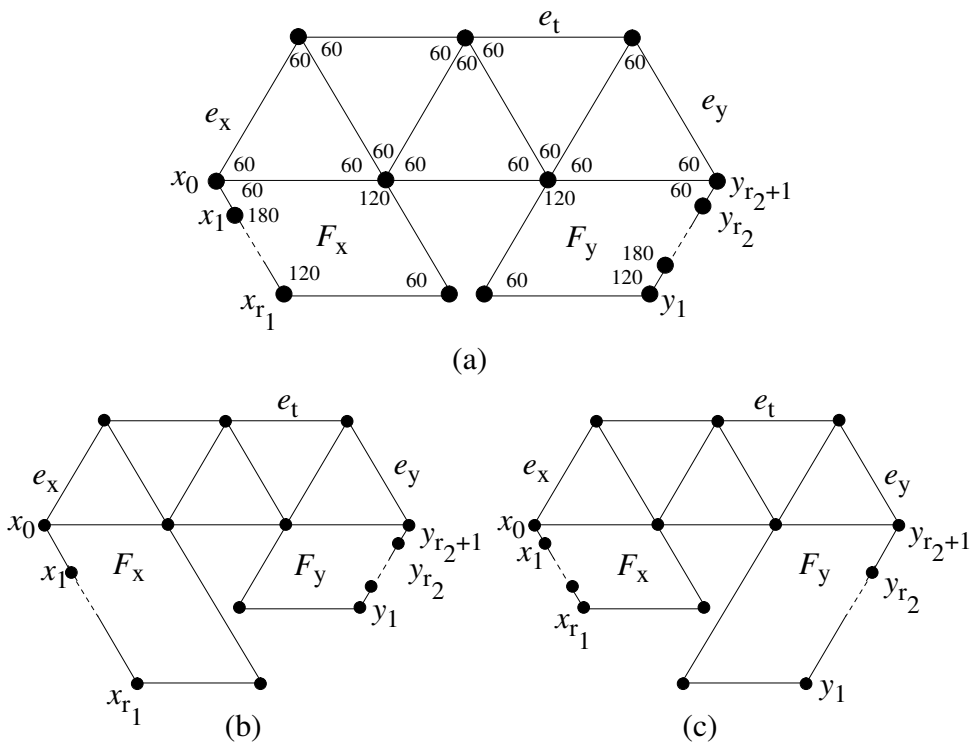

Figure 5.4: (a) A horse shoe H with size $r_1 \times r_2$; (b) A drawing of H when it is left heavy; (c) A drawing of H when it is right heavy.

- **Crown:** We use the crown to represent the clauses. Fig. 5.5 shows a *crown* C. Edges e_l and e_r are called the *left attachment* and *right attachment* respectively of C. Edge e is the *base* of C.

  Observation 3: In any drawing of C, if the base has unit length then $\sum_{0 \leq i \leq 2} d(c_{2i}, c_{2i+1}) = 5$.

- **Beam:** A *Beam* B is shown in Fig. 5.6. B consists of $24mn+1$ equilateral triangles. Edges e_l and e_r are called the *left attachment* and *right attachment* respectively of

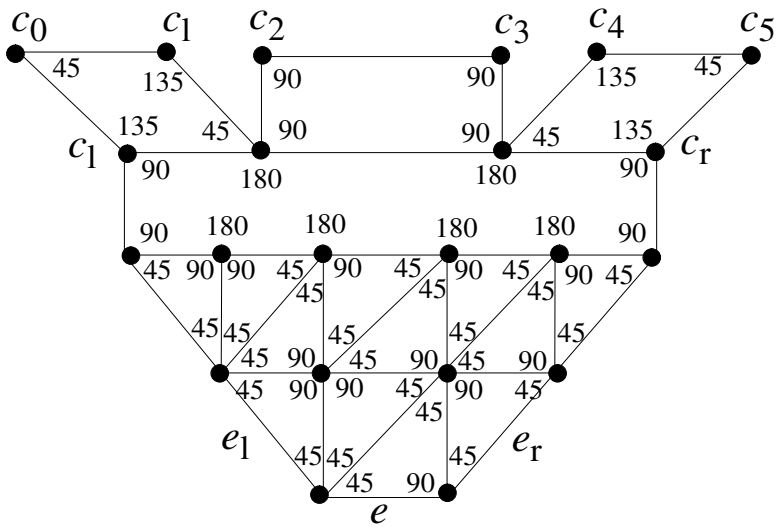

Figure 5.5: A Crown

the beam. Vertices u and v are called the *endpoints* of B. The *length* of B is equal to the distance between u and v. In any drawing of B,

Observation 4: All its edges are of equal length.

Observation 5: The length of B is $12mn$ if the length of an attachment is one unit.

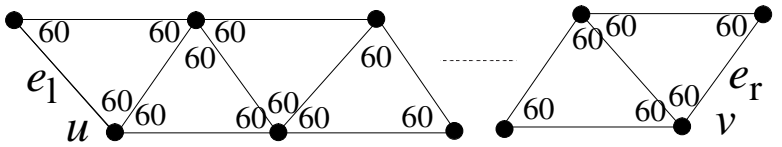

Figure 5.6: A Beam

- **Wiggle:** A *wiggle* W is shown in Fig. 5.7(b). Vertices u_1 and u_2 (v_1 and v_2) are called the *input* (*output*) vertices of the wiggle. In a drawing of W, the *width* of W is the horizontal distance between v_1 and v_2. In any drawing of W,

  Observation 6: In any drawing of W, the horizontal distance between its output vertices equals the vertical distance between its input vertices.

  Observation 7: In a drawing of W, because edges e_1 and e_2 can be assigned any arbitrary length, v_1 and v_2 can be assigned any x-coordinates, independent of the x-coordinates of u_1 and u_2.

- **Connector:** A *connector* Z is a subdivision of the angle graph shown in Fig. 5.7(c). Vertices u_1 and u_2 (v_1 and v_2) are called the *input* (*output*) vertices of Z. Vertices w_1 and w_2 are called the *first turn vertex* and *second turn vertex* of Z. In a drawing of Z, the *width* of Z is the horizontal distance between v_1 and v_2.

  Observation 8: In any drawing of Z, The horizontal distance between its output vertices equals the horizontal distance between its input vertices.

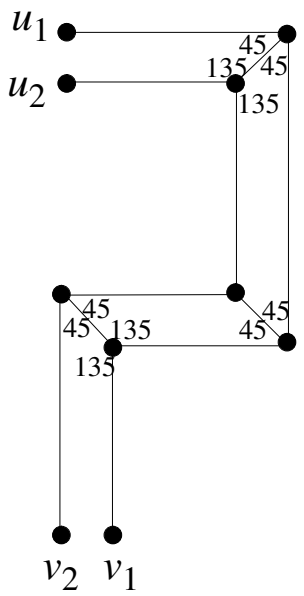

Figure 5.7: A Wiggle

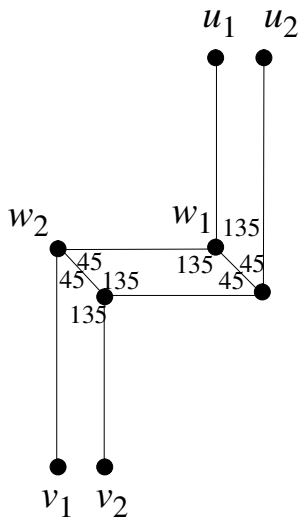

Figure 5.8: A Connector

- **Holder:** A holder H is shown in Fig. 5.7(d). Edges e_1, e_2 and e_3 are called its *upper clamp*, *lower clamp* and *shaft* respectively. In any drawing of H,

  Observation 9: The upper and lower clamps of H are of same length.

Fig 5.10(a) gives a high level view of A(T). We can identify three subgraphs of A(T), namely *variable* subgraph V_n, *connection* subgraph K and *clause* subgraph C_m. A(T) also has a holder R and some wiggles and connectors, that "connect" these three subgraphs.

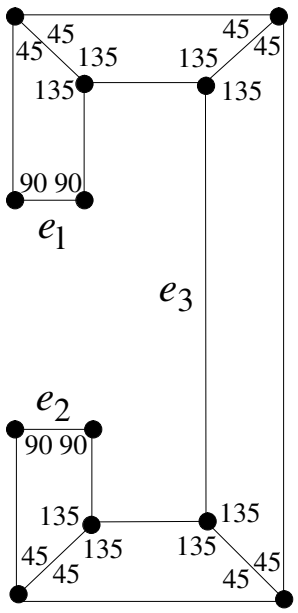

Figure 5.9: A Holder

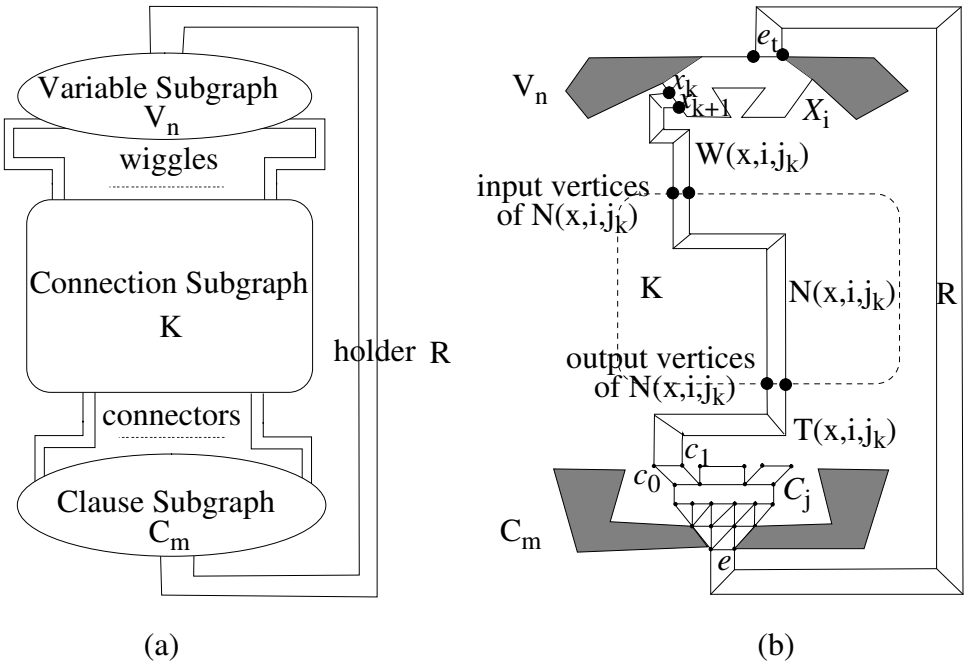

Figure 5.10: (*a*) A high level view of Angle Graph A(T); (*b*) Constructing A(T).

The Variable Subgraph

Denote by $occur(l_i)$, where $l = x$ or $l = \bar{x}$, the number of clauses in which l_i occurs. The variable subgraph V_n of A(T) (recall that n is the number of variables in T) is constructed recursively as follows:

- V_1 is a horse shoe X_1 with size $2occur(x_1) \times 2occur(\bar{x_1})$.
- V_i is constructed from V_{i-1} by identifying the right attachment of the horse shoe X_{i-1} (present in V_{i-1}) with the left attachment of the a beam B and identifying the right attachment of B with the left attachment of a horse shoe X_i with size $2occur(x_i) \times 2occur(\bar{x_i})$ (see Fig 5.11(a)).

We say that horse shoe X_i *represents* variable x_i.

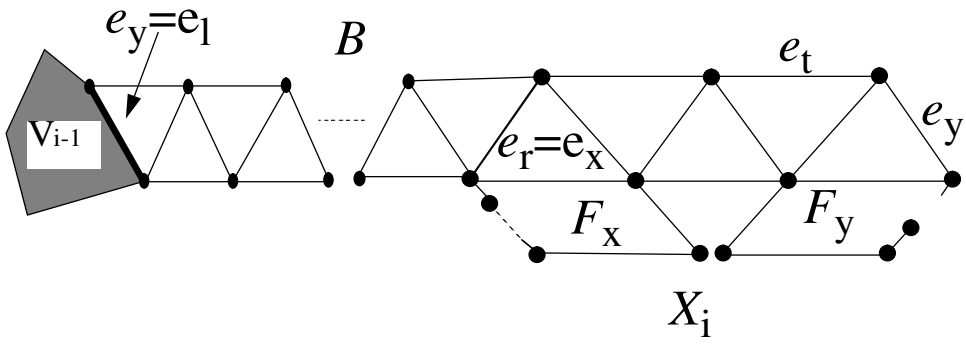

Figure 5.11: Constructing Variable Subgraph V_i from Variable Subgraph V_{i-1}

The Clause Subgraph

The clause subgraph S_m of A(T) (recall that m is the number of clauses in T) is constructed recursively as follows:

- W_1 is a crown C_1.
- W_j is constructed from W_{j-1} (see Fig 5.11(b)) by joining an endpoint u of a beam B with an endpoint of the crown C_{j-1} using an edge e', and joining the other endpoint v of B an endpoint of a crown C_j using an edge e''.

We say that crown C_j *represents* clause c_j.

Connection subgraph

The *connection* subgraph K of A(T) is constructed in following steps:

1. Let H be a bipartite graph whose vertices can be partitioned into two sets P and Q. There is a vertex $x_{i,j}$ in P if and only if literal x_i occurs in clause c_j. There is a vertex $y_{i,j}$ in P if and only if literal $\bar{x}_i$ occurs in clause c_j. $x_{i,j}$ ($y_{i,j}$) is called an *image* of the literal x_i ($\bar{x}_i$). Q has a vertex $c_{j,2i-1}$ if literal x_i occurs in clause c_j. Q has a vertex $c_{j,2i}$ if literal $\bar{x}_i$ occurs in clause c_j. $c_{j,k}$, where $1 \leq k \leq 2n$, is an *image* of clause c_j. There is an edge in H between $x_{i,j}$ and $c_{j,2i-1}$ (this can be so if and only if x_i occurs in clause c_j). There is an edge in H between $y_{i,j}$ and $c_{j,2i}$ (this can be so if and only if x_i occurs in clause c_j).

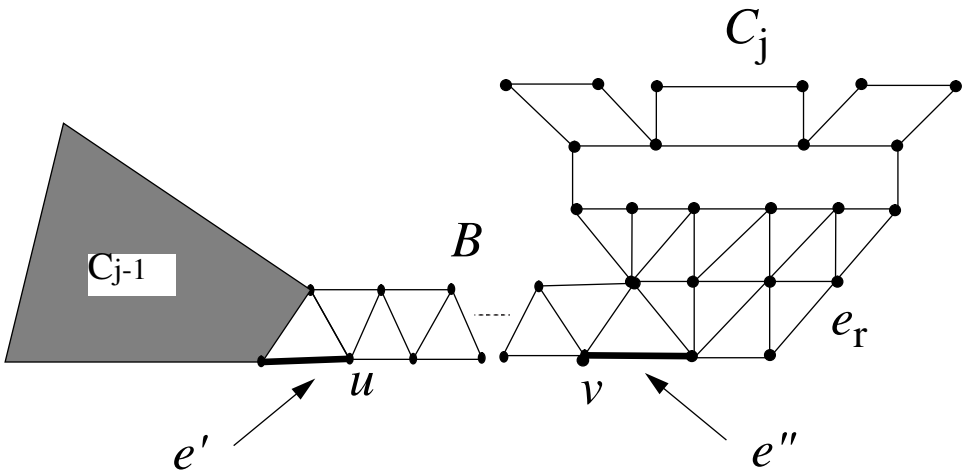

Figure 5.12: Constructing Clause Subgraph S_j from Clause Subgraph S_{j-1}

Construct a drawing D of H as shown in Fig. 5.13(a) in which the vertices of P are placed on the same horizontal level and so are the vertices of Q. D is constructed as follows: Place vertex $c_{j,k}$ at coordinates $(4n(j-1)+2k, 0)$. Place vertex $x_{i,j}$ at coordinates $(4m(i-1)+2j-1, 2mn+1)$ and vertex $y_{i,j}$ at coordinates $(4m(i-1)+2m+2j-1, 2mn+1)$. Each edge $(l_{i,j}, c_{j,k})$, where $l = x$ or y and k is $2i-1$ if $l = x$ and is $2i$ if $l = y$, is drawn as three line-segments– one horizontal line segment $h(l,i,j)$ and two vertical line-segments $t(l,i,j)$ and $b(l,i,j)$ (see Fig. 5.13(a), where $l = x$). If $l = x$, then $b(l,i,j)$ is drawn at the x-coordinate $4n(j-1)+2(2i-1)$, $t(l,i,j)$ is drawn at the x-coordinate $4m(i-1)+2j-1$, and $h(l,i,j)$ is drawn at the y-coordinate $2m(i-1)+j$. If $l = y$, then $b(l,i,j)$ is drawn at the x-coordinate $4n(j-1)+4i$, $t(l,i,j)$ is drawn at the x-coordinate $4m(i-1)+2m+2j-1$ and $h(l,i,j)$ is drawn at the y-coordinate $2m(i-1)+m+j$. Notice that because each $c_{j,i}$ is placed at an even x-coordinate, and each $x_{i,j}$ and $y_{i,j}$ is placed at an odd x-coordinate, $h(l,i,j)$ has non-zero length.

2. From D, construct a new drawing D', by replacing the drawing of each edge $(l_{i,j}, c_{j,k})$ where $l = x$ or y, and k is $2i-1$ if $l = x$ and is $2i$ if $l = y$, by the drawing of a connector $N(l,i,j)$, with width ϵ, where $0 < \epsilon < 0.5$, as shown in Fig. 5.13(b) (this also includes replacing $l_{i,j}$ and $c_{j,k}$ by the input and output vertices respectively of $N(l,i,j)$). The first turn vertex of $N(l,i,j)$ is placed at coordinates $(4m(i-1)+2j-1, 2m(i-1)+j)$ if $l = x$, and is placed at coordinates $(4m(i-1)+2m+2j-1, 2m(i-1)+j)$ if $l = y$. The second turn vertex of $N(l,i,j)$ is placed at coordinates $(4n(j-1)+2(2i-1), 2m(i-1)+j)$ if $l = x$, and is placed at coordinates $(4n(j-1)+4i, 2m(i-1)+j)$ if $l = y$.

3. Construct a planar drawing D'' by replacing the crossings in D' by new vertices (see Fig. 5.13(c)). The connection subgraph K of A(T) is such that

   - there is a one-to-one correspondence between the vertices of D and the vertices of D'',

- there is a one-to-one correspondence between the edges of D and the line-segments of D'', and
- the angle between any two consecutive edges incident on the same vertex of D is same as the one between the corresponding line-segments in D''.

Angle Graph A(T)

We have already described the construction of three subgraphs of A(T), namely, the variable subgraph V_n, the clause subgraph C_m and the connection subgraphs K. The rest of A(T) is constructed as follows:

- Suppose literal l_i where $l = x$ or $l = y = \bar{x}$ occurs in clauses $c_{j_1}, c_{j_2}, \ldots, c_{j_r}$, where $1 \leq j_k < j_{k+1} \leq m$. "Connect" siblings l_k and l_{k+1} in horse shoe X_i (recall that X_i represents variable x_i) with connector $N(l, i, j_k)$, using a wiggle $W(l, i, j_k)$ (see Fig. 5.10(b) which shows the connection of x_k and x_{k+1} with $N(x, i, j_k)$ using $W(x, i, j_k)$). Edge (v_k, v_{k+1}), where, $v = x$ if $l = x$ and $v = y$ if $l = \bar{x}$, is the *image* associated with literal l_i, of clause c_{j_k} in horse shoe X_i.
- Suppose clause c_j consists of literals $l^0_{p_0}$, $l^1_{p_1}$, and $l^2_{p_2}$, where, for q equal to 0 or 1, $p_q \leq p_{q+1}$, and $l^q = x$ or $l^q = \bar{x}$ (if $p_q = p_{q+1}$, then $l^q = x$ and $l^{q+1} = \bar{x}$). "Connect" vertices c_{2q} and c_{2q+1} of crown C_r with connector $N(l^q, p_q, j)$ using a connector $T(l^q, p_q, r)$ (Fig. 5.10(b) shows the connection of $N(x, i, j_k)$ with the vertices c_0 and c_1 of crown C_{j_k} using a connector $T(x, i, j_k)$). Edge (c_{2q}, c_{2q+1}) is the *image* of the literal l_{p_q} in the crown C_j.
- Now choose any horse shoe X_i and crown C_j, and "connect" X_i with C_j by identifying the upper clamp of a holder R with the top edge of X_i and the lower clamp of R with the base of C_j (see Fig. 5.10(b)).

Fig. 5.14 shows an example.

Lemma 18 *Let T be an instance of the 3-SAT problem with n variables and m clauses. Its associated angle graph A(T) has $O(n^2m^2)$ edges and vertices, and can be constructed in $O(n^2m^2)$ time.*

Proof:

From its recursive construction, it is easy to see that the variable subgraph of A(T) has $O(n)$ edges and can be constructed in $O(n)$ time. Similarly, the clause subgraph of A(T) has $O(m)$ edges and vertices, and can be constructed in $O(m)$ time.

As for the connection subgraph of A(T), consider the approach given in Section 5.2.1 for constructing it. Graph H has exactly $3m = O(nm)$ edges. The drawing D, and hence the drawing D', has $O(n^2m^2)$ crossings. Therefore, the connection subgraph of A(T) has $O(n^2m^2)$ edges and vertices, and can be constructed in $O(n^2m^2)$ time.

The rest of A(T) can easily be constructed in $O(3m) = O(nm)$ time, by adding $O(3m) = O(nm)$ edges and vertices. □

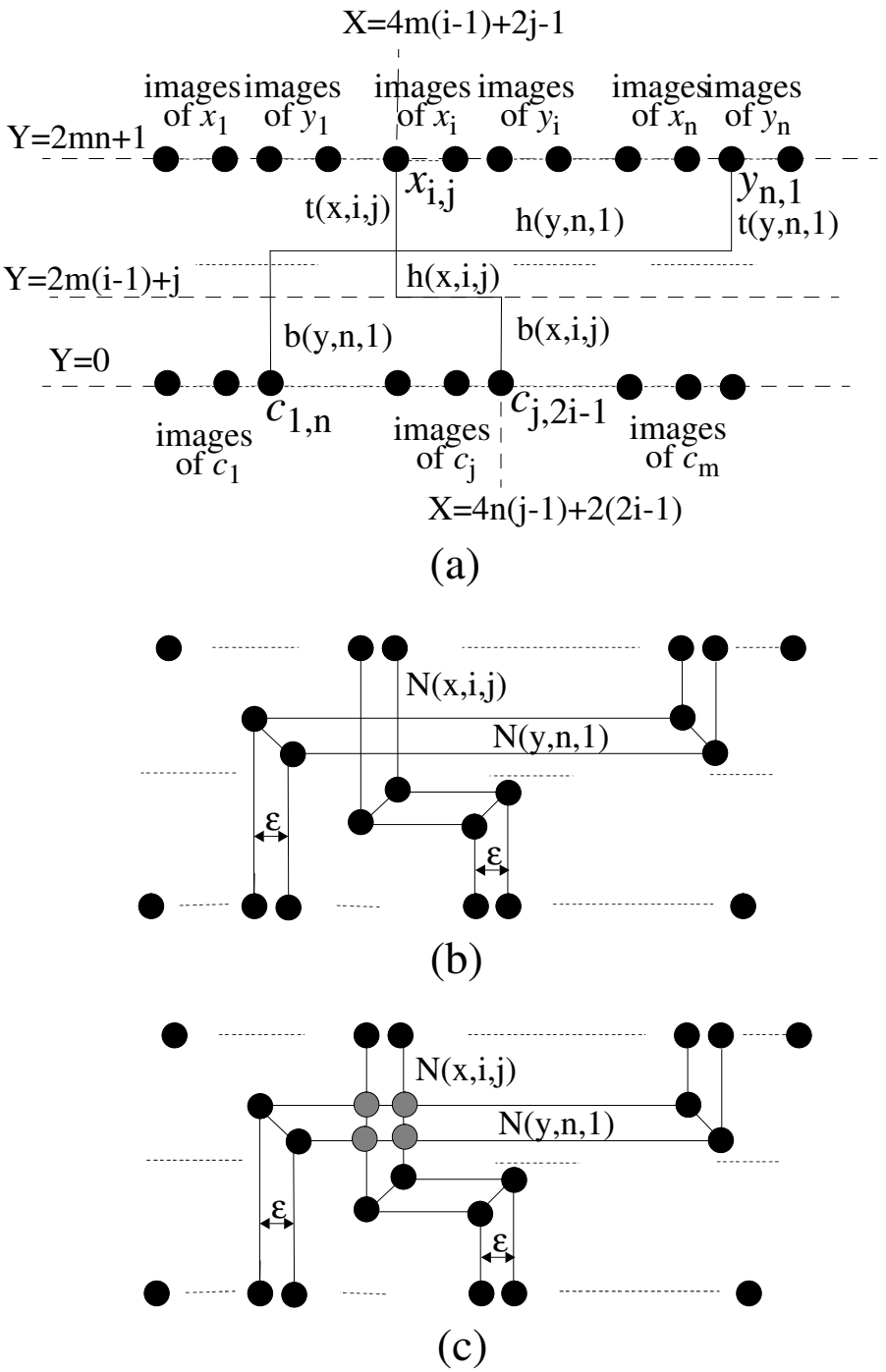

Figure 5.13: Bipartite graph H: (a) A drawing D of H; (b) Drawing D'; (c) Planar Drawing D''.

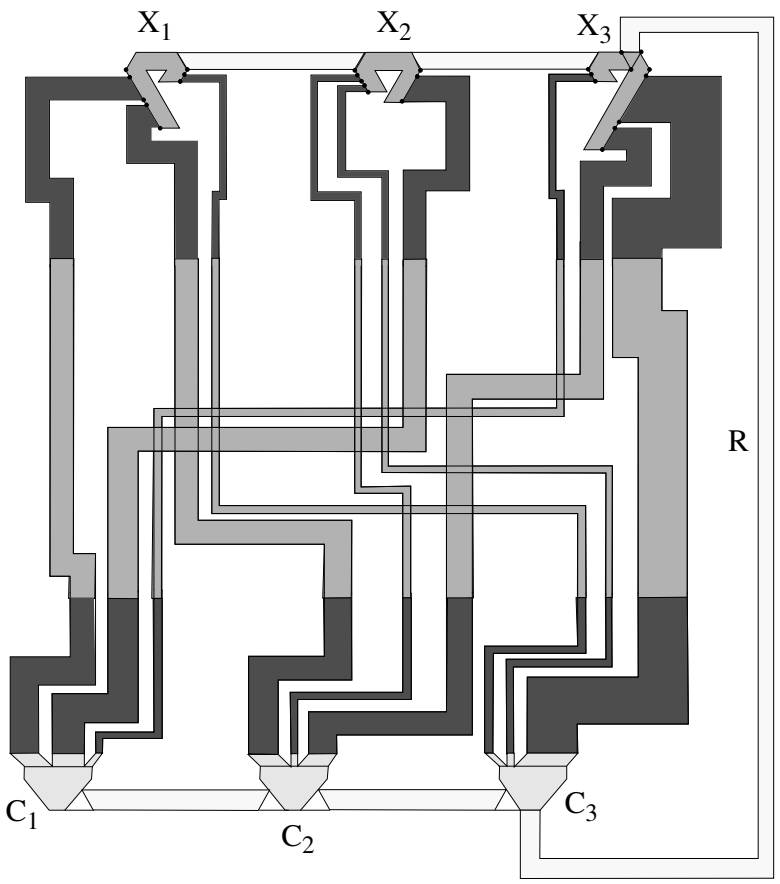

Figure 5.14: A planar drawing D of the associated angle graph A(T) of an instance T of the 3-SAT problem, where T consists of the variables x_1, x_2 and x_3, and the clauses $(x_1, \bar{x_2}, x_3)$, $(x_1, x_2, \bar{x_3})$ and $(\bar{x_1}, x_2, \bar{x_3})$. D corresponds to the truth-assignment x_1 =`true`, x_2 =`false` and x_3 =`false`. This figure is not to scale. Also, to reduce the visual complexity, we have not shown all the vertices and edges of A(T), and have filled some regions of the drawing with shading. In the planar drawing of A(T) described in the *only if* part of the proof of Lemma 19, C_1, C_2 and C_3 are drawn at x-coordinates that are to the left of the x-coordinates at which X_1 is drawn.

Lemma 19 *Let T be an instance of the 3-SAT problem. There is a satisfying truth assignment of T if and only if its associated angle graph $A(T)$ is planar.*

Proof: Suppose T has n variables and m clauses.

If: Suppose A(T) is planar. Let D be a planar drawing of A(T) (see Fig. 5.14 for an example). Let the clamps of R have unit lengths each in D. Every X_i is either left heavy or right heavy or balanced in D (Observation 2). Set variable x_i to `true` if X_i is left heavy, and to `false` otherwise. We show that this is a truth-assignment. That is, for every clause c_j in T, at least one of the literals occurring in c_j is `true`. Suppose literal l_i, where $l = x$ or $l = \bar{x}$, occurs in c_j. From Observations 6 and 8, the length of the image of l_i in crown C_j is equal to $\sqrt{3}/2$ times the length of the image associated

with l_i, of c_j in X_i. Therefore, the image of x_i in C_j has at least $\sqrt{3}/2$ units length only if literal l_i is `true`. From Observation 3, at least one of the images in C_j, of the literals occurring in c_j has length at least $\sqrt{3}/2$ units. Therefore, at least one of the literals occurring in c_j is `true`.

Only If: Let Ψ be a satisfying truth-assignment of T. From Ψ, we can construct a planar drawing D of A(T) (see Fig. 5.14 for an example). A horse shoe X_i is left heavy in D if variable x_i is `true`, and is right heavy in D if variable x_i is `false`.

Following observations provide the basis for the construction of D (assume that the clamps of R are of unit length):

- Recall that, in Section 5.2.1, the connection subgraph K of A(T) was constructed from a planar drawing D''. K has a similar drawing in D.
- The holder, wiggles, variable subgraph and clause subgraph of A(T), each admit a planar drawing.
- In any drawing of A(T), because of Observations 6 and 8, if the image of a literal l_i, where $l = x$ or $l = \bar{x}$, in crown C_j have unit length, then the image associated with l_i, of clause c_j in horse shoe X_i has length $2/\sqrt{3}$.
- In any drawing of A(T), because of Observations 3, 6 and 8, the width of any connector and of any wiggle is at most 5 units. Because there are at most $2nm$ connectors in the connection subgraph K, the cumulative width of the connectors in K is at most $10nm$. Similarly, the sum of the widths of the wiggles, and the remaining connectors is $10nm$ each. Therefore, all the connectors and wiggles can be drawn without crossings, in a total width of $12nm$, such that there is a horizontal separation of at least one unit between two connectors or wiggles, whenever they pass through the same y-coordinate. A similar reasoning shows that all the connectors and wiggles can be drawn without crossings, in a total height of $3 \cdot 12nm = 36nm$, such that there is a vertical separation of at least one unit between two connectors or wiggles, whenever they pass through the same x-coordinate.
- From Observation 3, the image of a literal in a crown has length at most 5 units. Therefore, from Observations 6 and 8, the image of a clause in a horse shoe X_i has length at most $10/\sqrt{3}$. Hence, X_i can be drawn with height at most $5m + 1$ units and a width at most $5m + 1$ units such that for each clause c_j in which literal l_i ($l = x$ or $l = \bar{x}$) occurs, the image of c_j in X_i, associated with l_i, has length $2/\sqrt{3}$ times the length of the image of l_i in crown C_j.
- Each crown can be drawn such that it has height 4 units and width at most 6 units.
- Recall that the length of a beam is $12mn$ (Observation 5).

We now describe D. Let ϵ be a constant, such that $0 < \epsilon < \sqrt{3}/(2m)$. In D,

- The clamps of the holder R of A(T) have unit length.

- The shaft of R is drawn vertically at the x-coordinate $12mn(n+m)$.
- The base of the crown C_j is drawn horizontally at the y-coordinate 0, and between x-coordinates $(12mn+3+2/\sqrt{3})(j-1)+1+1/\sqrt{3}$ and $(12mn+3+2/\sqrt{3})(j-1)+2+1/\sqrt{3}$.
- The top edge of the horse shoe X_i is drawn horizontally at the y-coordinate $4+3\cdot 36nm+5m+1=108nm+5m+5$, and between x-coordinates $12m^2n+(12mn+3)(i-1)+3/2$ and $12m^2n+(12mn+3)(i-1)+5/2$.
- If k literals appearing in a clause c_j are `true`, then the images in the crown C_j, of each of these `true` literals have lengths $(5-\epsilon)/k$ units.
- If k literals appearing in a clause c_j are `false`, then the images in the crown C_j, of each of these `false` literals have lengths ϵ/k units. Notice that because of our choice of ϵ, if a literal l_i, where $l =$ or $l = \bar{x_i}$, is `false`, then in horse shoe X_i, the sum of the images associated with l_i, of the clauses of T in which l_i occurs, is less than one unit.
- The Wiggles and connectors are drawn without crossings in the empty region of height $36nm$ and width $12mn^2$, that is to the left of the shaft of R and is bounded from above and below by the drawings of the variable subgraph and the clause subgraph of A(T) respectively. Furthermore, they are drawn such that there is a vertical (horizontal) separation of at least one unit between two connectors or wiggles, whenever they pass through the same x-coordinate (y-coordinate). As observed earlier, because drawing them without crossings in this manner requires a total width of at most $12nm$ and a height of at most $36nm$, they can be drawn in this manner in the aforementioned region.

□

This yields the main theorem of this section.

Theorem 16 *The problem of testing whether a given consistent angle graph is planar is NP-hard.*

Proof: Let T be an instance of the 3-SAT problem. From Lemmas 18 and 19, we can construct the associated angle graph A(T) of T in $O(n^2m^2)$ time, such that A(T) is planar if and only if T admits a satisfying truth-assignment. Hence, given a consistent angle graph, testing whether it is planar or not, is NP-hard. □

All the angles of A(T) are multiples of 15°. Therefore,

Corollary 5 *The problem of testing whether a consistent angle graph is planar is NP-hard even if all the angles are multiples of some non-negative integer.*

A *bounded degree* angle graph is one in which each vertex has a bounded number (i.e., independent of the number of edges and vertices in the graph) of edges incident on it. Each vertex of A(T) has at most seven edges incident on it. Therefore,

Corollary 6 *The problem of testing whether a consistent bounded degree angle graph is planar is NP-hard.*

5.3 Testing Triconnected Planar Graphs for High Angular Resolution

In this section we consider the problem of constructing planar straight line drawings of planar graphs with high angular resolution and show that the following problem, called the *angular resolution problem for triconnected planar graphs*, is NP-hard:

> Given a triconnected planar graph G and an angle α, determine whether G has a planar straight-line drawing with angular resolution at least α.

We show that the angular resolution problem for triconnected planar graphs is NP-hard by reducing the 3-SAT problem to it. Let T be an instance of the 3-SAT problem. We first construct the associated angle graph A(T) of T. Let H(T) be the underlying planar graph of A(T). We construct an *associated* triconnected planar graph G(T) of T, by adding new vertices and edges to H(T) such that (*a*) G(T) is constructed in polynomial time from H(T), and (*b*) A(T) is planar (and hence, from Lemma 19, T is satisfiable) if and only if G(T) admits a planar straight-line drawing with angular resolution 5°.

G(T) is constructed from H(T) by adding graphs called *fans* to it. *fans* are described by Kant [68, 69]. Fan F_6 consisting of 6 vertices is shown in Fig. 5.15(*a*). Edges e_1 and e_2 are called the *connectors* of the fan and are used to connect the fan to other vertices of the graph to which the fan is added (in our case, to make G(T) triconnected). Vertex v is called the *hub* of the fan. In general, a fan $\mathcal{F}_i$ has i vertices.

The purpose of using fans is to "fix" the angles between the edges of H(T) to desired magnitudes, in a planar straight-line drawing of G(T) with angular resolution 5° (notice that H(T) is a subgraph of G(T)). For example, as shown in Fig. 5.15(*b*), if a vertex v of H(T) has three edges incident on it, then we can guarantee that the angle between its three incident edges is 45°, 90°, and 225° respectively, in any planar straight-line drawing of G(T) with angular resolution 5°, by adding three fans $\mathcal{F}_9$, $\mathcal{F}_{18}$ and $\mathcal{F}_{45}$ to it by identifying their hubs with v. The angles will be so because after adding the fans, v has 72 edges incident on it and in any planar straight-line drawing of G(T) with angular resolution 5°, the angle between each pair of consecutive edges incident on v has the same magnitude (equal to 5°). In general, if a vertex v of H(T) has d edges incident on it, we can "fix" the angles between its edges to desired magnitudes, by adding d appropriate fans–one for each angle–to H(T) by identifying their hubs with v.

We now describe construction of G(T) from A(T). Let H(T) be the underlying graph of A(T). Denote by $a(e_1, e_2)$, the angle in A(T) between two consecutive edges e_1 and e_2 of H(T) incident on the same vertex. Constructing G(T) from A(T) is a two-step process:

1. First, for every pair of consecutive edges e_1 and e_2 of H(T) incident on the same vertex v, add a fan $\mathcal{F}_{a(e_1,e_2)/5}$ to H(T) by identifying the hub of the fan with v.

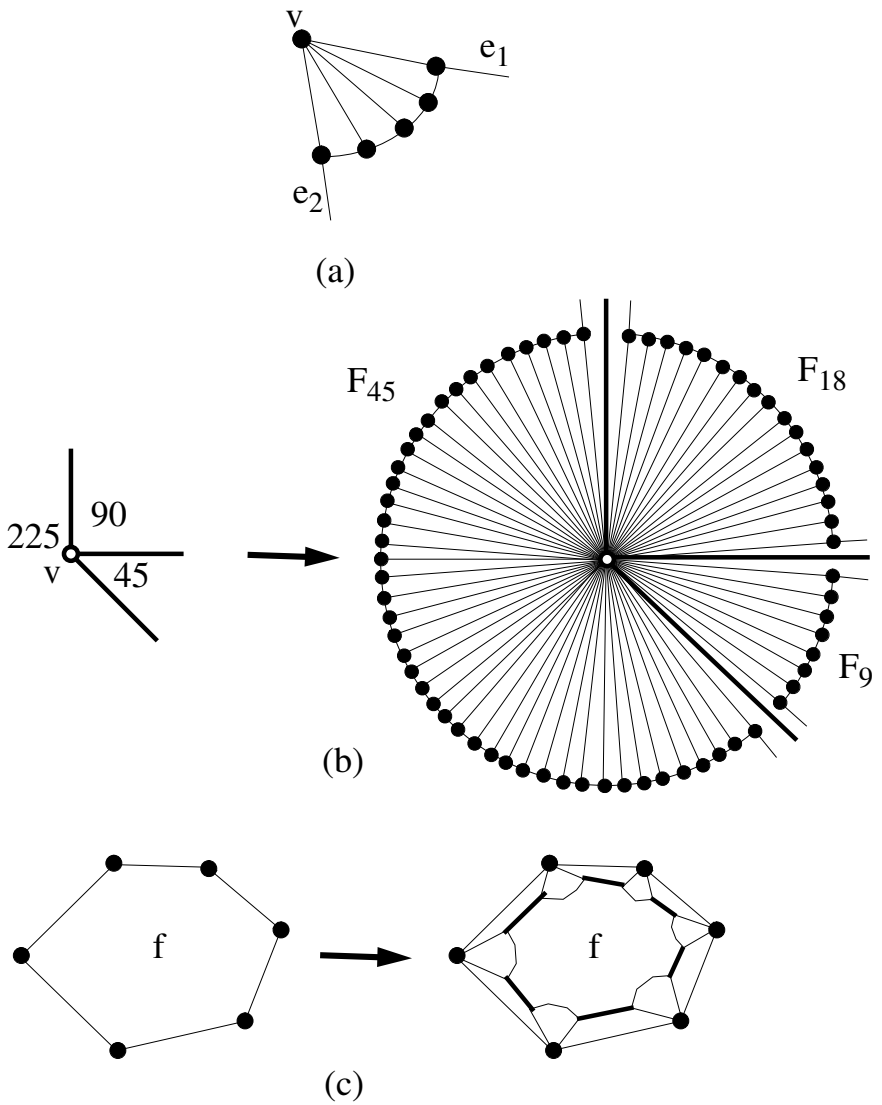

Figure 5.15: Constructing Graph G from A(T): (a) Fan F_6; (b) Attaching fans $\mathcal{F}_9$, $\mathcal{F}_{18}$ and $\mathcal{F}_{45}$ to "fix" the angle between the edges (shown as thick lines) incident on v to 45°, 90°, and 225° respectively; (c) Connecting fans introduced in the same face- The connectors are shown as thick lines.

Notice that each angle of A(T) is a multiple of 5°.

2. Next, for every face f of H(T), join the fans introduced in f, using their connectors, in a cyclic order as shown in Fig. 5.15(c).

For proving that G(T) is triconnected we use the following result of [100]. Two vertices u and v of G(T) are *adjacent* if there is an edge (u, v) in G(T).

Theorem 17 (Tamassia [100]) *Let G be a biconnected planar graph. Vertices u and v form a separating pair of G if and only if*

1. *u and v are adjacent and there are at least three faces that contain u and v; or*

2. *u and v are not adjacent and there are at least two faces that contain u and v.*

Now we are ready to present the main lemma of this section.

Lemma 20 *Given an instance T of the 3-SAT problem with n variables and m clauses, its associated triconnected planar graph $G(T)$ can be constructed in $O(n^2m^2)$ time. Moreover, $G(T)$ admits a planar straight-line drawing with angular resolution $5°$ if and only if T admits a satisfying truth-assignment.*

Proof: Using Lemma 18, first construct the associated angle graph A(T) of T in $O(n^2m^2)$ time. Let H(T) be the underlying graph of A(T). Constructing G(T) from H(T) requires adding $O(n^2m^2)$ fans, each with at most $300/5 = 60$ vertices (because the largest angle in A(T) is $300°$). This construction can be carried out very easily in constant time per fan. Therefore, G(T) can be constructed from T in $O(n^2m^2)$ time.

Now we show that G(T) is triconnected. From the construction of A(T) it is easy to see that A(T) and therefore, G(T) is biconnected. Because of the introduction of fans and connecting them in a cyclic order in each face, using their connectors (see Fig. 5.15(c)), the following holds: (a) If f, g and h are three faces of G(T), then f, g and h have at most one vertex in common. This satisfies Condition 1 of Theorem 17. (b) If f and g are two faces of G(T), then f and g have at most two vertices in common, and if u and v are common vertices of f and g, then u and v are adjacent. This satisfies Condition 2 of Theorem 17. Therefore, from Theorem 17, G(T) is triconnected.

We now show that G(T) admits a planar straight-line drawing with angular resolution $5°$ if and only if A(T) is planar. Let the edges of G(T) that are also edges of H(T) be called its *primary edges*. Given a planar drawing of A(T) it is easy to get a planar straight-line drawing of G(T) with angular resolution $5°$, by simply placing the edges of each fan incident on its hub, at an angular separation of $5°$ from each other and from the primary edges of G(T). The remaining edges of G(T) (such as connector edges of the fans, etc.) can easily be placed at an angle at least $5°$ from their neighboring edges.

Now suppose that G(T) has a planar straight-line drawing D with angular resolution $5°$. Because of the use of fans, in D, the angles between the primary edges of G(T) is equal to the angle specified between them in A(T). We can get a planar drawing of A(T) from D by simply removing the fans of G(T) from D. □

Theorem 18 *Given a triconnected planar graph and an angle α, determining whether or not, it admits a planar straight-line drawing with angular resolution at least α, is NP-hard.*

Proof: Let T be an instance of the 3-SAT problem. From Lemma 20, we can construct the associated triconnected planar graph G(T) of T in $O(n^2m^2)$ time, such that G(T) admits a planar straight-line drawing with angular resolution at least $5°$, if and only if T admits a satisfying truth-assignment. Hence, given a triconnected planar graph and an angle α, determining whether or not, it admits a planar straight-line drawing with angular resolution at least α, is NP-hard. □

5.4 Testing a Series-Parallel Angle Graph for Consistency

In this section we give a linear-time algorithm for testing whether a series-parallel angle graph with non-zero angles is consistent or not, i.e., whether it admits a drawing or not.

A *series-parallel* graph G with *source* s and *sink* t is defined recursively as follows: G is either a single edge (s,t) (see Fig 5.16(a)), or a *series composition* (see Fig 5.16(b)), or a *parallel composition* (see Fig 5.16(c)) of two series-parallel graphs G_1 and G_2. Let s_1 and t_1 be the source and sink respectively, of G_1. Let s_2 and t_2 be the source and sink respectively, of G_2. A *series composition* of G_1 and G_2 is done by identifying vertices s_1 and t_2 (see Fig 5.16(b)); For G in this case, $s = s_2$ and $t = t_1$. A *parallel composition* of G_1 and G_2 is done by identifying vertices t_2 and t_1, and the vertices s_2 and s_1(see Fig 5.16(c)); For G in this case, $s = s_1 = s_2$ and $t = t_1 = t_2$. G_1 and G_2 are called the *constituents* of G. A *series-parallel subgraph* of G is either G, or a series-parallel subgraph of G_1 or G_2. Associated with G is a SPQ-tree called the *decomposition* tree T(G) of G. T(G) has three types of nodes- S, P and Q. Each Q node corresponds to an edge of G (see Fig 5.16(a)), each S node corresponds to a series-composition (see Fig 5.16(b)), and each S node corresponds to a series-composition (see Fig 5.16(c)). Fig 5.16(e) shows the associated decomposition tree of the series-parallel graph shown in Fig 5.16(d). T(G), therefore, maintains information about the structure of G in terms of its series-parallel subgraphs, and can be constructed in linear time [6].

A *series-parallel angle graph* is an angle graph with a series-parallel graph as its underlying graph.

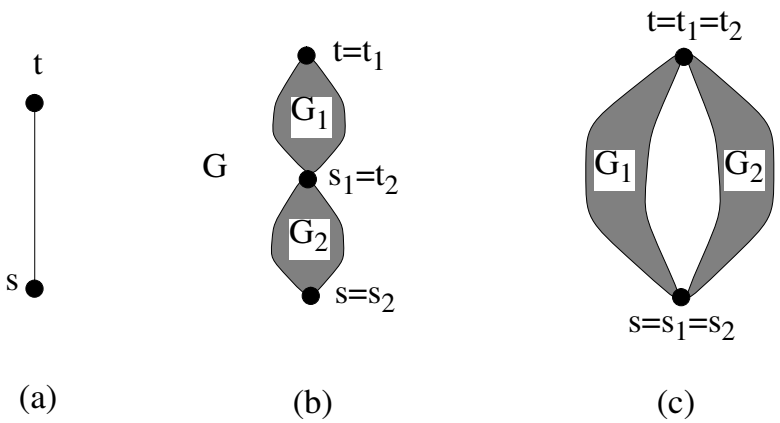

Figure 5.16: A series-parallel graph G: (a) Single edge (s,t); (b) Series composition of G_1 and G_2; (c) Parallel composition of G_1 and G_2.

We now describe how to test in linear time, whether a given series-parallel angle graph G with non-zero angles admits a drawing or not. We assume a polar coordinate system, with an origin o and a reference axis. We denote the angle made by a line segment $\vec{ab}$ with the the reference axis by $angle(\vec{ab})$. l(θ) denotes a line drawn from o at an angle θ with the reference axis.

In a preprocessing phase, we align the reference axis with an arbitrary edge of G and using this alignment compute the angles made by each edge of G with the the reference axis.

Let A be a series-parallel subgraph of G. Let s and t be the source and sink respectively of A. The testing of A for consistency is done recursively. Let A_1 and A_2 be the two constituents of A. We define a tuple $sector(A)$ = (*lowerboundtype,upperboundtype, lowerboundangle,upperboundangle*) that represents the smallest (geometric) sector (see Fig. 5.17(a)) that contains all the possible drawings of A, assuming that s is always placed at the origin of the sector. Notice that $sector(A)$ is bounded only by its boundary angles *lowerboundangle* and *upperboundangle*; It has infinite radius. $sector(A)$ is completely characterized as follows:

1. $0 \leq lowerboundangle \leq 360$, $0 \leq upperboundangle \leq 720$, and $lowerboundangle \leq upperboundangle$.

2. *lowerboundtype* and *upperboundtype* are each either *NONE*, *CLOSED* or *OPEN*.

3. *lowerboundtype* = *upperboundtype* =*NONE* if and only if A does not admit any drawing.

4. *lowerboundtype* and *upperboundtype* are not *NONE*, $lowerboundangle = \alpha$, and $upperboundangle = \beta$ if and only if

   (a) A admits at least one drawing,

   (b) In no drawing of A, Line $\vec{st}$ is at an angle less than α or more than β with the reference axis.

   (c) If $\alpha \neq \beta$, then for each θ, such that $\alpha < \theta < \beta$, there exists a drawing of A in which Line $\vec{st}$ is at an angle θ with the reference axis.

5. *lowerboundtype* is *CLOSED* and $lowerboundangle = \alpha$ if and only if there is a drawing of A in which Line $\vec{st}$ is at an angle α with the reference axis, but there is no drawing of A in which Line $\vec{st}$ is at an angle less than α with the reference axis.

6. *upperboundtype* is *CLOSED* and $upperboundangle = \beta$ if and only if there is a drawing of A in which Line $\vec{st}$ is at an angle β with the reference axis, but there is no drawing of A in which Line $\vec{st}$ is at an angle greater than β with the reference axis.

7. *lowerboundtype* is *OPEN* and $lowerboundangle = \alpha$ if and only if there is no drawing of A in which Line $\vec{st}$ is at an angle α with the reference axis, but for any ϵ, such that $0 < \epsilon < upperboundangle - \alpha$, there is a drawing of A in which Line $\vec{st}$ is at an angle equal to $\alpha + \epsilon$ with the reference axis.

8. *upperboundtype* is *OPEN* and $upperboundangle = \beta$ if and only if there is no drawing of A in which Line $\vec{st}$ is at an angle β with the reference axis, but for any ϵ, such that $0 < \epsilon < \beta - lowerboundangle$, there is a drawing of A in which Line $\vec{st}$ is at an angle equal to $\beta - \epsilon$ with the reference axis.

For example, each drawing of the series-parallel graph G shown in Fig. 5.18(a), is contained in the sector R shown in Fig. 5.18(b), assuming that the edge (s, u) is aligned with the reference axis. $sector(G)$, which represents R, is equal to $(OPEN, OPEN, 0, 90)$. The *upperboundarytype* and *lowerboundarytype* for $sector(G)$ are both $OPEN$ because both su and tu have non-zero lengths in every drawing of G.

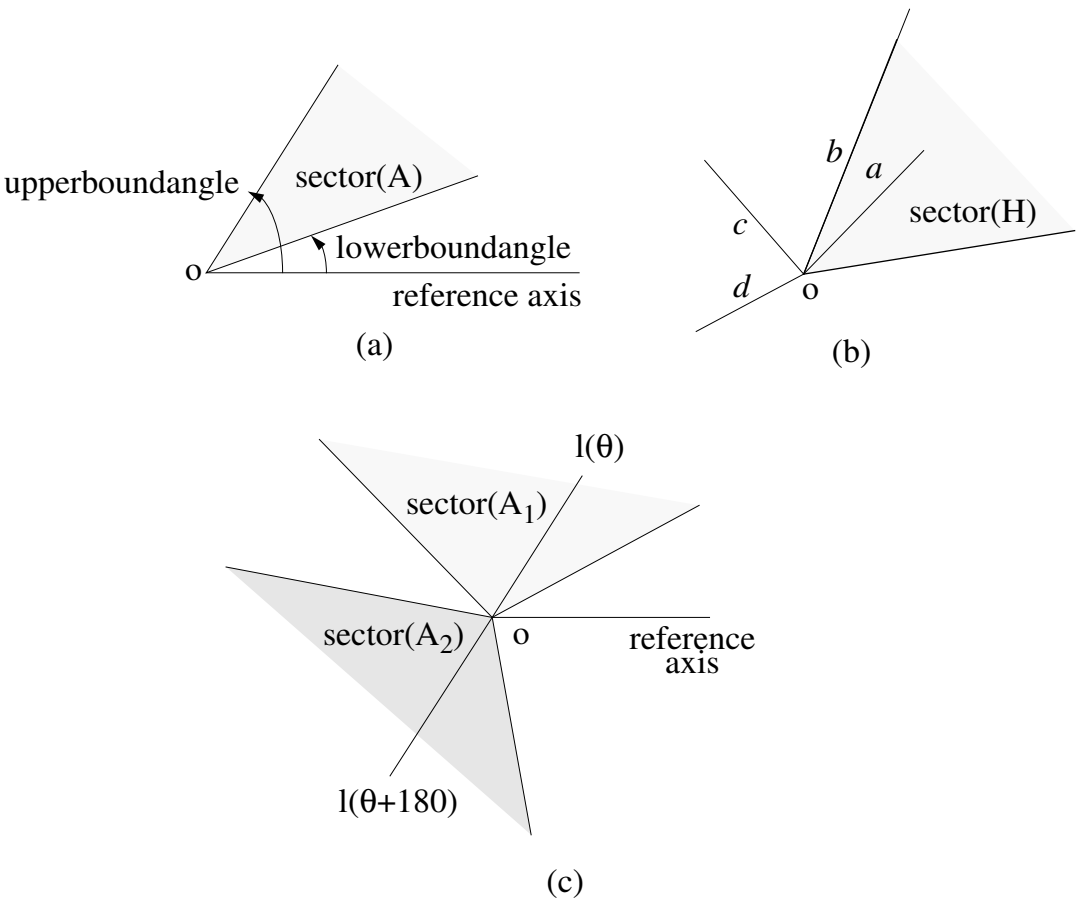

Figure 5.17: Illustration of the concepts of: (a) a sector, (b) *lies inside* and *reverse included*, and (c) *opposite* sectors.

Let $sector(A) = (lb, ub, \alpha, \beta)$. *Width* of $sector(A)$ is equal to $\beta - \alpha$. Therefore, $sector(A)$ is a disk if it has width 360 and $lb = ub = CLOSED$. $sector(A)$ is *simple* if $sector(A)$ is either a disk, or has width at most 180. We say that a line $l(\theta)$ *lies inside* $sector(A)$ if the following holds: (a) $\theta > \alpha$ if lb= *OPEN* and $\theta \geq \alpha$ if lb= *CLOSED*, and (a) $\theta < \beta$ if ub= *OPEN* and $\theta \leq \beta$ if ub= *CLOSED*. We say that a line $l(\theta)$ *lies properly inside* $sector(A)$ if $\alpha < \theta < \beta$. Notice that if $\alpha = \beta$, then no line can lie properly inside $sector(A)$. We say that a line $l(\theta)$ is *reverse included* in $sector(A)$ if Line $l(\theta + 180)$ lies properly inside $sector(A)$. For example, in Fig. 5.17(b), Line a lies properly inside $sector(A)$, Line b lies inside $sector(A)$, Line d is reverse included in $sector(A)$, and Line c neither lies inside nor is reverse included in $sector(A)$. $sector(A_1)$ and $sector(A_2)$ are *opposite* if there is a line l, such that, either l lies inside $sector(A_1)$ and is reverse included in $sector(A_2)$, or l lies inside $sector(A_2)$ and is reverse included in $sector(A_1)$

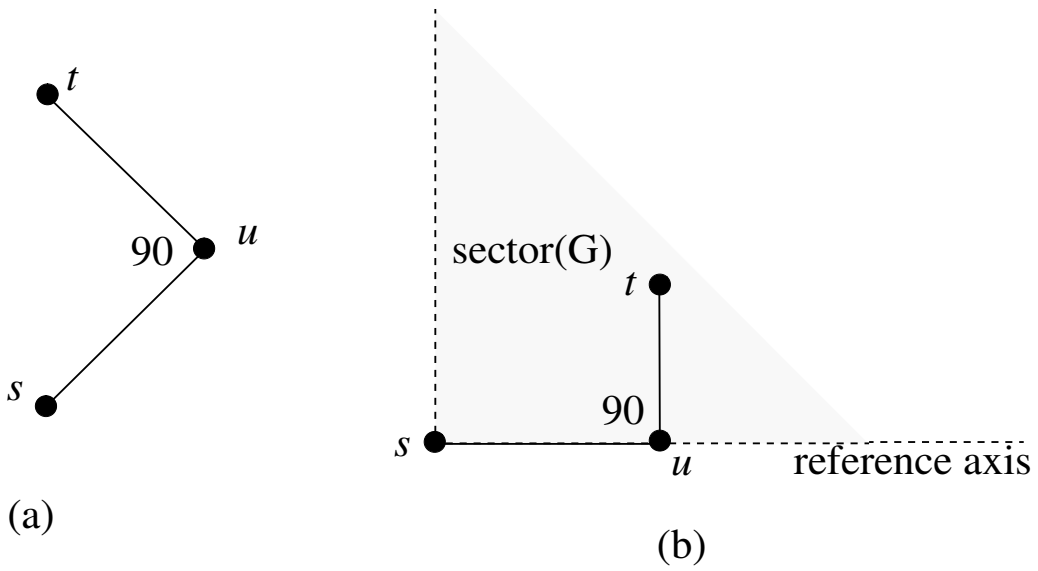

Figure 5.18: (a) A series-parallel angle graph G; (b) $Sector(G)$, assuming that su is aligned with the reference axis.

(see Fig. 5.17(c)).

$sector(A)$, such that it is a simple sector, is determined recursively as follows:

1. (Recursively) determine $sector(A_1)$.
2. (Recursively) determine $sector(A_2)$.
3. Determine $sector(A)$ from $sector(A_1)$ and $sector(A_2)$.

$sector(A)$ can be determined from $sector(A_1)$ and $sector(A_2)$ by considering the following three cases:

1. **A is a single edge $e = (s,t)$:** Then $sector(A) = (CLOSED, CLOSED, \alpha, \alpha)$, where α is the angle (between 0 and 360) made by e with the reference axis. Geometrically, $sector(A)$ is a straight-line at an angle α with the reference axis.
2. **A is a series-composition of A_1 and A_2:** Let $sector(A_1) = (lb_1, ub_1, \alpha_1, \beta_1)$ and $sector(A_2) = (lb_2, ub_2, \alpha_2, \beta_2)$. Set $sector(A)$ to (lb, ub, α, β), where lb, ub, α, and β are determined as follows:

   Case 1: $sector(A_1)$ and $sector(A_2)$ are opposite. This is illustrated in Fig. 5.19(a). In this case, $sector(A) = (CLOSED, CLOSED, 0, 360)$. Geometrically, $sector(A)$ is a disk. Hence, in this case for every point p in the plane, there is drawing of A with s at origin and t at p.

   Case 2: $sector(A_1)$ and $sector(A_2)$ are not opposite. Geometrically $sector(A)$ is the sector with smallest width that contains both the sectors $sector(A_1)$ and $sector(A_2)$ (see Fig. 5.19(b,c)). Notice that $sector(A)$ in this case has

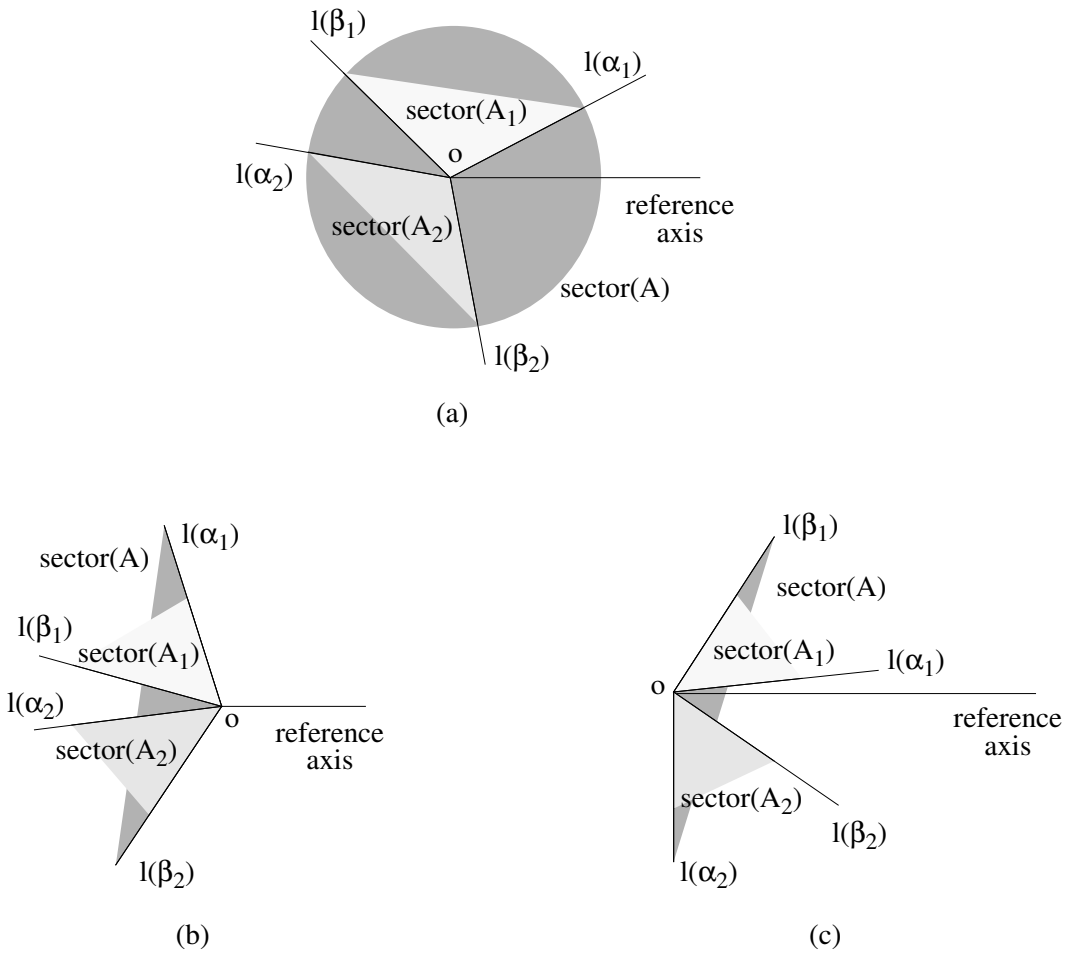

Figure 5.19: $sector(A)$, shown as darkly shaded region, when the series-parallel angle graph A is a series-composition of series-parallel angle graphs A_1 and A_2 ($sector(A_1)$ and $sector(A_2)$ are shown as lightly shaded regions): (a) $sector(A_1)$ and $sector(A_2)$ are opposite; (b) $sector(A_1)$ and $sector(A_2)$ are not opposite, and $\alpha_2 - \beta_1 \leq 180$; (c) $sector(A_1)$ and $sector(A_2)$ are not opposite, and $\alpha_2 - \beta_1 > 180$;

width less than or equal to 180 (because otherwise $sector(A_1)$ and $sector(A_2)$ are opposite, and Case 1 applies). Assume without loss of generality that $\alpha_1 \leq \alpha_2$. Now we show how to compute $sector(A)$ numerically. α, and β can be determined by considering two cases:

$\alpha_2 - \beta_1 \leq 180$: This case is illustrated in Fig. 5.19(b). In this case, $\alpha = \alpha_1$ and $\beta = \max(\beta_1, \beta_2)$.

$\alpha_2 - \beta_1 > 180$: This case is illustrated in Fig. 5.19(c). In this case, $\alpha = \alpha_2$, and $\beta = \max(\beta_1 + 360, \beta_2)$.

lb and ub can be determined as follows:

- If lb_1=$OPEN$, or lb_2=$OPEN$, or $\alpha_1 \neq \alpha_2$, then $lb = OPEN$, otherwise, $lb = CLOSED$.
- If ub_1=$OPEN$, or ub_2=$OPEN$, or $\beta_1 \neq \beta_2$, then $ub = OPEN$, otherwise,

$ub = CLOSED$.

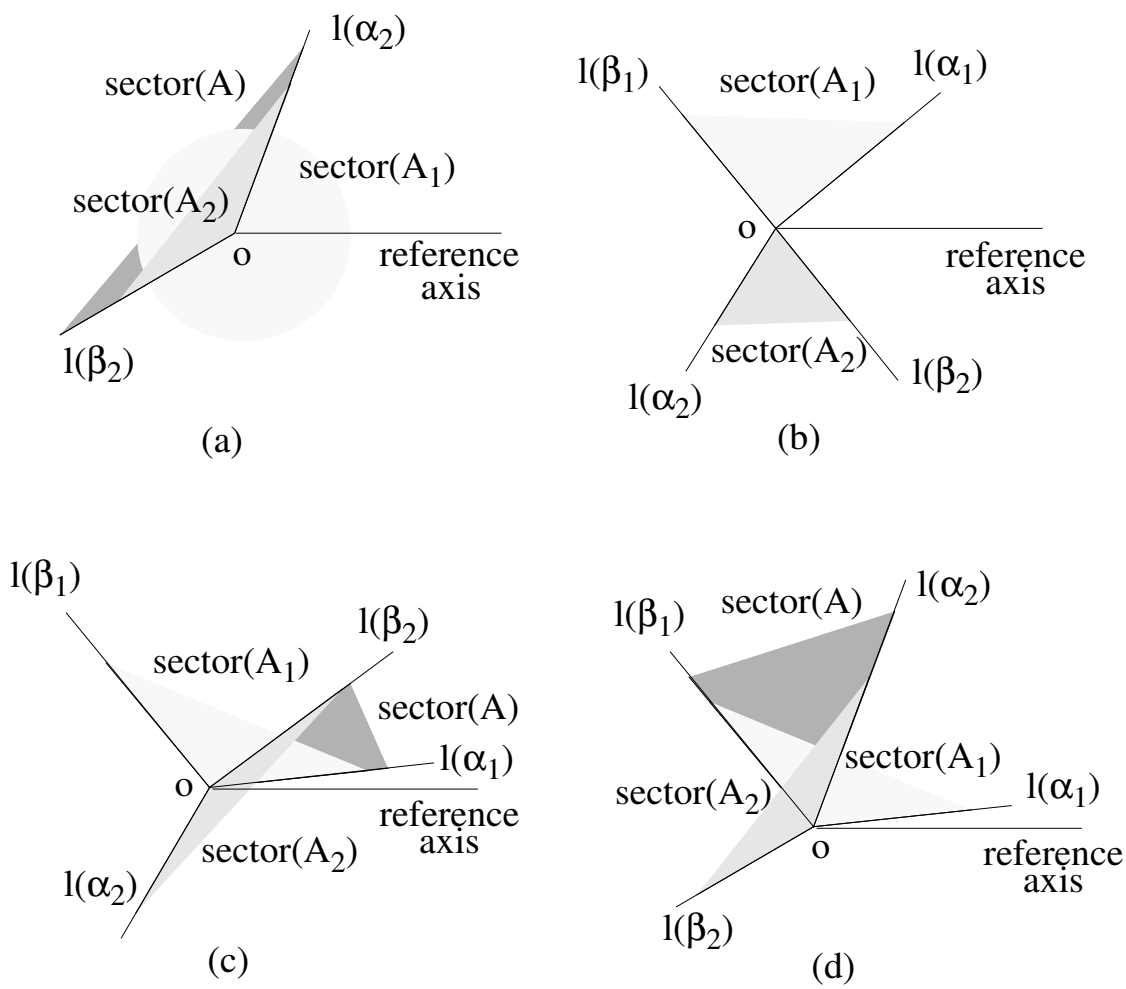

Figure 5.20: $sector(A)$, shown as darkly shaded region, when the series-parallel angle graph A is a series-composition of series-parallel angle graphs A_1 and A_2 ($sector(A_1)$ and $sector(A_2)$ are shown as lightly shaded regions): (a) $sector(A_1)$ is a disk; (b) $\beta_1 < \alpha_2$ and $\beta_2 < \alpha_1 + 360$; (c) $\beta_1 < \alpha_2$ and $\beta_2 \geq \alpha_1 + 360$; (d) $\beta_1 \geq \alpha_2$.

3. A **is a parallel-composition of** A_1 **and** A_2: Let $sector(A_1) = (lb_1, ub_1, \alpha_1, \beta_1)$ and $sector(A_2) = (lb_2, ub_2, \alpha_2, \beta_2)$. Geometrically, $sector(A)$ is the sector with largest width, that is contained in both $sector(A_1)$ and $sector(A_2)$ (see Fig. 5.20). Therefore, A does not admit a drawing if $sector(A_1)$ and $sector(A_2)$ are non-overlapping. Assume without loss of generality that $\alpha_1 \leq \alpha_2$. Also assume that if $\alpha_1 = \alpha_2$ and lb_1 is $OPEN$, then lb_2 is also $OPEN$. By $*$ we denote a value that can be arbitrary chosen because the choice is not important to us. Because $sector(A_1)$ and $sector(A_2)$ are simple sectors, to compute $sector(A)$ numerically, we consider the following three cases:

   $sector(A_1)$ **is a disk.** This is illustrated in Fig. 5.20(a). In this case, $sector(A) = sector(A_2)$.

$\beta_1 < \alpha_2$ **and** $\beta_2 < \alpha_1 + 360$. This is illustrated in Fig. 5.20(b). In this case, $sector(A)$ is empty. Hence, $sector(A) = (NONE, NONE, *, *)$.

$\beta_1 < \alpha_2$ **and** $\beta_2 \geq \alpha_1 + 360$. We have four cases:

$\beta_2 = \alpha_1 + 360$. Then, if $lb_1 = CLOSED$ and $lb_2 = CLOSED$, then $sector(A) = (CLOSED, CLOSED, \alpha_1, \alpha_1)$, otherwise, $sector(A) = (NONE, NONE, *, *)$.

$\beta_1 + 360 > \beta_2 > \alpha_1 + 360$: This is illustrated in Fig. 5.20(c). In this case, $sector(A) = (lb_1, ub_2, , \alpha_1, \beta_2 - 360)$.

$\alpha_1 + 360 < \beta_1 + 360 = \beta_2$: Then, if $ub_1 = CLOSED$, then $sector(A) = (lb_1, ub_1, \alpha_1, \beta_1)$, otherwise, $sector(A) = (lb_1, ub_2, \alpha_1, \beta_2 - 360)$.

$\beta_1 + 360 < \beta_2$: In this case, $sector(A) = (lb_1, ub_1, \alpha_1, \beta_1)$.

$\beta_1 \geq \alpha_2$. Because $sector(A_1)$ and $sector(A_2)$ are both simple, we have that $\beta_2 \leq \beta_1 + 180 \leq \alpha_1 + 360$. We have five cases:

$\alpha_2 = \beta_1$ Then, if $ub_1 = CLOSED$ and $lb_2 = CLOSED$, then otherwise, $sector(A) = (NONE, NONE, *, *)$.

$\beta_2 > \beta_1 > \alpha_2$. This is illustrated in Fig. 5.20(d). In this case, $sector(A) = (lb_1, ub_1, \alpha_1, \beta_1)$.

$\beta_2 = \beta_1 > \alpha_2$. Then, if $ub_1 = OPEN$, then $sector(A) = (lb_1, ub_1, \alpha_1, \beta_1)$, otherwise, $sector(A) = (lb_1, ub_2, \alpha_1, \beta_2)$.

$\beta_1 > \beta_2 > \alpha_1$. Then, $sector(A) = (lb_1, ub_2, \alpha_1, \beta_2)$.

$\beta_2 = \alpha_2 = \alpha_1$. Then, if $lb_1 = CLOSED$, then $sector(A) = (CLOSED, CLOSED, \alpha_1, \alpha_1)$, otherwise, $sector(A) = (NONE, NONE, *, *)$.

Theorem 19 *Given a series-parallel angle graph G with non-zero angles and with n vertices, we can test whether G admits a drawing or not, as well as compute $sector(A)$ for each series-parallel subgraph A of G, in $O(n)$ time.*

Proof: Assume that we are given the associated decomposition tree of G, since it can be constructed in $O(n)$ time [6]. For testing whether G admits a drawing or not, first align the reference axis with an arbitrary edge of G and using this alignment, for each edge e of G, compute the angle between e and the reference axis. This takes $O(n)$ time. Next, compute $sector(G)$ using the recursive procedure given above. Each step of this recursive procedure takes constant time. Since there are $O(n)$ recursive steps, one for each series-parallel subgraph of G, the total time-complexity is $O(n)$.

Now we prove the correctness of the above algorithm. Let $sector(G) = (lb_G, ub_G, \alpha_G, \beta_G)$, as computed by the algorithm. We prove that G admits a drawing if and only if neither lb_G nor ub_G is equal to *NONE*. Let A be a series-parallel subgraph of G. Let $sector(A) = (lb, ub, \alpha, \beta)$, as computed by the algorithm. We show that $sector(A)$ is computed correctly and A admits a drawing if and only if neither lb nor ub is equal to *NONE*. Since G is a a series-parallel subgraph of itself, this will prove the claim for G.

For proving the correctness, we also show that $sector(A)$ has the the *upper boundary* and *lower boundary inheritance* properties given below (recall that in the preprocessing phase, we compute the angles between the edges of G and the reference axis):

lower boundary inheritance property: If $sector(A)$ is not a disk and $lb = CLOSED$, then there are vertices u and v in A, such that angle between edge su and the reference axis is α, and the angle between edge vt and the reference axis is α.

upper boundary inheritance property: If $sector(A)$ is not a disk and $ub = CLOSED$, then there are vertices u' and v' in A, such that angle between edge su' and the reference axis is either β or $\beta - 360$, and the angle between edge $v't$ and the reference axis is either β or $\beta - 360$.

If A consists of one and only one edge, then clearly $sector(A)$ is computed correctly. A admits at least one drawing and correspondingly neither lb nor ub is equal to $NONE$. Also, $sector(A)$ trivially has the upper and lower boundary inheritance properties.

Now suppose A consists of at least two edges. Then, A is a series or parallel composition of two series-parallel graphs A_1 and A_2. Let $sector(A_1) = (lb_1, ub_1, \alpha_1, \beta_1)$ and $sector(A_2) = (lb_2, ub_2, \alpha_2, \beta_2)$, as computed by the algorithm. Let s_1 and t_1 be the source and sink respectively of A_1. Let s_2 and t_2 be the source and sink respectively of A_2. Let s and t be the source and sink respectively of A. There are two cases:

1. **A is a series-composition of A_1 and A_2**: Assume without loss of generality that $s_1 = t_2$, $s = s_2$ and $t = t_1$ (see Fig. 5.16). We consider two cases.

   Case 1: $sector(A_1)$ and $sector(A_2)$ are opposite (see Fig. 5.19(a)). $sector(A_1)$ and
   $sector(A_2)$ can be opposite under two situations (recall the definition of opposite given earlier). We give the proof for the situation when $sector(A_1)$ and $sector(A_2)$ are opposite because there is a line l' such that l' lies inside $sector(A_1)$ and is reverse included in $sector(A_2)$; The proof for the other situation is similar. Suppose l' is at an angle θ with the reference axis. Let l'' be a line that passes through o, and is at angle $\theta + 180$ with the reference axis (see Fig. 5.21(a, b)).

   Our claim is that $sector(A)$ is a disk, i.e., for every point p in the plane, there is a drawing of A in which s is placed at the origin o and t is placed at p. Because l'' lies properly inside $sector(A_2)$, there exists an angle μ such that line $l(\mu)$ lies inside $sector(A_2)$, and the angle between Line $l(\phi)$ and Line $l(\mu)$ is less than 180°. Let q be a point on Line $l(\phi)$ such that Line qp is at angle θ with the reference axis. Let $r_1 = qp$, and $r_2 = oq$. Let D_1 be a drawing of A_1 in which $angle(\vec{s_1 t_1}) = \theta$ and $s_1 t_1 = r_1$. Let D_2 be a drawing of A_2 in which $angle(\vec{s_2 t_2}) = \mu$ and $s_2 t_2 = r_2$. We can construct a drawing of A in which s is at o and t is at p by placing D_1 and D_2 such that s_1 and t_2 are placed at q, s_2 is placed at o, and t_1 is placed at p (see Fig. 5.21(b)).

   Because $sector(A)$ is a disk, it trivially has the upper and lower boundary inheritance properties.

   Case 2: $sector(A_1)$ and $sector(A_2)$ are not opposite (see Fig. 5.19(b, c)). This is illustrated in Fig. 5.21(c, d). Let p be a point that lies inside sector A. Because $sector(A)$ has width at most 180, basic geometric considerations show

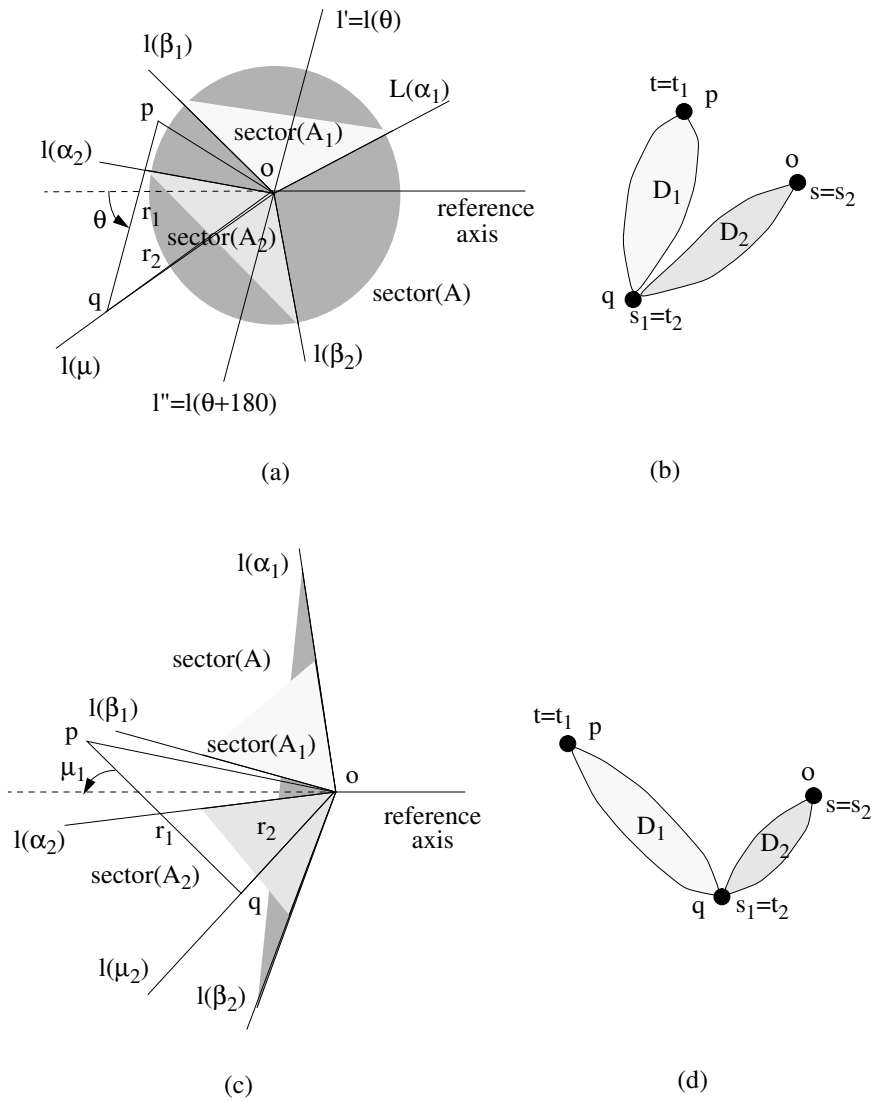

Figure 5.21: Proof of Theorem 19: Series-parallel graph A is a series-composition of series-parallel graphs A_1 and A_2. (a) *Case 1: sector*(A_1) and *sector*(A_2) are opposite; (b) Constructing a drawing of A for Case 1. (c) *Case 1: sector*(A_1) and *sector*(A_2) are not opposite; (d) Constructing a drawing of A for Case 2.

that, except for the case when $\alpha_1 = \beta_1 = \alpha_2 - 180 = \beta_2 - 180$, there exist angles μ_1 and μ_2, such that Line $l(\mu_1)$ lies inside *sector*(A_1), Line $l(\mu_2)$ lies inside *sector*(A_2), angle between $l(\mu_1)$ and $\vec{op}$ is less than 180, and angle between $l(\mu_2)$ and $\vec{op}$ is also less than 180. Let q be a point on $l(\mu_2)$ such that Line qp is at an angle μ_1 with the reference axis. Let $r_1 = qp$, and $r_2 = oq$. Let D_1 be a drawing of A_1 in which $angle(s_1\vec{t}_1) = \mu_1$ and $s_1t_1 = r_1$. Let D_2 be a drawing of A_2 in which $angle(s_2\vec{t}_2) = \mu_2$ and $s_2t_2 = r_2$. We can construct a drawing of A in which s is at o and t is at p by placing D_1 and D_2 such that s_1 and t_2 are placed at q, s_2 is placed at o, and t_1 is placed at p (see Fig. 5.21(d)).

It therefore, remains to be shown that the case $\alpha_1 = \beta_1 = \alpha_2 - 180 = \beta_2 - 180$ can not happen. Suppose for contradiction that it happens. Since $\alpha_1 = \beta_1$, we have that $lb_1 = ub_1 = CLOSED$. Since, $sector(A_1)$ has the lower boundary property, there is a vertex u_1 in A_1, such that edge s_1u_1 is at angle α_1 with the reference axis. Since $\alpha_2 = \beta_2$, we have that $lb_2 = ub_2 = CLOSED$. Since, $sector(A_2)$ has the upper boundary property, there is a vertex u_2 in A_2, such that edge u_2t_2 is at angle α_2 with the reference axis. Since $\alpha_2 = \alpha_1 + 180$, and $s_1 = t_2$ (see Fig. 5.21(d)), s_1 has two edges incident on it, namely, u_2s_1 and s_1u_1, that have zero angle between them, contradicting the fact that G has non-zero angles only. Hence, the case $\alpha_1 = \beta_1 = \alpha_2 - 180 = \beta_2 - 180$ can not happen.

If p is a point that does not lie inside $sector(A)$, then simple geometric considerations show that there do not exist any angles μ_1 and μ_2, as described above. Therefore, A does not have any drawing in which t is placed at p and s is placed at o.

Therefore, $sector(A)$ is computed correctly.

We now show that $sector(A)$ has the upper and lower boundary inheritance properties. We give the proof for the lower boundary inheritance property; The proof for the upper inheritance property is similar. lb is $CLOSED$ only if both lb_1 and lb_2 are $CLOSED$, and $\alpha = \alpha_1 = \alpha_2$. By inductive hypothesis, $sector(A_1)$ and $sector(A_2)$ both have the lower boundary inheritance properties. Therefore, there are vertices u_1 and u_2 such that edges u_1t_1 and s_2u_2 are at an angle α with the reference axis. Since $s = s_2$ and $t = t_1$, $sector(A)$ also has the lower boundary inheritance property. The proof for the upper boundary inheritance property is similar.

2. A **is a parallel-composition of** A_1 **and** A_2: Denote by $\cap(sector(A_1), sector(A_2)$, the sector with greatest width that is contained in both $sector(A_1)$ and $sector(A_2)$. We prove that $sector(A) = \cap(sector(A_1), sector(A_2))$.

   Because $s = s_1 = s_2$ and $t = t_1 = t_2$, in every drawing of A, $angle(\vec{s_1t_1}) = angle(\vec{s_2t_2}) = angle(\vec{st})$. Therefore, Line $l(angle(\vec{st}))$ lies inside both $sector(A_1)$ and $sector(A_2)$. Hence, $sector(A) \subseteq \cap(sector(A_1), sector(A_2))$.

   Let $l' = l(\theta)$ be a line lying inside $\cap(sector(A_1), sector(A_2)$. There exists a drawing D_1 of A_1 in which $angle(\vec{s_1t_1}) = \theta$. Also, there exists a drawing D_2 of A_2 in which $angle(\vec{s_2t_2}) = \theta$. Therefore, there exists a drawing of A, constructed from D_1 and D_2 in which $angle(\vec{st}) = \theta$. Hence, $\cap(sector(A_1), sector(A_2)) \subseteq sector(A)$.

   Therefore, $sector(A) = \cap(sector(A_1), sector(A_2)$. Hence, $sector(A)$ is computed correctly by the algorithm.

   As for lower and upper boundary properties, when $lb \neq NONE$ and $ub \neq NONE$, lb is either lb_1 or lb_2 or $NONE$, ub is either ub_1 or ub_2 or $NONE$, α is either α_1 or α_2, β is either β_1 or β_2 or $\beta_1 - 360$ or $\beta_2 - 360$. Since, $s = s_1 = s_2$ and $t = t_1 = t_2$, and $sector(A_1)$ and $sector(A_2)$ have the lower and upper boundary properties, it follows that $sector(A)$ also has the lower and upper boundary properties.

□

5.5 Area Requirement of Angle Graphs

In this section, we investigate the area requirement of angle graphs. *Area* of a drawing D of an angle graph A is defined as the area of the region bounded by the external face of A in D. We show that there exist certain angle graphs that require exponential area for any drawing.

Following theorem states the main result of this section:

Theorem 20 *For every $m \geq 1$, there exists an angle graph A_m with $3m$ vertices that requires area $\Omega(4^m)$ for any drawing.*

Proof: Angle graph A_m can be described recursively. A_1 is an equilateral triangle as shown in Fig. 5.22(a). A_m is constructed from A_{m-1} (see Fig 5.22(b)) by adding vertices u_m, v_m and w_m and edges (u_{m-1}, u_m), (v_{m-1}, u_m), (v_{m-1}, v_m), (w_{m-1}, v_m), (w_{m-1}, w_m) and (u_{m-1}, w_m) in the exterior of A_{m-1}. Each of the three newly added faces F_u, F_v and F_w is an equilateral triangle.

Let D_m be a drawing of A_m. We denote by $area(D_m)$, the area of D_m. For proving a lower bound on the area of D_m, we show that D_m has following properties:

1. **Shape-property**: Triangle $u_{m-1}v_{m-1}w_{m-1}$ is an equilateral triangle.

2. **Area-property**: $area(D_m) = 4\, area(D_{m-1})$, for $m \geq 2$.

D_1 trivially has the shape-property. Now suppose D_{m-1} has the shape-property. From plane geometry, because each of the faces F_u, F_v and F_w is an equilateral triangle, it is easy to see that D_m also has the shape-property.

As for the area-property, because each of the faces F_u, F_v and F_w is an equilateral triangle and D_{m-1} has the shape-property, it follows that the area of each of these three faces is equal to $area(D_{m-1})$. Therefore, for $m \geq 2$, $area(D_m) = 4\, area(D_{m-1})$.

Since $area(D_m) = 4\, area(D_{m-1})$, it follows that $area(D_m) = 4^{m-1}\, area(D_1)$. Assuming that $area(D_1)$ is a constant, we get that $Area(A_m) = \Omega(4^m)$.

Let $vertex(A_m)$ denote the number of vertices in A_m. We construct A_m from A_{m-1} by adding three vertices to it. Hence, $vertex(A_m) = vertex(A_{m-1}) + 3$, for $m \geq 2$. Since A_1 consists of three vertices, it is easy to see that the number of vertices in A_m is $3m$. □

Notice that each and every drawing of the angle graph A_m of Theorem 20 is a planar drawing. Therefore,

Corollary 7 *For every $m \geq 1$, there exists an angle graph A_m with $3m$ vertices that requires area $\Omega(4^m)$ for any planar drawing.*

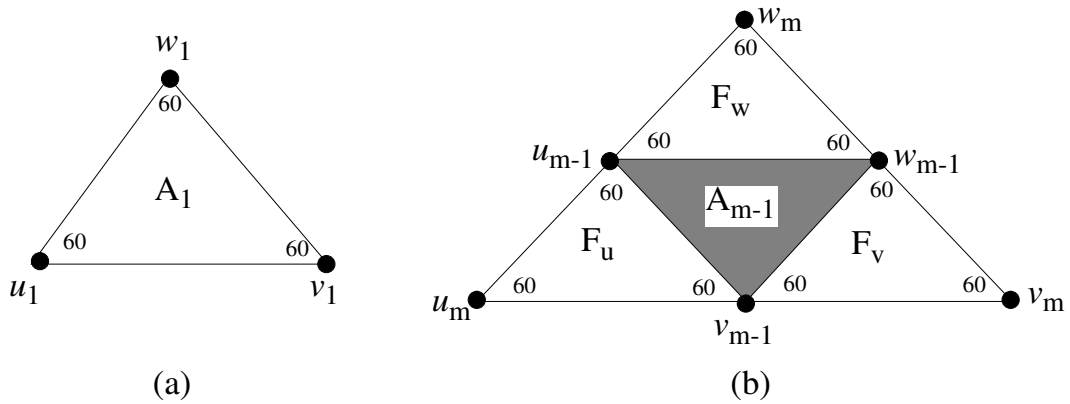

Figure 5.22: Angle graph A_m: (a) A_1; (b) Constructing angle graph A_m from A_{m-1}.

5.6 Multilayered Angle Graphs

We recall that a *multilayered* angle graph is an angle graph in which each edge is assigned to one of the several *layers* available. A *biplanar* angle graph is one whose edges are assigned to two layers only. A multilayered angle graph is *multiplanar* if it admits a drawing in which edges assigned to the same layer do not cross. The *multiplanarity testing problem* can stated as:

> Given a consistent multilayered angle graph G, test whether G is multiplanar or not, i.e., whether or not, G admits a drawing in which there is no crossing between the edges assigned to the same layer.

In this section, we show that the multiplanarity testing problem is NP-hard. In fact, we show something stronger, that even if we consider only consistent biplanar rectilinear angle graphs, the problem of testing them for biplanarity is NP-hard. We prove the NP-hardness of the biplanarity testing problem by reducing a well-known NP-hard problem, namely, the 3-SAT problem [50] given below, to the planarity testing problem:

> given a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables and a set $C = \{c_1, c_2, \ldots, c_m\}$ of clauses over X such that every clause has exactly three literals, is there a satisfying truth assignment for C?

Let T be an instance of the 3-SAT problem, with n variables and m clauses. We construct a bilayered rectilinear angle graph A(T), called the *associated angle graph* of T, in polynomial time such that T admits a satisfying truth-assignment if and only if A(T) is biplanar.

Let the two layers to which the edges of A(T) are assigned, be called *red* and *blue* layers. The edges of A(T) assigned to the red and blue layers are called *red* and *blue* edges respectively.

The rest of this section is organized as follows. In Section 5.6.1, we describe some gadgets (which are consistent bilayered rectilinear angle graphs) that are used for the

construction of A(T). In Section 5.6.2, we describe a consistent bilayered rectilinear angle graph called a *Completed Course*. This graph consists of some of these gadgets and is used to construct A(T). Finally, in Section 5.6.3, we completely describe A(T).

5.6.1 Some Gadgets

We now describe some consistent bilayered rectilinear angle graphs that are used for constructing A(T).

- **Subcourse:** A subcourse S is shown in Fig. 5.23(a). Vertex u_j, where $0 \leq j \leq 2m+2$, is called the j^{th} *input* vertex of S. Vertex v_j, where $0 \leq j \leq 2m+2$, is called the j^{th} *output* vertex of S. Edge (t_j, t_{j+1}), where $1 \leq j \leq m$, is called the j^{th} *reference edge* of S. Edge (s_j, s_{j+1}), where $1 \leq j \leq m$ is called the j^{th} *value edge* of S. Edge (a_{2j-1}, a_{2j}), where $1 \leq j \leq m$, is called the j^{th} *inner* edge of S. Edges (u_{2j}, s_{j+1}), (s_{j+1}, a_{2j}), and (a_{2j}, v_{2j}), where $1 \leq j \leq m$, are the j^{th} *value-separation* edges of S. Edges (u_{2j-1}, t_j), (t_j, a_{2j-1}), and (a_{2j-1}, v_{2j-1}), where $1 \leq j \leq m$, are the j^{th} *reference-separation* edges. Each reference, value, inner and separation edge is either blue or red (as will be described later in Section 5.6.3). Edge (v_{2j-1}, v_{2j}), where $1 \leq j \leq m$, is called the j^{th} *output* edge of S and is red. Edge (u_{2j-1}, u_{2j}), where $1 \leq j \leq m$, is called the j^{th} *input* edge of S and is red. Edges (t_0, t_1) and (t_{m+1}, t_{m+2}) are both red. Edge (s_{m+1}, s_{m+2}) is red. All the remaining edges of S are blue.

  Let D be a drawing of S. Assume without loss of generality that the reference and value edges of A(T) are drawn vertically in D (see Fig. 5.23(a)). The line-segment $\mathrm{l}_v = s_1 s_{m+1}$ is called the *value line-segment* of S in D. The line-segment $\mathrm{l}_r = t_1 t_{m+1}$ in D is called the *reference line-segment* of S in D. In D, either l_v is to the right of l_r (Fig. 5.23(b)), or l_v is to the left of l_r (see Fig. 5.23(c)), or both l_v and l_r overlap. S is *left heavy* (*right heavy* , *balanced*, respectively) if l_v is left of l_r (right of l_r, overlapping with l_r, respectively). As we will see later, each subcourse S corresponds to a variable x, and the left heavy (right heavy) "state" of S corresponds to x being `true` (`false`) in a satisfying truth-assignment of T.

- **Starting and Ending Location:** A starting location L is shown in Fig. 5.24(a) and an ending location L' is shown in Fig. 5.24(b). Vertex c_j (c'_j), where $1 \leq j \leq m$, is called the j^{th} *starting* (*ending*) vertex of L (L'). Vertex w_j (w'_j), where $0 \leq j \leq 2m+2$, is called the j^{th} *separation* vertex of L (L'). All the edges of both L and L' are blue.

- **Hurdle:** A hurdle H is shown in Fig. 5.24(c). Vertex g_j, where $0 \leq j \leq 2m+4$, is the j^{th} *input* vertex of H. Vertex h_j, where $0 \leq j \leq 2m+2$, is called the j^{th} *output* vertex of H. Each edge (h_j, h_{j+1}), where $0 \leq j \leq 2m+2$, is red. The other edges of H are all blue.

- **Runner:** A runner R (see Fig. 5.24(d)) is simply a connected sequence of 13 edges that alternates between red and blue. The angle between consecutive edges is 180°. Vertex a is the *first end-point* of R and vertex a' is its *last end-point*. The edges

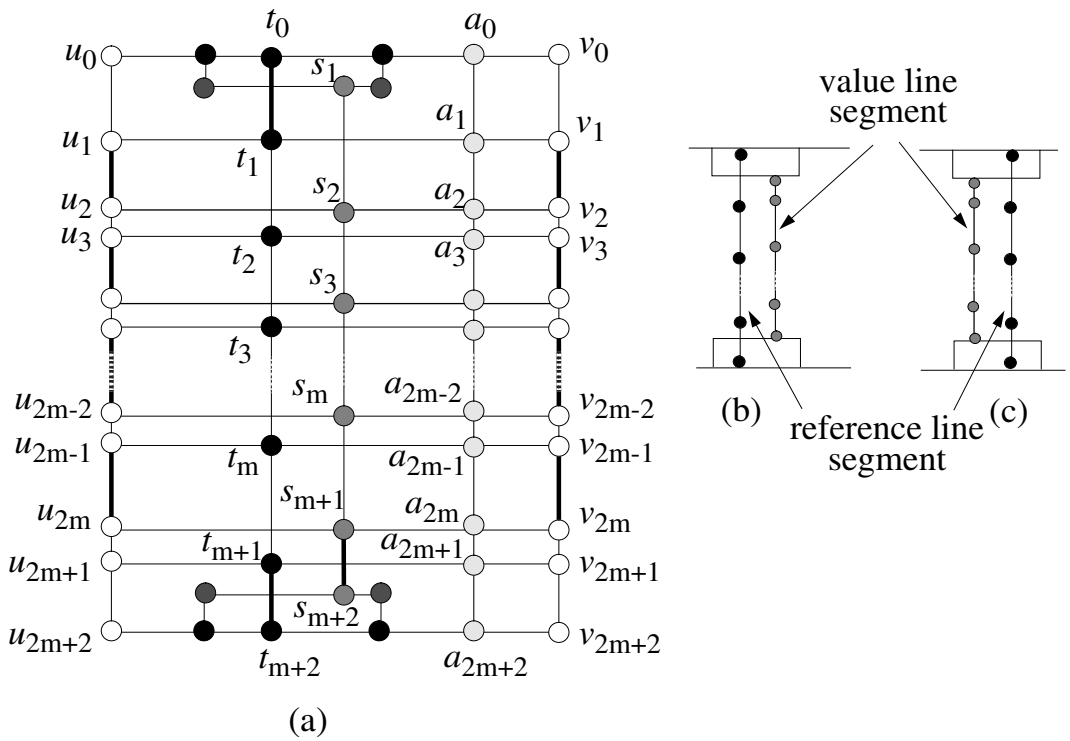

Figure 5.23: (a) A subcourse S; (b) A right heavy subcourse; (c) A left heavy subcourse. In the figure, Except for the reference, value, inner and separation edges, an edge is shown as a thick line if it is a red edge, and is shown as a thin line if it is a blue edge.

of R incident on a and a' are called the *first* and *last* edges of R respectively. The first edge of R is blue. As we will see later, each runner corresponds to a clause.

5.6.2 Completed Course

We now describe a consistent bilayered rectilinear angle graph called *completed course*.

A *course* K_i is constructed recursively as follows:

1. K_1 is the subcourse S_1. S_1 is called the $first$ subcourse of K_1.
2. K_i is constructed from K_{i-1} by joining the k^{th} output vertex of the $i-1^{th}$ subcourse S_{i-1} of K_{i-1} with the k^{th} input vertex of subcourse S_i, using a blue edge. S_i is called the i^{th} subcourse of K_i. For each j, such that $1 \leq j \leq i-1$, the j^{th} subcourse of K_{i-1} is also called the j^{th} subcourse of K_i.

A *completed course* G_n, consists of a course K_n, a hurdle H, a starting location L and an ending location L'. In G_n, the j^{th} separation vertex of L is connected with the j^{th} input vertex of the first subcourse of K_n, with a blue edge. Also, the j^{th} output vertex of the n^{th} subcourse of K_n is connected with the j^{th} input vertex of H, with a

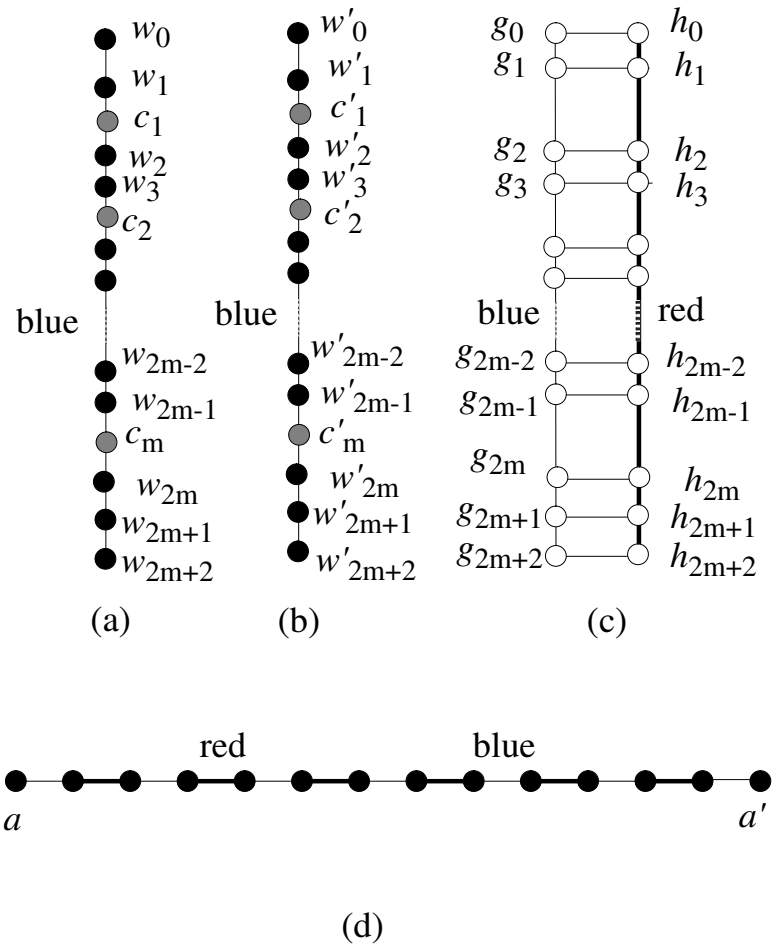

Figure 5.24: (a) A Starting Location; (b) An Ending Location; (c) A Hurdle; (d) A Runner. Blue edges are shown as thin lines and red edges are shown as thick lines.

blue edge, and the j^{th} output vertex of H is connected with the j^{th} separation vertex of L', also with a blue edge. Fig. 5.25(a) shows a high-level view of G_n.

5.6.3 Angle Graph A(T)

We are now in a position to describe A(T). As shown in Fig. 5.25(b), A(T) consists of m runners $R_1, R_2, \ldots, R_m$ (one for each clause of T), and a completed course G consisting of a hurdle H, starting and ending locations L and L' respectively, and a course K_n consisting of n subcourses $S_1, S_2, \ldots, S_n$ (one for each variable of T). Runner R_j is called the *representative* of the clause c_j in A(T). For each R_j, its first end-point coincides with j^{th} starting vertex of L, and its last end-point coincides with j^{th} ending vertex of L'. Subcourse S_i is called the *representative* of the variable x_i in A(T).

The value, reference, inner and separation edges of Subcourse S_i are assigned to the red and blue layers as follows: If variable x_i does not occur in clause c_j or occurs both negated and un-negated, then the j^{th} reference edge, the j^{th} value edge, and the j^{th} inner

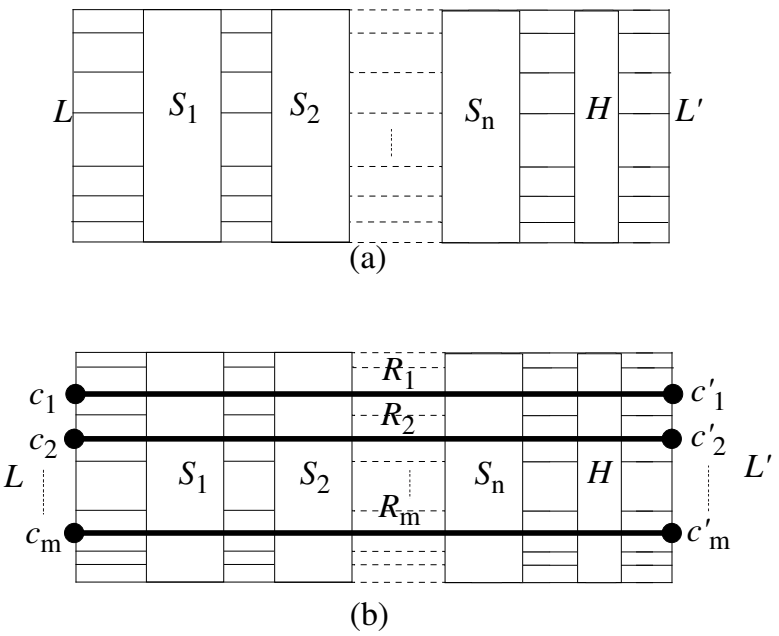

Figure 5.25: A high level view of (a) Completed Course G_n; (b) Angle graph A(T).

edges are all red, and all the j^{th} separation edges are blue. If variable x_i occurs in clause c_j either as negated or un-negated but not as both, then the j^{th} inner edge is blue. If variable x_i occurs negated in c_j then the j^{th} reference edge is red, j^{th} value edge is blue, all the j^{th} value-separation edges are blue, and all the j^{th} reference-separation edges are red. If variable x_i occurs un-negated in c_j then the j^{th} reference edge is blue, j^{th} value edge is red, all the j^{th} value-separation edges are red, and all the j^{th} reference-separation edges are blue. This coloring scheme is summarized in Table 5.1.

	Color in S_i of its				
	j^{th} reference edge	j^{th} value edge	j^{th} inner edge	j^{th} value separation edges	j^{th} reference separation edges
$x_i \notin c_j$ and $\bar{x}_i \notin c_j$	red	red	red	blue	blue
$x_i \in c_j$ and $\bar{x}_i \in c_j$	red	red	red	blue	blue
$\bar{x}_i \in c_j$ and $x_i \notin c_j$	red	blue	blue	blue	red
$x_i \in c_j$ and $\bar{x}_i \notin c_j$	blue	red	blue	red	blue

Table 5.1: The color of the j^{th} reference, value, inner and separation edges of Subcourse S_i.

Fig. 5.26 shows an example.

Lemma 21 *Let T be an instance of the 3SAT problem, with n variables and m clauses. Its associated angle graph $A(T)$ has $O(nm)$ vertices and edges. $A(T)$ can be constructed in $O(nm)$ time.*

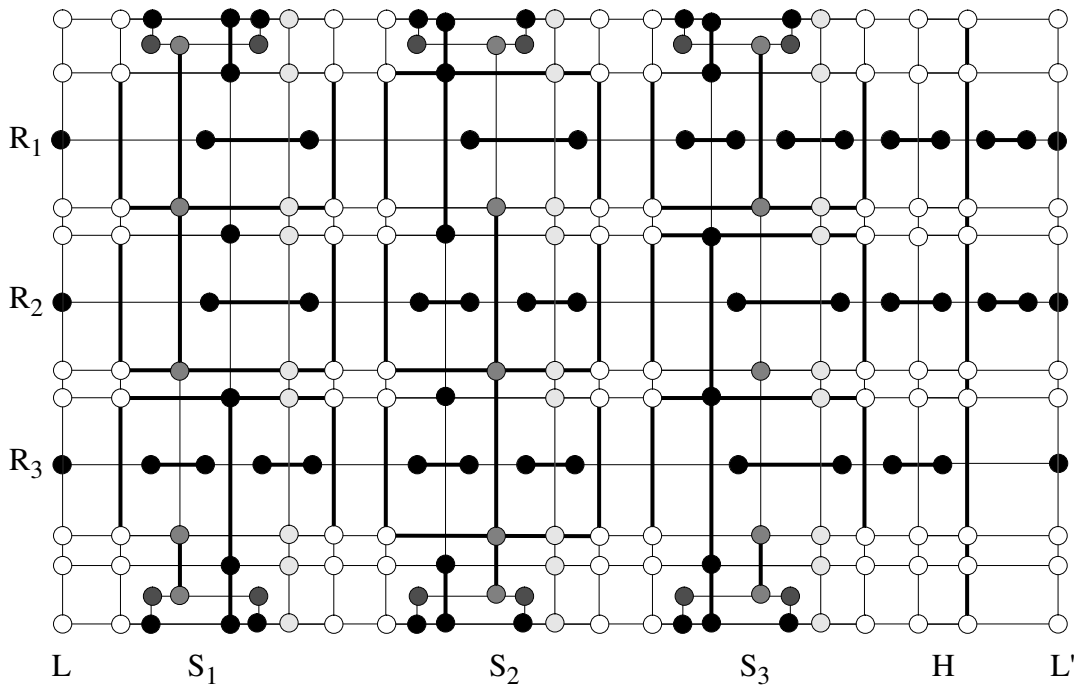

Figure 5.26: A biplanar drawing D of the associated angle graph A(T) of an instance T of the 3SAT problem, where T consists of the variables x_1, x_2 and x_3, and the clauses $(x_1, \bar{x_2}, x_3)$, $(x_1, x_2, \bar{x_3})$ and $(\bar{x_1}, x_2, \bar{x_3})$. D corresponds to the truth-assignment x_1 =`true`, x_2 =`false` and x_3 =`false`. The red edges are shown as thick lines, and the blue edges are shown as thin lines.

Proof: A(T) has n subcourses, one for each variable, m runners, one for each clause, a starting location L, an ending location L' and a hurdle H. Graphs L, L', and H, each have $O(m)$ vertices and edges. Each subcourse has $O(m)$ vertices and edges. Each runner has $O(n)$ vertices and edges. Therefore A(T) has a total of $O(nm)$ vertices and edges. A(T) can easily be constructed in $O(nm)$ time. □

Lemma 23 shows that T admits a satisfying truth-assignment if and only if A(T) is biplanar. It uses the following correspondence between a biplanar drawing D of A(T) and a satisfying truth-assignment Ψ of T: Variable x_i is `true` in Ψ if and only if Subcourse S_i is left heavy in D.

We need to do some groundwork before we prove Lemma 23. First some definitions. We assume a right-handed three-dimensional cartesian coordinate system. We denote by $x(p)$, $y(p)$ and $z(p)$, the x-, y- and z-coordinates of a point p. A point p is to the *left* (*right*) of another point q if $x(p) < x(q)$ ($x(p) > x(q)$). A vertical line l is to the *left* (*right*) of another vertical line l' if l is drawn at an x-coordinate less than (greater than) the x-coordinate at which l' is drawn. Let D be a biplanar drawing of A(T). Assume without loss of generality, that the runners of A(T) are drawn horizontally in D and the starting location L is drawn to the left of the ending location L'. Let R be a runner of A(T). Let $e = (u, v)$ be an edge of R. Notice that e is drawn horizontally in D. In D, the *right* (*left*) end-point of e is the one that is placed to the right (left) of its

other end-point. Assume without loss of generality that u is the left end-point of e in D. Edge e *spans* a vertically drawn edge $e' = (u', v')$ if $x(u) \leq x(u') \leq x(v)$ and either $y(u') \leq y(u) \leq y(v')$ or $y(v') \leq y(u) \leq y(u')$. e is the *incoming* (*outgoing*) edge of R in Subcourse S_i if it spans the j^{th} input (output) edge of S_i. R *spends* d edges to *span* Subcourse S_i if d consecutive vertices $v_1, v_2, \ldots, v_d$ of R are placed such that for each v_k, following holds: (a) v_k is placed to the left of the output vertices of S_i, and (b) v_k is placed to the right of the output vertices of S_{i-1} if $i \geq 2$. R *spends* d edges to *span* Hurdle H if d consecutive vertices $v_1, v_2, \ldots, v_d$ of R are placed such that for each v_k, following holds: (a) v_k is placed to the left of the output vertices of H, and (b) v_k is placed to the right of the output vertices of S_n.

Subcourse S_i is said to be *neutral* to Runner R_j in D if variable x_i either does not occur in clause c_j, or occurs both negated and un-negated in clause c_j. If variable x_i occurs either only negated or only un-negated in clause c_j, then Subcourse S_i is said to be *compatible* (*not compatible*, otherwise) to Runner R_j in D if:

1. Variable x_i occurs only negated in clause c_j and S_i is right heavy in D, or
2. Variable x_i occurs only un-negated in clause c_j and S_i is left heavy in D.

Lemma 22 *Let D be a biplanar drawing of A(T). Each subcourse of A(T) is either left heavy or right heavy in D.*

Proof: Assume for contradiction, that there is a subcourse S_i of A(T) that is balanced in D, i.e., the reference line-segment and the value line-segment of S_i overlap in D. Consider a runner R_j of A(T) such that variable x_i occurs in clause c_j. R_j consists of a sequence of alternating red and blue edges. Let e be the edge of R_j that spans the j^{th} reference as well as the j^{th} value edge of S_i (notice that these two edges are overlapping, therefore e spans both of them). Since the j^{th} reference and j^{th} value edge of S_i edges have different colors, it follows that one of them has the same color as e, contradicting the assumption that D is a biplanar drawing. Therefore, S_i is either left heavy or right heavy in D. □

Lemma 23 *A(T) is biplanar if and only if T admits a satisfying truth-assignment.*

Proof:

If: Suppose T admits a satisfying truth-assignment Ψ. We construct a biplanar drawing of D of A(T) (see Fig 5.26 for an example). In D, Subcourse S_i is left heavy if variable x_j is `true` and is right heavy if variable x_j is `false`. The runners of A(T) are drawn such that, for every Subcourse S_i, and Runner R_j, following holds:

1. If S_i is neutral to R_j, then R_j spends exactly 0 edges to span S_i.
2. If S_i is compatible to R_j, then R_j spends exactly 2 edges to span S_i,
3. If S_i is not compatible to R_j, then R_j spends exactly 4 edges to span S_i,

4. R_j spends exactly 2 edges to span Hurdle H.

5. The first edge of R_j is the incoming edge of R_j in S_1.

6. The outgoing edge of R_j in S_i is also the incoming edge of R_j in S_{i+1}.

Because Ψ is a satisfying truth-assignment, for each runner R_j, there exists a subcourse S_i, such that S_i is compatible with R_j. Hence, in D, each runner spends a maximum of 12 edges for spanning all the subcourses and the hurdle H (see Fig 5.26).

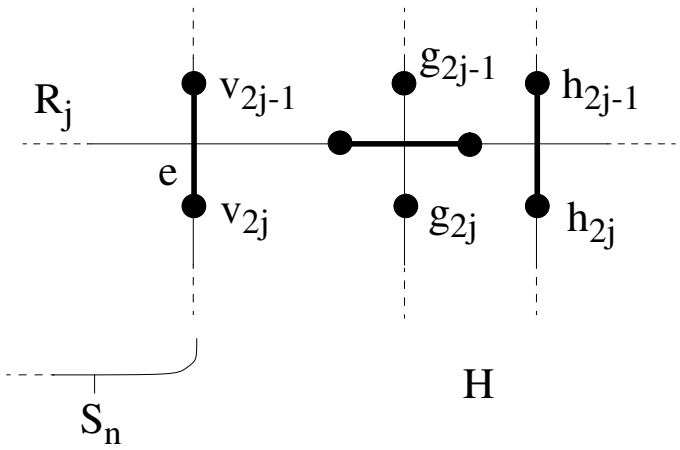

Figure 5.27: Runner R_j spends two edges to span Hurdle H in any biplanar drawing. e is the j^{th} output edge of S_n. Notice that the outgoing edge of R_j in S_n is blue in any biplanar drawing. The red edges are shown as thick lines and the blue edges are shown as thin lines.

We now show that each runner R_j can be drawn as described above. Because, D is biplanar, and the input and output edges of S_n are red, the outgoing edge R_j in S_n is blue. Therefore, as shown in Fig. 5.27, R_j can be drawn such that it spends exactly 2 edges to span Hurdle H.

Now we prove our claim about the number of edges spent by the runners for spanning the subcourses of A(T). Again, because D is biplanar, the incoming edge of R_j in S_i is blue. R_j spans the j^{th} input, j^{th} value, j^{th} reference, j^{th} inner and j^{th} output edges of S_i, in that order from left to right, if S_i is left heavy. On the other hand, R_j spans the j^{th} input, j^{th} reference, j^{th} value, j^{th} inner and j^{th} output edges of S_i, in that order from left to right, if S_i is right heavy. We consider the following cases:

1. S_i **is neutral to** R_j: Since the j^{th} reference, j^{th} value, j^{th} input, j^{th} output and j^{th} inner edges of S_i are all red, R_j can easily be drawn without any crossings between same colored edges such that it spends 0 edges for spanning S_i (See Fig 5.28(a)).

2. S_i **is compatible with** R_j: In both the cases when S_i is left heavy or right heavy, R_j spans in S_i, two red, two blue and one red edge, in that order from left to right. R_j can easily be drawn without any crossings between same colored edges such that it spends only 2 edges for spanning S_i (See Fig 5.28(b), where only $\bar{x_i}$ occurs in c_j and S_i is right heavy).

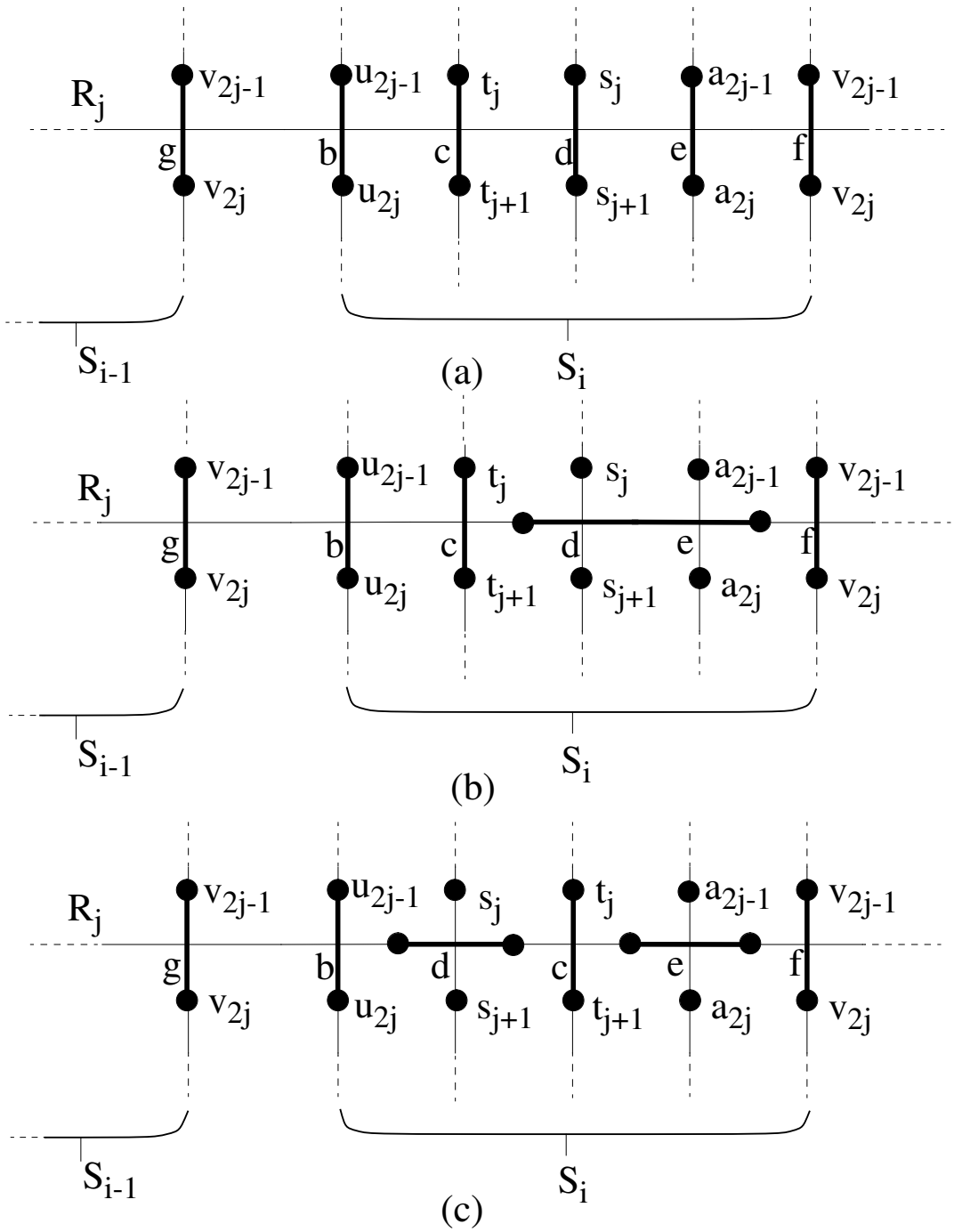

Figure 5.28: Subcourse S_i spanned by a Runner R_j in a biplanar Drawing. g is the j^{th} output edge of S_{i-1}. b, c, d, e and f are the j^{th} input, reference, value, inner and outer edges respectively of S_i. The red edges are shown as thick lines and the blue edges are shown as thin lines. (a) S_i is neutral to R_j: R_j spends 0 edges; (b) S_i is compatible with R_j: R_j spends 2 edges. An example is when variable x_i occurs only negated in c_j and S_i is right heavy; (c) S_i is not compatible with R_j: R_j spends 4 edges. An example is when variable x_i occurs only negated in c_j and S_i is left heavy.

3. S_i **is not compatible with** R_j: In both the cases when S_i is left heavy or right heavy, R_j spans in S_i, one red, one blue, one red, one blue and one red edge, in that order from left to right. R_j can easily be drawn without any crossings between same colored edges such that it spends only 4 edges for spanning S_i (See Fig 5.28(c), where only $\bar{x_i}$ occurs in c_j and S_i is left heavy).

Only If: Let D be a biplanar drawing of A(T) (see Fig 5.26 for an example). From D, we construct a truth-assignment Ψ of T. From Lemma 22, each Subcourse S_i is either left heavy or right heavy. In Ψ, a variable x_i is `true` if and only if Subcourse S_i is left heavy.

We claim that Ψ is a satisfying truth-assignment of T. A clause is trivially satisfiable if a variable occurs in it both negated and un-negated. Therefore, from the definition of compatibility, this is equivalent to claiming that for each runner R_j of A(T), such that no variable occurs both negated and un-negated in R_j, there exists a subcourse of A(T) that is compatible with R_j in D. Suppose for contradiction, there is a runner R_j of A(T) with no compatible subcourse in D. We show that, in D, R_j spends a total of at least 14 edges to span all the subcourses and the hurdle of A(T). Suppose a variable x_i occurs in Clause c_j. Consider Subcourse S_i. From Lemma 22, S_i is either left heavy or right heavy in D. Therefore, in D, R_j either spans the j^{th} input, j^{th} value, j^{th} reference, j^{th} inner and j^{th} output edges of S_i, in that order from left to right (when S_i is left heavy), or R_j spans the j^{th} input, j^{th} reference, j^{th} value, j^{th} inner and j^{th} output edges of S_i, in that order from left to right (when S_i is right heavy). In both the cases, since S_i is not compatible with R_j, R_j spans in S_i, one red, one blue, one red, one blue and one red edge, in that order from left to right. Since D is biplanar, the incoming edge of R_j in S_i is blue. R_j therefore, needs to spend at least 4 edges to span S_i (See Fig 5.28(c), where only $\bar{x_i}$ occurs in c_j and S_i is left heavy). Because clause c_j has exactly three literals, R_j spends at least 12 edges to span all the subcourses of A(T). Because D is biplanar, R_j spends at least 2 more edges to span H (see Fig. 5.27). Hence, in D, R_j spends a total of at least 14 edges to span all the subcourses and the hurdle of A(T). This is not possible since R_j has only 13 edges. Therefore, R_j has at least one compatible subcourse in D. □

We are now ready to present the main theorem of this section

Theorem 21 *The problem of testing whether a consistent bilayered angle graph is biplanar is NP-hard.*

Proof: Let T be an instance of the 3SAT problem. From Lemmas 21 and 23, we can construct the associated angle graph A(T) of T in $O(nm)$ time, such that A(T) is biplanar if and only if T admits a satisfying truth-assignment. Hence, given a consistent bilayered angle graph, testing whether it is biplanar or not, is NP-hard. □

Following corollary is immediate-

Corollary 8 *The problem of testing whether a multilayered angle graph is multiplanar is NP-hard.*

5.7 Discussion

A (general) graph can be tested for planarity in linear time [14, 63]. Similarly, the equivalent multiplanarity problem for general graphs is also solvable in linear time: A multilayered (general) graph is multiplanar if and only if each subgraph consisting of the edges assigned to the same layer is planar [110]. We have however shown both of these problems to be NP-hard for angle graphs. Also every graph can be drawn in quadratic area [109], whereas we have shown that certain angle graphs require exponential area. Thus angle graphs are more "difficult" then general graphs in some sense, at least for the problems that we have studied. It may be interesting to do a further comparative study of angle graphs and general graphs.

One important open problem is to give an improved characterization of a consistent angle graph so that testing an angle graph for consistency can be done faster. A possible approach could be to express the consistency of an angle graph in terms of the consistency of its cycles. The following characterization of a consistent cycle may be useful: Let $C = \{v_0, v_1, \ldots, v_m, v_0\}$ be a cycle. Introduce variables $X_{i,(i+1) \bmod (m+1)}$ and $Y_{i,(i+1) \bmod (m+1)}$ for every edge $(v_i, v_{(i+1) \bmod (m+1)})$ of C. Intuitively $X_{i,j}$ ($Y_{i,j}$) corresponds to the difference in the x-coordinates (y-coordinates) of the vertices v_i and v_j. Let $s_{i,j}$ be the slope of the edge (v_i, v_j). We have that C is consistent if and only if there is an assignment of $X_{i,j}$'s such that $\sum_{0 \leq i \leq m} \alpha_{i,(i+1) \bmod (m+1)} X_{i,(i+1) \bmod (m+1)} = 0$, $\sum_{0 \leq i \leq m} \beta_{i,(i+1) \bmod (m+1)} Y_{i,(i+1) \bmod (m+1)} = 0$, and for every edge (v_i, v_j), $Y_{i,j} = |s_{i,j}| X_{i,j}$. Here, $\alpha_{i,j}$ is 1 if the edge (v_i, v_j) is at an angle between $-90°$ and $90°$ with the positive x-axis and is -1 otherwise, and $\beta_{i,j}$ is 1 if the edge (v_i, v_j) is at an angle between $0°$ and $180°$ with the positive x-axis and is -1 otherwise.

Another open problem of great theoretical and practical importance is applying the techniques of Tamassia's bend-minimization algorithm [99] to construct planar drawings of general planar graphs, not just orthogonal planar graphs.

Finally, although we have shown NP-hardness of the planarity testing problem for general angle graphs, it is useful to characterize angle graphs for which this problem can be solved in polynomial time.

Chapter 6

Rectilinear Planarity Testing

6.1 Introduction

In this chapter, we study the problem of testing a graph for rectilinear planarity i.e., testing whether it admits a planar rectilinear drawing. A drawing is *planar* if no two edges cross. A graph is *planar* if it admits a planar drawing. An *orthogonal* drawing maps each edge into a chain of horizontal and vertical segments. A *rectilinear* drawing is an orthogonal straight-line drawing, i.e., a drawing where every edge is either a horizontal or a vertical segment. A graph is *rectilinear planar* if it admits a planar rectilinear drawing. Fig. 6.1 shows a planar rectilinear drawing.

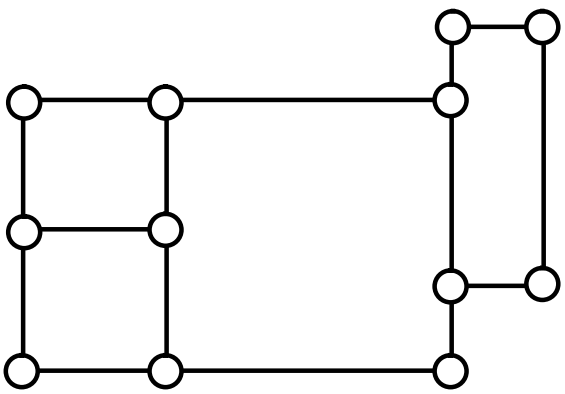

Figure 6.1: Example of a planar rectilinear drawing of a graph.

Testing for Rectilinear planarity is a fundamental problem in the display of circuit schematics and entity-relationship diagrams. This problem has challenged the researchers for many years. In this chapter we show that the following problem is NP-complete:

Rectilinear planarity testing: testing whether a graph is rectilinear planar.

We also show that it is NP-hard to approximate the minimum number of bends in a

planar orthogonal drawing of an n-vertex graph with an $O(n^{1-\epsilon})$ error, for any $\epsilon > 0$.

Shiloach [95] and Valiant [109] show that any planar graph of degree at most 4 admits a planar orthogonal drawing. Vijayan and Wigderson [111] study structural properties of rectilinear planar drawings. From their results, the membership of rectilinear planarity testing in NP is easy to establish. Storer [97], Tamassia and Tollis [102], Liu, Marchioro, Morgana, Petreschi, and Simeone [79, 80, 81, 78], Even and Granot [45], and Biedl and Kant [68, 13] give various techniques for constructing planar orthogonal drawings with $O(n)$ bends. Tamassia [99] gives an $O(n^2 \log n)$-time algorithm that constructs a planar orthogonal drawing with the minimum number of bends for an embedded planar graph. Di Battista, Liotta, and Vargiu [33] give polynomial time algorithms for minimizing bends in planar orthogonal drawings of series-parallel and cubic graphs. The latter two results show that rectilinear planarity testing can be done in polynomial time for a fixed embedding or for special classes of graphs.

Our proof techniques are based on a two-phase reduction from the known NP-complete problem NOT-ALL-EQUAL-3-SAT. In the first phase, we reduce NOT-ALL-EQUAL-3-SAT to an auxiliary undirected flow problem. This reduction is same as the one we use for showing the NP-completeness of the upward planarity testing problem in Chapter 2. In the second phase, we reduce this undirected flow problem to the rectilinear planarity testing of a special class of digraphs. The latter reduction is interesting in its own and provides new insights on the characterization by flow networks of the angles formed by the edges of orthogonal drawings [33, 99]. Most of the results and techniques presented in this chapter can also be found in [57, 59].

The rest of this chapter is organized as follows. Preliminary definitions and results are provided in Section 6.2. In Section 6.3 we describe the reduction from NOT-ALL-EQUAL-3-SAT to rectilinear planarity testing. Conclusive remarks are given in Section 6.4.

6.2 Preliminaries

We assume standard concepts and definitions on NP-completeness [50]. Our results use reductions from the following well-known NP-complete problem:

NOT-ALL-EQUAL-3-SAT Given a set of clauses with three literals each, is there a truth assignment such that each clause has at least one true literal and one false literal?

An *embedding* of a planar graph is the collection of circular permutations of the edges incident upon each vertex in a planar drawing of the graph. An *embedded graph* is a planar graph equipped with an embedding. We do not distinguish between a graph and its embedding if the embedding is unique and the meaning is clear from the context.

We denote by $G - \{v\}$ the subgraph of graph G obtained by removing vertex v and its incident edges from G.

The *angles* of an embedded graph are the pairs of consecutive edges incident on the same vertex. Such angles are mapped to geometric angles in a straight-line drawing of the graph.

A *rectilinear embedding* of a graph G is an embedding of G where each angle is assigned a label in the set $\{1,2,3,4\}$, such that there exists a rectilinear drawing of G where each angle labeled ℓ in the embedding measures $\ell\pi/2$ in the drawing. Each rectilinear embedding has a unique external face.

6.2.1 Rectilinear Embedding

In this section we give Lemma 24 that will be extensively used in the proofs.

Let G be an undirected graph of degree at most 4. Consider an assignment of labels from the set $\{1,2,3,4\}$ to the angles of an embedding of G. For a face f of the embedding, let $N_i(f)$ be the number of angles of f with label i. Face f is said to be *consistently rectilinearly assigned* if:

$$2 \cdot N_4(f) + N_3(f) - N_1(f) = \begin{cases} -4 & \text{if } f \text{ is an internal face,} \\ +4 & \text{if } f \text{ is the external face.} \end{cases}$$

An assignment of labels to the angles of an embedding of a graph G is a *consistent rectilinear assignment* if:

- the sum of the labels of the angles around each vertex is 4; and
- each face is consistently rectilinearly assigned.

The following lemma is an immediate consequence of the results in [99, 111].

Lemma 24 *An embedding of a graph G can be extended to a rectilinear embedding if and only if it admits a consistent rectilinear assignment of labels to its angles.*

6.2.2 Tendrils and Wiggles

We now define two kinds of graphs, namely *rectilinear tendrils* and *rectilinear wiggles* that will be used as gadgets in our reductions.

We show in Fig. 6.2 an undirected graph called *rectilinear tendril T_k*, which also has two designated poles denoted by s and t. We also define rectilinear tendril T_0 as a a graph consisting of a single edge.

Lemma 25 *Rectilinear tendril T_k is rectilinear planar and admits exactly four rectilinear embeddings.*

Proof: The proof has the same flavor as the proof of Lemma 2 of Chapter 2. We use induction on k and apply Lemma 24. Rectilinear tendril T_1 is shown in Fig 6.3(a). As shown in Fig. 6.3(c), T_k can be constructed from T_{k-1} by removing vertex s_{k-1} and

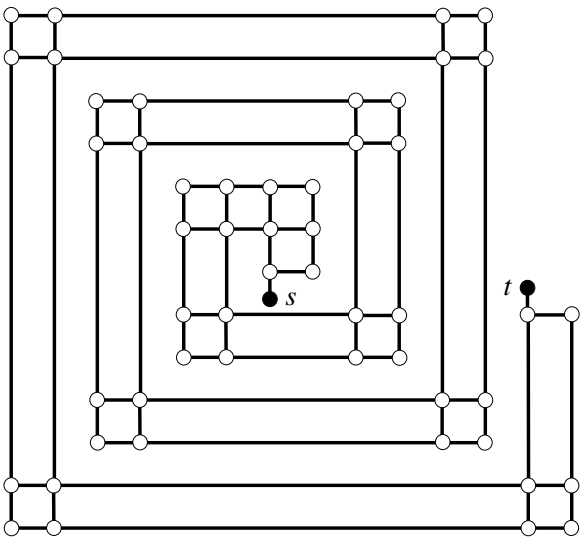

Figure 6.2: Example of rectilinear tendril T_3.

identifying its edge e_{k-1} with the edge f_k of the rectilinear planar graph H_k shown in Fig. 6.3(b). □

We show in Fig. 6.5 *rectilinear wiggle* W_k, which is a chain of $4k+1$ edges.

The *contribution* of a rectilinear tendril (or wiggle) to a face containing it is the number of angles of the tendril (or wiggle) labeled 3 minus the number of angles labeled 1 that lie in the face. Rectilinear tendril T_k contributes $4k$, $4k+1$, or $4k+2$ to one face, and the opposite value to the other face. Rectilinear wiggle W_k contributes to one face any value between 0 and $4k$, and the opposite value to the other face.

6.3 Rectilinear Planarity Testing

In this section we show how to reduce NOT-ALL-EQUAL-3-SAT to rectilinear planarity testing. The reduction is done in two steps. Starting from an instance $\mathcal{S}$ of NOT-ALL-EQUAL-3-SAT, in the first step, we construct a planar switch-flow network $\mathcal{P}$ in polynomial time such that Instance $\mathcal{S}$ is satisfiable if and only if network $\mathcal{P}$ admits a feasible flow. In the second step, we reduce the problem of computing a feasible flow in the network $\mathcal{P}$ to the problem of testing the rectilinear planarity of a suitable graph $\mathcal{G}$.

The construction of network $\mathcal{P}$ is described in detail in Section 2.3 of Chapter 2. We restate Theorem 1 of Section 2.3 of Chapter 2 here for the ease of reference.

Theorem 22 (Theorem 1, Section 2.3, Chapter 2) *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses, the associated planar switch-flow network $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges, and can be constructed in $O(n^2m^2)$ time. Instance $\mathcal{S}$ is satisfiable if and only if network $\mathcal{P}$ admits a feasible flow. Also, given a feasible flow for $\mathcal{P}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^2m^2)$.*

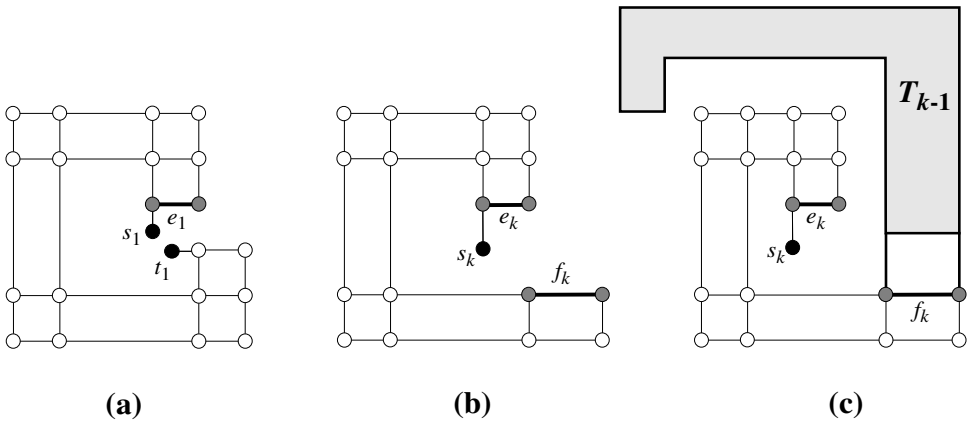

Figure 6.3: (a) Rectilinear Tendril T_1. (b) Graph H_k. (c) Constructing T_k from T_{k-1}.

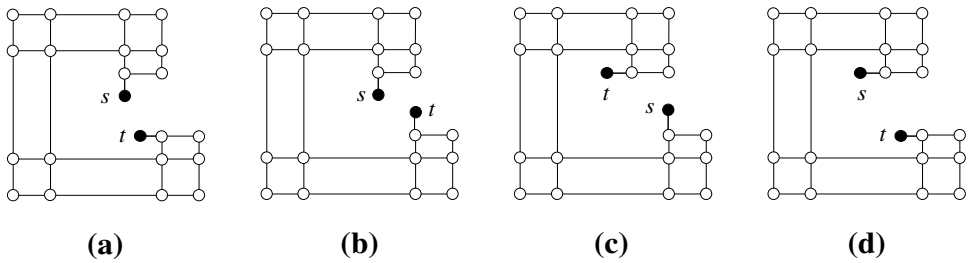

Figure 6.4: The four different rectilinear embeddings of rectilinear tendril T_1.

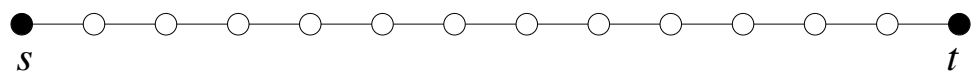

Figure 6.5: Example of a rectilinear wiggle W_3.

We now show how to reduce the problem of computing a feasible flow in the planar switch-flow network $\mathcal{P}$ associated with a NOT-ALL-EQUAL-3-SAT instance $\mathcal{S}$ to the problem of testing the rectilinear planarity of a suitable graph $\mathcal{G}$. The construction of graph $\mathcal{G}$ is carried out in several stages, where at each stage an intermediate graph is produced.

Given an instance $\mathcal{S}$ of NOT-ALL-EQUAL-3-SAT with n variables and m clauses, let $\mathcal{P}$ be the associated planar switch-flow network with parameter $\theta = 32nm$.

Let $\mathcal{D}$ be the dual graph of $\mathcal{P}$. The edges of $\mathcal{D}$ are classified as literal, clause, and dummy edges according to the type of their dual edge in $\mathcal{P}$. Starting from $\mathcal{D}$, we construct a planar graph $\mathcal{F}$ with vertices of maximum degree 3 by first replacing every vertex of degree d with a binary tree with d leaves, and then replacing every edge of the resulting graph with a chain of five edges and edges.

We classify the edges of $\mathcal{F}$ as expansion, literal, clause, and dummy edges, where the edges forming the binary trees replacing the former vertices of $\mathcal{D}$ are *expansion edges*, and the remaining edges of $\mathcal{F}$ are classified according to the type of the edge of $\mathcal{D}$ that

originated them. Note that each edge e of $\mathcal{P}$ is thus associated with exactly five edges of $\mathcal{F}$ forming a path, and we call the middle edge the *representative* of e in $\mathcal{F}$.

Lemma 26 *Graph $\mathcal{F}$ has a unique planar embedding and admits a rectilinear embedding.*

Proof: Each embedding of $\mathcal{F}$ corresponds to exactly one embedding of $\mathcal{D}$, namely the embedding obtained by first replacing each chain of five edges by a single edge between the end points of the chain, and then replacing each binary tree induced by the expansion edges into a single vertex. Since $\mathcal{D}$ has a unique embedding, we have that $\mathcal{F}$ also has a unique embedding.

Every planar graph with vertices of degree at most 4 admits an orthogonal drawing with at most 4 bends per edge, which can be constructed in linear time (see, e.g., [102]). Hence, graph $\mathcal{F}$ admits a planar rectilinear drawing.. □

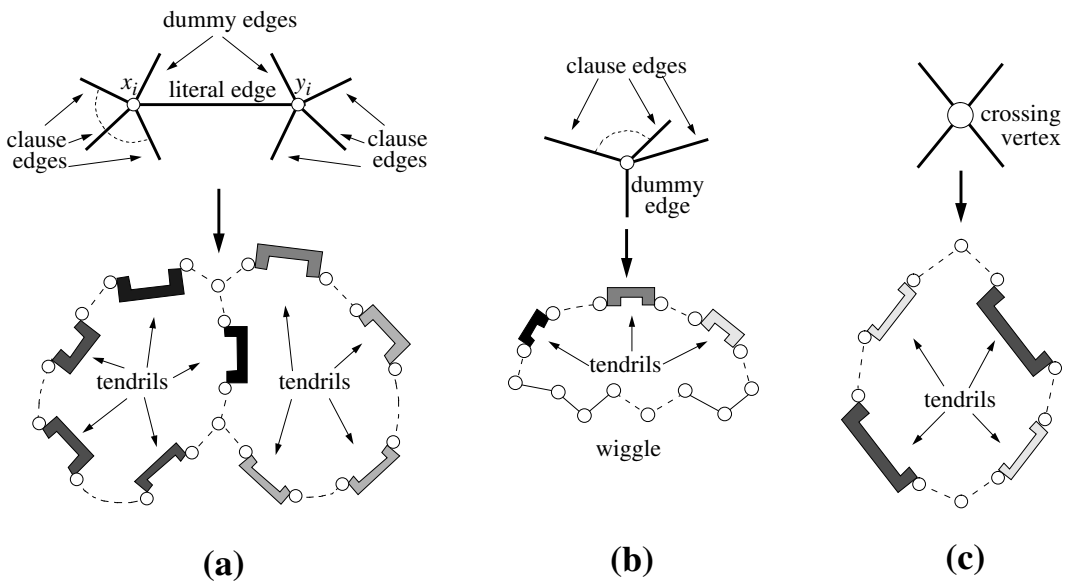

Figure 6.6: Schematic illustration of graph $\mathcal{G}$ obtained from $\mathcal{F}$ by replacing edges with rectilinear tendrils and wiggles: **(a)** the two faces of $\mathcal{G}$ associated with literal vertices x_i and y_i of $\mathcal{P}$. **(b)** the face of $\mathcal{G}$ associated with a clause vertex of $\mathcal{P}$. **(c)** the face of $\mathcal{G}$ associated with a crossing vertex of $\mathcal{P}$.

Finally, we construct graph $\mathcal{G}$ as follows (see Fig. 6.6). Let e be an edge of $\mathcal{P}$. If e is a dummy clause edge with capacity range $[0 \cdots c]$, we replace the representative of e in $\mathcal{F}$ with rectilinear wiggle W_c. Else, e is a literal, fragment, or dummy literal edge with capacity range $[c]$, and we replace the representative of e in $\mathcal{F}$ with rectilinear tendril T_c. The vertices of $\mathcal{G}$ that are also vertices of $\mathcal{F}$ are called its *primary* vertices.

By Lemma 26 and the construction of graph $\mathcal{G}$, all the embeddings of $\mathcal{G}$ are obtained by choosing one of the two possible flips for each rectilinear tendril. In an embedding of $\mathcal{G}$, the face that corresponds to the dummy vertex of $\mathcal{P}$ is called the *dummy face* of the embedding. The other faces are called the *nondummy* faces.

Recall that a rectilinear tendril T_k contributes one of $4k$, $4k+1$, $4k+2$, $-4k$, $-(4k+1)$, and $-(4k+2)$ to a face of $\mathcal{G}$. We say that the *significant* contribution of a rectilinear tendril T_k to a face is $4k$ if its contribution is positive and is $-4k$ if it is negative. The *significant* contribution of a rectilinear wiggle to a face is simply its contribution to the face. A face of an embedding of $\mathcal{G}$ is called *balanced* if the total significant contribution of its rectilinear tendrils and wiggles to it is zero. The *contribution* of primary vertices to a face f is equal to the number of angles at them in f labeled 3 minus the number of angles at them in f labeled 1.

Lemma 27 *If $n \geq 3$ and $m \geq 3$, then in any rectilinear embedding of $\vec{\mathcal{G}}$, the contribution of primary vertices to each face is at most $40nm$.*

Proof: Each face of $\mathcal{D}$ has $\max\{m+2, 2n+1, 4, m+2n\} \leq nm$ vertices for $n, m \geq 3$. The expansion of a vertex with degree d of $\mathcal{D}$ by a binary tree adds at most d vertices to an incident face. Hence, after we expand each vertex of $\mathcal{D}$ by a binary tree, we increase the number of vertices in a face by at most the facial degree of its dual vertex in $\mathcal{P}$, which is at most $7nm$ by Theorem 22. We further replace each edge by a chain of five edges resulting in a further increase of at most $4(7nm + nm) = 32nm$ vertices in each face. Hence each face of $\mathcal{F}$ has at most $32nm + 7nm + nm = 40nm$ vertices. Therefore, the contribution of the primary vertices to each face is at most $40nm$. □

Lemma 28 *Graph $\mathcal{G}$ admits a rectilinear planar drawing if and only if the rectilinear tendrils can be flipped and the rectilinear wiggles can be arranged such that every non-dummy face is balanced.*

Proof:

If: By Lemma 26, $\mathcal{F}$ has a rectilinear embedding. Let A be the assignment of labels to the angles in this embedding. A is a consistent assignment. Therefore if the tendrils can be flipped and the wiggles can be arranged to get an embedding ψ in which, for every face, the total significant contribution of the tendrils and wiggles is zero, we obtain from ψ a rectilinear embedding of $\mathcal{G}$ by assigning to the angles of the primary vertices of $\mathcal{G}$ the same label as in assignment A, and making the contribution of each tendril to a face equal to its significant contribution.

Only If: Suppose $\mathcal{G}$ has a rectilinear embedding ψ. Let B be the assignment of labels to the angles of ψ. By Lemma 24, B is a consistent assignment, so that, denoting with $N_3(f)$ and $N_1(f)$, the number of angles of face f with label 3 and 1, respectively, we have that $|N_3(f) - N_1(f)| = 4$ for every face f of ψ.

Let f be a nondummy face of ψ. By Lemma 27, f has at most $40nm$ primary vertices. Let $\tau(f)$ be the total contribution of the tendrils to f. Let $\sigma(f)$ be the total significant contributions of the tendrils to f. Recall from Section 6.2.2, that the contribution of a tendril T_k is one of $4k$, $4k+1$, $4k+2$, $-4k$, $-4k-1$, and $-4k-2$. Therefore, $\sigma(f)$ is a multiple of $4\theta = 128nm$. Also, because each face has at most nm tendrils, $|\sigma(f) - \tau(f)| \leq 2nm$. We have two cases:

Case 1: f corresponds to a literal or a crossing vertex of $\mathcal{P}$. Face f has no wiggles. $N_3(f) - N_1(f) = \tau(f) + t$, with $|t| \leq 40nm$. By Lemma 24, $|N_3(f) - N_1(f)| = 4$. Hence, $|\tau(f) + t| = 4$, with $|t| \leq 40nm$. Consequently, $|\sigma(f) + t| \leq 4 + 2nm$. Since $\sigma(f)$ is a multiple of $128nm$, this implies that $\sigma(f) = 0$.

Case 2: f corresponds to a clause vertex of $\mathcal{P}$. Face f has three tendrils and one wiggle. Let $\omega(f)$ be the contribution of the wiggle. We have $N_3(f) - N_1(f) = \tau(f) + \omega(f) + t$, with $|t| \leq 40nm$. By Lemma 24, $|N_3(f) - N_1(f)| = 4$. Hence, $|\tau(f) + \omega(f) + t| = 4$, with $|t| \leq 40nm$. Since $|\sigma(f) - \tau(f)| \leq 2nm$, we have $|\sigma(f) + \omega(f) + t| \leq 4 + 2nm$.

Let η_j be the sum of the absolute values of the significant contributions of the tendrils to f. By construction, we have $|\omega(f)| \leq \eta_j - 64nm$. We claim that $|\sigma(f)| \leq \eta_j - 128nm$. Indeed, since $\sigma(f)$ is a multiple of $128nm$, $|\sigma(f)| > \eta_j - 128nm$ implies $|\sigma(f)| = \eta_j$ and thus $|\sigma(f) + \omega(f)| \geq 64nm > 4 + 2nm + 40nm$, a contradiction.

Since the wiggle of f can be rearranged so that $\omega(f)$ takes any value between $-\eta_j + 64nm$ and $\eta_j - 64nm$, ψ can be modified so that $\sigma(f) + \omega(f) = 0$ without affecting the other faces of the embedding.

□

We establish the following correspondence between graph $\mathcal{G}$ and network $\mathcal{P}$ (see Fig. 6.6):

- The faces of $\mathcal{G}$ correspond to the vertices of $\mathcal{P}$.
- The rectilinear tendrils and wiggles of $\mathcal{G}$ correspond to the the edges of $\mathcal{P}$.
- Flipping a rectilinear tendril T_k of $\mathcal{G}$ corresponds to orienting an edge e of $\mathcal{P}$. e is the dual of the edge which is replaced by T_k in constructing $\mathcal{G}$ from $\mathcal{F}$. Edge e is oriented towards the dual vertex of a face f if and only if the significant contribution of T_k to f is $4k$.
- The significant contribution of a rectilinear tendril or wiggle U of $\mathcal{G}$ corresponds to the flow in an edge e of $\mathcal{P}$, where e is the dual of the edge replaced by U in constructing $\mathcal{G}$ from $\mathcal{F}$. The significant contribution of U to a face f is equal to the amount of the flow coming into the dual vertex of f through edge e.
- The sum of the significant contributions of the rectilinear tendrils and wiggles to the faces of $\mathcal{G}$ corresponds to the conservation of flow at the vertices of $\mathcal{P}$, i.e., the total significant contribution of the rectilinear tendrils and wiggles to a face f is zero if and only if there is a conservation of flow at the dual vertex of f.

Theorem 23 *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses, graph $\mathcal{G}$ associated with $\mathcal{S}$ has $O(n^4m^3)$ vertices and edges, and can be constructed in $O(n^4m^3)$ time. Instance $\mathcal{S}$ is satisfiable if and only if graph $\mathcal{G}$ is rectilinear planar. Also, given a rectilinear planar embedding for $\mathcal{G}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^4m^3)$.*

Proof: From Theorem 22 and the fact that each rectilinear tendril and wiggle has $O(n^2m)$ vertices and edges, it follows that $\mathcal{G}$ has $O(n^4m^3)$ vertices and edges, and can be constructed in $O(n^4m^3)$ time.

From Lemma 28 and the correspondence established above between graph G and network P it follows that instance $\mathcal{S}$ is satisfiable if and only if graph $\mathcal{G}$ is rectilinear planar. Also, given a rectilinear planar embedding for $\mathcal{G}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^4m^3)$. □

From Theorem 23 we conclude:

Corollary 9 *Rectilinear planarity testing is NP-complete.*

Corollary 10 *Computing a planar orthogonal drawing with the minimum number of bends is NP-hard.*

We can strengthen Corollary 10 as follows:

Corollary 11 *Let G be an n-vertex planar graph whose minimum number of bends in any planar orthogonal drawing is b^*. Computing a planar orthogonal drawing of G with $O(b^* + n^{1-\epsilon})$ bends is NP-hard for $\epsilon > 0$.*

Proof: Suppose there is a polynomial-time algorithm A that computes a planar orthogonal drawing of G with at most $c(b^* + n^{1-\epsilon})$ bends, where c is some constant. We can then use algorithm A to test in polynomial time whether graph G is rectilinear planar as follows: Construct a graph G' consisting of $K = \left\lceil (cn^{1-\epsilon})^{1/\epsilon} \right\rceil + 1$ copies of G and give G' as an input to algorithm A. Clearly, G is rectilinear planar if and only if G' has a planar orthogonal drawing with less than K bends. Since G' has Kn vertices and $K > c(Kn)^{1-\epsilon}$, algorithm A computes a drawing of G' with less than K bends if and only if G is rectilinear planar. □

6.4 Conclusions

Finding an efficient algorithm for rectilinear planarity testing had been an open problem for many years. In this chapter we have shown that a polynomial time algorithm for this problem is unlikely to exist by proving that the problem is NP-complete. NP-completeness of rectilinear planarity testing also implies that the bend-minimization problem for planar orthogonal drawings is NP-hard.

Chapter 7

Bend-Minimization

7.1 Introduction

This chapter addresses the problem of constructing orthogonal planar drawings of planar degree 4 graphs (each vertex has at most 4 edges incident on it) with minimum number of bends. An *orthogonal drawing* of a degree 4 graph is one in which each edge is drawn as a polygonal chain consisting of alternating horizontal and vertical line-segments (see Fig. 7.1(b)). A *planar* drawing is one with no edge-crossings. A *planar* graph is one which admits a planar drawing. The desire to construct orthogonal drawings with small number of bends is motivated by many reasons. Drawings with fewer bends tend to give more aesthetically pleasing drawings [31, 62]. Minimizing the number of bends seems to be one of the key factors in producing drawings that meet a number of other aesthetic criteria such as small total edge-length, good height-to-width ratio (also known as aspect ratio), small area etc., as demonstrated by an experimental study presented in the ACM symposium on computational geometry last year [31]. For improved readability of a drawing, it is important that the angles between edges be large in the drawing. This notion is formalized by defining *angular resolution* of a drawing, which is equal to the magnitude of the smallest angle in the drawing, between two edges incident on a vertex. Studies have shown that while some graphs do not admit any planar straight-line drawing (edges drawn as straight-lines) with optimal angular resolution, there are others who can be drawn with optimal angular resolution but only at the cost of having a large area [58, 83]. Orthogonal drawings with small number of bends seem to provide a good compromise in the sense that they have large angles (multiples of $90°$), small area, and few bends (every degree 4 graph admits am orthogonal drawing with $O(n^2)$ area and $O(n)$ bends [109, 75]). Minimization of bends is also important in VLSI layouts where bends act as "hot points" because of their higher current density. Finally we notice that even though orthogonal drawings are usually defined for degree 4 graphs, the results are useful for graphs with higher degree also. This is so because we can convert a graph with higher degree into a degree 4 graph by expanding each vertex with more than 4 edges incident on it, into a cycle. Many graph drawing systems are available that follow this approach, e.g., [32, 106].

Because of the importance of bend-minimum orthogonal drawings, researchers have

studied the problem extensively. We denote the number of vertices in a graph with n. Valiant [109] showed that every degree 4 graph admits a (nonplanar) drawing with at most $4n$ bends. As for the problem of constructing bend-minimum planar orthogonal drawings, Storer [97] gave a polynomial time algorithm for constructing a planar orthogonal drawing of a biconnected degree 4 planar graph with at most $2n+4$ bends. Tamassia and Tollis [102] gave a linear time algorithm for constructing a planar orthogonal drawing of a biconnected degree 4 planar graph with at most $2n + 4$ bends and area $O(n^2)$. Tamassia, Tollis and Vitter [103] gave a parallel algorithm and lower bounds for the problem. Kant [67, 69] gives linear-time algorithms for constructing planar orthogonal drawings of triconnected degree 4 planar graphs with at most $1.5n + 3$ bends and area at most n^2, and of triconnected degree 3 planar graphs with at most $0.5n + 1$ bends and area at most $n^2/4$. Biedl and Kant [13] give a linear time algorithm for constructing a planar orthogonal drawing of a planar degree 4 graph with at most $2n+2$ bends and area n^2. Papakostas and Tollis [87] show that each degree 4 planar graph admits a planar orthogonal drawing with at most $2n+4$ bends and area $0.76n^2$. They also show that each degree 3 planar graph admits a planar orthogonal drawing with at most $0.5n$ bends and area $0.25n^2$. Even and Granot [45] show that each degree 4 planar graph admits a planar orthogonal drawing with $O(n)$ bends such that each edge has at most 3 bends. Garg and Tamassia [59] showed that given a degree 4 planar graph G, it is NP-hard to construct a bend-minimum drawing of G. They also showed that even approximating the number of bends to within $O(n^{1-\epsilon})$ of the optimal, for any $\epsilon < 1$ is NP-hard. Di Battista, Liotta and Vargiu [33] have given polynomial time algorithms for minimizing bends in planar orthogonal drawings of series-parallel and degree 3 graphs.

An *embedded* planar graph is a planar graph equipped with an embedding. The *bend-minimization* problem for embedded planar degree 4 graphs is stated as follows:

> Given an embedded planar degree 4 graph G, construct a planar orthogonal drawing of G with minimum number of bends.

Tamassia [99] in 1987 gave an $O(n^2 \log n)$ algorithm that takes an embedded planar degree 4 graph G as input and constructs a planar orthogonal drawing of G with minimum number of bends, disproving the conjecture of Storer [97], who suspected that the problem was NP-hard. This algorithm not only made a fundamental theoretical contribution to the topic of bend-minimum planar orthogonal drawings, but also has been found to have good performance in practice. Using Dial's implementation for the priority queue used in the Dijkstra's shortest-path algorithm [1], Tamassia's algorithm can be implemented with $O(n^2)$ time-complexity. Experimental studies [31, 62] have shown that Tamassia's algorithm in general constructs better drawings in practice than some other well-known algorithms of [18, 19, 101, 13, 87]. The studies however indicate clearly that Tamassia's algorithm has a high running time in practice.

The problem of finding a faster algorithm for constructing an orthogonal planar drawing with minimum number of bends, of a planar degree 4 graph with fixed embedding, has been open for many years. Not only it is considered a fundamental theoretical problem, but as the discussion given above indicates, is of practical significance too.

In this chapter, we give the first subquadratic-time algorithm for constructing an

orthogonal planar drawing with minimum number of bends, of an embedded planar degree 4 graph. Namely, we give an algorithm with time-complexity $O(n^{1.75}\log n)$, improving the best known time-complexity by a factor of $n^{0.25}/\log n$. Given an embedded planar degree 4 graph G, Tamassia's algorithm reduces the bend-minimization problem for G to a min-cost max-flow problem on a nonplanar flow network $\mathrm{N}(G)$ such that a min-cost max-flow f of $\mathrm{N}(G)$ corresponds to a bend-minimum orthogonal planar drawing of G. The time-complexity of Tamassia's algorithm is dominated by the time-complexity of computing f. We give in this chapter, a new algorithm, called *compute-flow*, that computes a min-cost max-flow g in an integer flow network P with no cost free arcs, in time $O(\chi^{3/4}m\log n + m\log n)$, where n and m are the number of nodes and arcs of P, and χ is the cost of g. An *integer flow network with no cost free arcs* is one in which each arc has integer capacity, and an integer cost which is at least 1. Because it can be shown that f has cost $O(n)$, it follows that Algorithm *compute-flow* computes f in time $O(n^{1.75}\log n)$, giving us a bend-minimization algorithm with time-complexity of $O(n^{1.75}\log n)$. Since Algorithm *compute-flow* computes min-cost max-flow in a more general flow network than the ones used for bend-minimization, we believe that it may have applications in other flow-based problems also.

This chapter is organized as follows. In Section 7.2, we give some definitions. In Section 7.3, we review some basic concepts of network flows. In Section 7.4, we give a brief description of Tamassia bend-minimization algorithm, which we call *the min-bend* algorithm. In Section 7.5, we present Algorithm *compute-flow*. We conclude with Section 7.6.

7.2 Preliminaries

Let H be a graph. *Degree* of a vertex v of H is equal to the number of edges incident on v. *Degree* of H is the maximum degree of any vertex in H. A *drawing* of H is a mapping of its vertices to points and its edges to continuous curves joining their end-points. H is *planar* if it admits a planar drawing. In each planar drawing of H, the edges of a vertex are incident on it in a cyclic order. An *embedding* of H is the collection of circular permutations of the edges incident on each vertex in some planar drawing of H. H is an *embedded* planar graph if it is equipped with an embedding. All planar drawings of an embedded planar graph are topologically equivalent. Let K be an embedded planar graph. Each planar drawing of K divides the plane into several regions, which are separated from each other by cycles consisting of the edges and vertices of K; These cycles are called the *faces* of K. Notice that exactly one region is unbounded; The cycle separating it from the other regions is called the *external face* of K. The other faces of K are called its *internal faces*. v is an *external* vertex (*internal* vertex, otherwise) of K if it is on the external face of K.

Let G be an embedded degree 4 planar graph. An *orthogonal* representation O of G is one in which each vertex is represented as points, and each edge e is represented as a polygonal chain $c(e)$ consisting of alternating horizontal and vertical line-segments (see Fig 7.1(b)). A vertex of $c(e)$, that is not a vertex of G, is called a *bend* of e in O. Notice that each face f of G is mapped to a polygon $P(f)$ in O. The *angles* of O in a face f of G

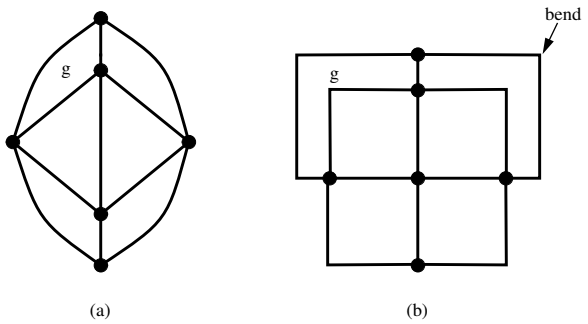

Figure 7.1: (a) An embedded degree 4 planar graph G. (b) An orthogonal representation of G.

are the interior angles of $P(f)$ if f is an internal face, and are the exterior angles of $P(f)$ if f is the external face of G. For example, in Fig 7.1(b), the angles of $P(g)$ are 90°, 90°, 270°, 90°, 90°, and 90°. An *angle* of O is an angle in O of a face of G. An *orthogonal planar* drawing $D(O)$, *associated* with O, is a planar drawing of G that preserves the angles of O (see Fig 7.1(b)). A *consistent* orthogonal representation is one that admits an orthogonal planar drawing. The following important theorem which follow immediately from the results of [99] characterizes consistent orthogonal representations:

Theorem 24 *Let G be an embedded planar degree 4 graph. An orthogonal representation O of G is consistent if and only if, in O:*

- *for every vertex v of G, The sum of the angles between consecutive line-segments incident on v is equal to* 360°.
- *for every face f of G, the sum of its angles is equal to $(m-2)180°$ if it is an internal face, and is equal to $(m+2)180°$ if it is the external face of G, where m is the total number of line-segments in the polygon to which f is mapped in O.*

7.3 Network Flows

In this section, we review some basic concepts and definitions related to network flows. We also give a brief description of Algorithms *SAP*, *BlockFlow* and *PD*, which embody three well-known techniques for computing flows in flow networks. The notation followed is generally the one of [85].

A *flow network* $N = (V, E, c, u)$, consists of a directed graph $G = (V, E)$ with a node set V and an arc set E, a cost function $c : E \rightarrow R_+$, and a capacity function $u : E \rightarrow R_+$. See Fig 7.2 for an example of a flow network. The *cost* of an arc e is equal to $c(e)$. The *capacity* of an arc e is equal to $u(e)$. N is an *integer* flow network if both the costs and the capacities of all its arcs are integers. N has no *cost-free* arc if the cost of each arc is greater than 0. In particular, if N is an integer flow network with no cost-free arc, then the cost of each arc is at least one. Let s and t be two distinct vertices that are called the *source* and *sink* respectively of N. Let $arc(v)$ be the set of

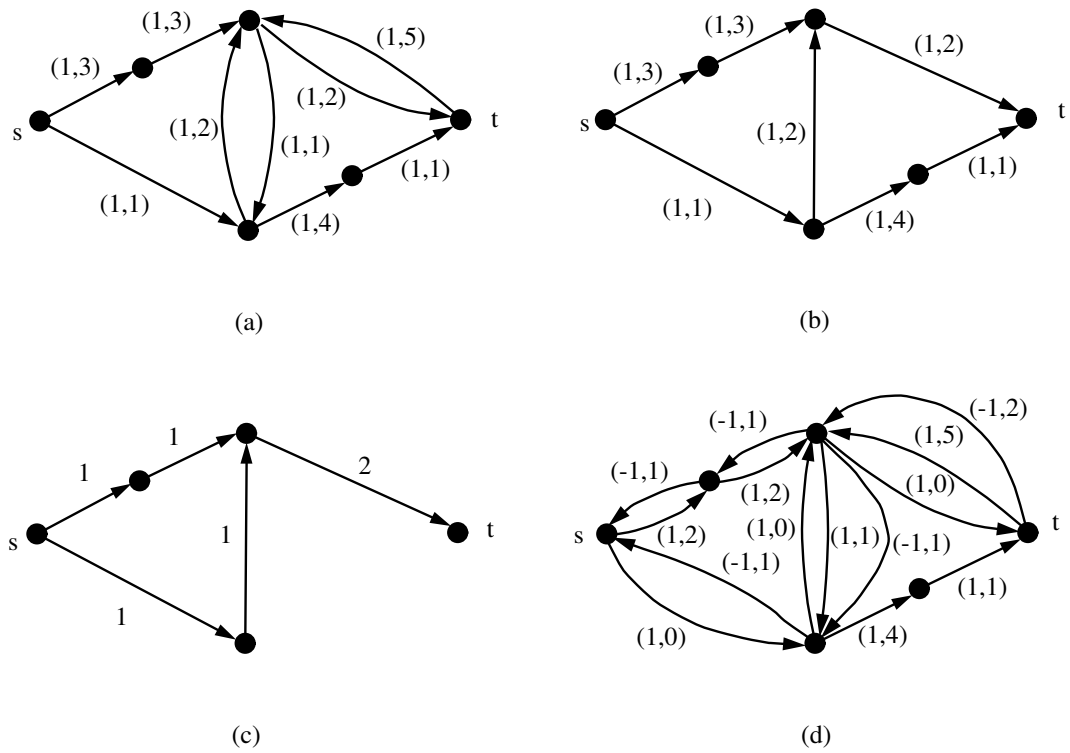

Figure 7.2: (a) A flow network $N(V, E, c, u)$ with source s and sink t. (b) The shallowest layered network $L(N, \mathbf{0})$ of N with respect to zero flow; L is also the admissible flow network of N with respect to zero flow because each arc of N has cost 1. (c) A Blocking flow f of L. (d) Residual network $R(N, f)$ of N with respect to f. In parts (a), (b), and (d) we have labeled each arc e by the pair $(c(e), u(e))$. In part (d) we have labeled each arc by the amount of flow in it, and have shown only the arcs with non-zero flow.

arcs incident on a node v. A *feasible* flow f is a function $f : E \to R_+$ that satisfies, (a) the capacity constraints, i.e., for every arc e, $f(e) \leq u(e)$, and (b) the conservation law, i.e., for every vertex $v \in V - \{s, t\}$, $\sum_{e \in arc(v)} f(e) = 0$. *Magnitude* of a flow f in N is equal to the sum of magnitudes of flows in individual arcs, i.e., $|f| = \sum_{e \in E} |f(e)|$. Let e be an arc. *Cost* of a flow $f(e)$ in arc e, denoted by $c(f(e))$, is equal to $c(e)$ times $|f(e)|$, i.e., $c(f(e)) = c(e) \cdot |f(e)|$. *Cost* of a a flow f, denoted by *c(f)*, is equal to the sum of the costs of flow f over all the arcs of N, i.e, $c(f) = \sum_{e \in E} c(f(e))$. An arc e is *saturated* in a flow f if $f(e) = u(e)$. A *maximum* flow (*max-flow* in short) of N is a feasible flow with maximum magnitude among all the feasible flows of N. A *minimum cost maximum* flow (*min-cost max-flow* in short) of N is a maximum flow with minimum cost among all the maximum flows of N. A *zero* flow, denoted by $\mathbf{0}$, is a flow with zero magnitude. A *non-zero* flow is a flow with non-zero magnitude.

A *directed path* p of *length* l from u to v is a sequence of (directed) arcs $(u = v_1, v_2)$, $(v_2, v_3), \ldots, (v_{l-1}, v_l = v)$. p is also denoted as $p = uv_2v_3 \ldots v$, and also sometimes simply as $p = u \rightsquigarrow v$. A *shortest* path from u and v is a directed path from u to v with smallest length. A path p' is a *subpath* of p if each and every arc of p' is also in p. *cost* of p, denoted by $c(p)$, is equal to the sum of the costs of the arcs of p, i.e.,

$c(p) = \sum_{1 \leq i \leq l-1} c((v_i, v_{i+1}))$. A path p from u to v is a *capacitated* path if each and every arc of p has a non-zero capacity. An *augmenting* path p of N is a capacitated path from s to t. The concept of augmenting path is very important in network flows, and is used in a number of algorithms for computing flows. The intuition is that if N has an augmenting path p, then the amount of (feasible) flow in N can be increased by "pushing" a flow of non-zero magnitude from s to t along p.

A *residual* flow network $R(N, f)$ of N with respect to a feasible flow f (see Fig. 7.2(d)) is a flow network such that for every arc $e(u, v)$ of N, it consists of arcs $e' = (u, v)$ and $e'' = (v, u)$ such that $c(e') = -c(e'') = c(e)$, $u(e') = u(e) - f(e)$, and $u(e'') = f(e)$. Notice that from this definition $R(N, f)$ may have two arcs from one vertex to another. Usually, if $R(N, f)$ has two arcs from a vertex u to a vertex v, both with same cost c, and capacities u_1 and u_2 respectively, then we replace the two arcs by a single arc from u to v with cost c and capacity $u_1 + u_2$. We therefore assume in this chapter that if $R(N, f)$ has two arcs from one vertex to another vertex, then the two arcs have different costs. Intuitively, $R(N, f)$ consists of arcs which admit more flow, and hence can be used to "push" more (feasible) flow from s to t. A *forward* arc of $R(N, f)$ is an arc with non-negative cost. A *backward* arc of $R(N, f)$ is an arc with negative cost. Notice that from the definition of a backward arc, for every backward arc $e = (u, v)$ in $R(N, f)$, there is a forward arc $e' = (v, u)$ in $R(N, f)$ such that $f(e') = u(e)$, and $c(e') = -c(e)$. A *layered* flow network $L(N, f)$ of N with respect to a feasible flow f is a maximal subgraph of the residual network $R(N, f)$ such that, $L(N, f)$ contains both s and t, all the augmenting paths of $L(N, f)$ have the same length d, and each and every arc of $L(N, f)$ is in some augmenting path of $L(N, f)$; *Depth* of $L(N, f)$ is equal to d. Fig. 7.2(b), for example, shows a layered network with depth 3. $L(N, f)$ is the *shallowest* layered network of N with respect to f if all the layered networks of N with respect to f have depth at least d (see Fig. 7.2(b)). A *blocking flow* f in a layered flow network L is one in which for every directed path $p = s(= v_0)v_1v_2 \ldots (t = v_l)$ from s to t in L, at least one arc of p is saturated, i.e., for at least one arc $e_k = (v_k, v_{k+1})$, where $0 \leq k \leq l-1$, $f(e_k) = u(e_k)$ (see Fig. 7.2(c)). Notice that a blocking flow may not be a maximum flow. For example, the layered network of Fig. 7.2(b) admits a maximum flow of magnitude 3, and also admits a blocking flow of magnitude 2 (which is shown in Fig. 7.2(c)). Let p be an augmenting path of $R(N, f)$ with least cost. Let c_{min} be the cost of p. The *admissible* flow network $A(N, f)$ of N with respect to a feasible flow f is the maximal subgraph of the residual network $R(N, f)$ such that, $A(N, f)$ contains both s and t, and all the augmenting paths of $A(N, f)$ have the same cost, equal to c_{min}, and each and every arc of $A(N, f)$ is on some augmenting path of $A(N, f)$ (see Fig. 7.2(b)).

An *st-cut* A is a set of arcs that partition the nodes of N into two sets X and Y such that, $s \in X$, $t \in Y$, and each capacitated directed path from s to t contains an arc e belonging to set A. Following Lemma is immediate from the definition of an augmenting path:

Lemma 29 *Each augmenting path of N contains at least one arc belonging to A.*

Research in the area of network flows has a rich tradition (see [1] for an extensive survey). A number of algorithms have been proposed for finding min-cost max-flows and

max-flows in flow networks. Fig 7.3 shows a well-known algorithm, which we call the *SAP* algorithm (short for shortest augmenting path algorithm), which given N as input, finds a max-flow in N by successively pushing flow along shortest augmentation paths in the residual networks of N (with respect to the flow already pushed in the network). Many different implementations, with different time-complexities, of this algorithm are available in literature.

```
Algorithm SAP(N): /*N is a flow network*/
begin
    R_0 ← N; i ← 0;
    while R_i has at least one augmenting path
    begin
        Find a shortest augmenting path p_i of R_i;
        Push a non-zero flow f from s to t through p_i;
        Let F_i be the total flow in N after establishing flow f in R_i;
        Let R_{i+1} be the residual network of N with respect to flow F_i;
        i ← i + 1;
    end
end
```

Figure 7.3: Algorithm *SAP*.

Lemma 30 follows immediately from Theorem 8.13 of [105].

Lemma 30 ([105](Chapter 8, pp. 110)) *Let N be a flow network with integer capacities, and consisting of n nodes and m arcs. Suppose N admits a max-flow with magnitude ϕ.* Algorithm SAP *can be implemented such that it finds a max-flow (of magnitude ϕ) of N in time $O((\phi+1)\cdot m\log n)$.*

The implementation of *Algorithm SAP* that achieves the $O((\phi+1)\cdot m\log n)$ upper bound of Lemma 30 uses Dijkstra's shortest path algorithm for non-negative edge weights (which has time-complexity $O(m\log n)$ when implemented using priority queues [20]), for computing p_i. See [105] or [1] for details.

Fig 7.4 shows another well-known algorithm, which we call the *Blockflow* algorithm, that computes a max-flow f in a flow network N. This algorithm computes f in phases, where in each phase it computes a blocking flow in the shallowest layered network of N (with respect to the flow already computed). See [85] for details about this algorithm.

Lemma 31 summarizes some important results proved in [85].

Lemma 31 *[85](chapter IV, Section 9) The following hold for each phase i:*

1. *Phase i, except when it is the last phase, increases the amount of flow in N by at least one.*
2. *The depth of layered network L_i is strictly greater than the depth of the layered network L_{i-1}.*

Algorithm *Blockflow(N)*: /*N is a flow network*/
begin
    $L_0 \leftarrow N$, $i \leftarrow 0$;
    while L_i admits a non-zero blocking flow (phase i)
    begin
        Find a blocking flow f_i of A_i;
        Let F_i be the total flow in N after establishing flow f_i in L_i;
        Let L_{i+1} be the shallowest layered network of N with respect to flow F_i;
        $i \leftarrow i + 1$;
    end
end

Figure 7.4: Algorithm *Blockflow*.

3. *Phase i can be executed in time $O(m_i \log n_i)$, where n_i and m_i denote the number of nodes and arcs respectively in A_i.*

The implementation of *Blockflow* in which each stage i has time-complexity $O(m_i \log n_i)$ uses dynamic weighted trees for computing blocking flows. See [85, 96] for details.

Although Algorithm *Blockflow* computes a maximum flow, it can be used for computing a min-cost max-flow in a flow network. For example, Algorithm *PD* described later in this section reduces the problem of computing a min-cost max-flow to the problem of computing max-flows in some intermediate flow-networks. Each of these max-flows can be computed using Algorithm *Blockflow*. We present Lemma 32 which is useful in such a scenario. Suppose each arc e of N has a cost $c(e)$ associated with it. Let s and t be the source and sink respectively of N. Let ϕ_i be the magnitude of the blocking flow f_i computed in phase i of Algorithm *Blockflow*. Therefore, f_i corresponds to ϕ_i flows, $g_1, g_2, \ldots, g_{\phi'}$, in A_i, from s to t, where each g_j has magnitude exactly one. Let $R(N, F_i)$ be the residual network of N with respect to flow F_i. The proof of Lemma 32 is on the lines of the proof of a similar lemma, namely Lemma 8.4, of [105](chapter 8, pg. 110).

Lemma 32 *If each and every augmenting path of N has the same cost, and each and every arc of N is in some augmenting path of N, then the cost of an augmenting path in $R(N, F_i)$ is same as the cost of an augmenting path in N.*

Proof: Let $d(N', v)$ denote the minimum cost of a capacitated path from the source s' to node v in a flow network N'. In a network N', a node v is *reachable* from its source s' if there is a capacitated path from s' to v. We now show by induction over the magnitude of a flow f established in N, that the residual flow-network $R(N, f)$ has (a) *distance-retention property*, i.e., for every node v reachable from s in $R(N, f)$, $d(R(N, f), v) = d(N, v)$, and (b) *critical arc property*, i.e., for every arc $e = (u, v)$ in $R(N, f)$, $c(e) = d(N, v) - d(N, u)$. The residual flow-network $R(N, \mathbf{0})$ with respect to a zero flow trivially has the distance-retention property. $R(N, \mathbf{0})$ also has the critical arc property because each augmenting path of N has the same cost, and each and every arc of

N is in some augmenting path of N. Now suppose that the residual flow-network $R(N,g)$ with respect to a flow g has the distance-retention and critical arc properties. Suppose we establish in $R(N,g)$, a non-zero flow f' along an augmenting path p, giving us a total flow f^*. Let $R(N,f^*)$ be the residual flow-netork of N with respect to flow f^*. The only arcs in $R(N,f^*)$ that are not in $R(N,g)$, are of the form (w,u), where (u,w) is on path p, and $c((w,u)) = -c((u,w)) = d(N,u) - d(N,w)$. Hence, $R(N,f^*)$ also has the critical-arc property. We now show by induction over the length $l(v)$ of a minimum cost capacitated path p_v from s to v in $R(N,f^*)$ that, $d(R(N,f^*),v) = d(N,v)$. This is trivially true when $l(v) = 0$, in which case $v = s$. Now suppose $l(v) = k$, where $k \geq 1$. Let there be an arc $e = (u,v)$ in p_v. Path $p' = s \rightsquigarrow u$, where p' is a subpath of p_v, has length $k-1$. Since p_v is a minimum cost capacitated path from s to v, p' is a minimum cost capacitated path from s to u. Hence, by the inductive hypothesis, $d(R(N,f^*),u) = d(N,u)$. Because $R(N,f^*)$ has the critical-arc property, $c(e) = d(N,v) - d(N,u)$. Therefore, $d(R(N,f^*),v) = d(R(N,f^*),u) + c(e) = d(N,u) + d(N,v) - d(N,u) = d(N,v)$. Therefore, $R(N,f^*)$ has distance-retention property also.

Therefore, $R(N,F_i)$ also has the distance-retention property. Hence, the cost of an augmenting path in $R(N,F_i)$ is equal to $d(N,t)$, which is also the cost of an augmenting path in N. □

Fig. 7.5 shows another well-known algorithm, which we call *PD* algorithm (short for primal-dual algorithm), that finds a min-cost max-flow f in a network N. Algorithm *PD* was developed first by Ford and Fulkerson [46, 47]. This algorithm computes f in stages, where in each stage it computes a maximum flow in the admissible network of N (with respect to the flow already computed). See [1, 46, 47] for details.

```
Algorithm PD(N): /*N is a flow network*/
begin
    Let A_0 be the admissible network of N with respect to a zero-flow;
    i ← 0;
    while A_i admits a non-zero maximum flow (stage i)
    begin
        Find a maximum flow f_i of A_i;
        Let F_i be the total flow in N after establishing flow f_i in A_i;
        Let A_{i+1} be the admissible network of N with respect to flow F_i;
        i ← i + 1;
    end
end
```

Figure 7.5: Algorithm *PD*.

Let c_i be the cost of an augmentation path in A_i (if A_i has no augmenting path, then c_i is ∞). Lemma 33, which follows directly from the discussion on the primal-dual algorithm in [1] states that $c_{i+1} > c_i$. Intuitively, this is so because after establishing the maximum flow f_i in A_i, there are no augmenting paths left in N with cost c_i.

Lemma 33 $c_{i+1} > c_i \geq 0$.

The following corollary is immediate.

Corollary 12 *If each arc of N has non-zero integer cost, then $c_i \geq 1$, and $c_{i+1} \geq c_i + 1$.*

Let r be the total number of stages used by *Algorithm PD*. If we denote by T_i, the time-complexity of stage i, then we have:

Lemma 34 *[1] Total time taken by* Algorithm PD *to compute a min-cost max flow in N is equal to $\sum_{1 \leq i \leq r-1} T_i + O(m \log n)$, where n and m denote the number of nodes and arcs respectively in N.*

7.4 Bend-Minimization and Network Flow

It is shown by Tamassia in [99], that the problem of constructing an orthogonal planar drawing with minimum bends of an embedded degree 4 planar graph, can be solved by solving a related min-cost max-flow problem. In this section, we give a brief description of the algorithm of [99], which we call *the min-bend* algorithm. Details can be found in [99].

Let G be an embedded degree 4 planar graph. The *min-bend* algorithm is shown in Fig 7.7. As shown in the figure, it consists of two phases, namely, the *orthogonalization phase*, and the *find-length phase*. In the orthogonalization phase, the algorithm constructs a consistent orthogonal representation $\mathrm{O}(G)$ of G. In the find-length phase, it constructs an orthogonal planar drawing associated with $\mathrm{O}(G)$.

Algorithm *Orthogonalize* is shown in Fig. 7.8. As is shown in the figure, Algorithm *Orthogonalize* first constructs the *associated* network $\mathrm{N}(G)$ of G, then computes a min-cost max-flow f in $N(G)$, which is then used to construct a consistent orthogonal representation $\mathrm{O}(G)$ of G. The *associated* network $\mathrm{N}(G)$ of G is described as follows (see Fig 7.6): Except for two distinguished nodes s and t , the nodes of $\mathrm{N}(G)$ correspond to the vertices and faces of G. s is the source of $\mathrm{N}(G)$, and t is the sink of $\mathrm{N}(G)$. $\mathrm{N}(G)$ has four types of arcs: *vertex-face* arcs, *face-face* arcs, *source-node* arcs, and *face-sink* arcs. Let us denote by $degree(v)$, the degree of vertex v in G. Let us denote by $length(h)$ the number of edges in face h of G. Let S be a set of nodes of $\mathrm{N}(G)$, such that for every node $v \in S$, the following holds: (a) either v corresponds to a vertex of G, or (b) v corresponds to a face h of G such that $length(h)$ is at most 3. Let T be a set of nodes of $\mathrm{N}(G)$, such that every node in T corresponds to a face h of G such that $length(h)$ is at least 4. The arcs of $\mathrm{N}(G)$ are as follows:

- $\mathrm{N}(G)$ has a *vertex-face* arc $e = (v, f)$ if and only if vertex v is on face f (see Fig 7.6(b)). e has capacity $4 - degree(v)$ and cost 1.
- For every triplet (f, g, e), such that e is an edge that is common to faces f and g, $\mathrm{N}(G)$ has two *face-face* arcs $e' = (f, g)$ and $e'' = (g, f)$(see Fig 7.6(c)). Arcs e' and e'' have infinite capacity and cost 1.

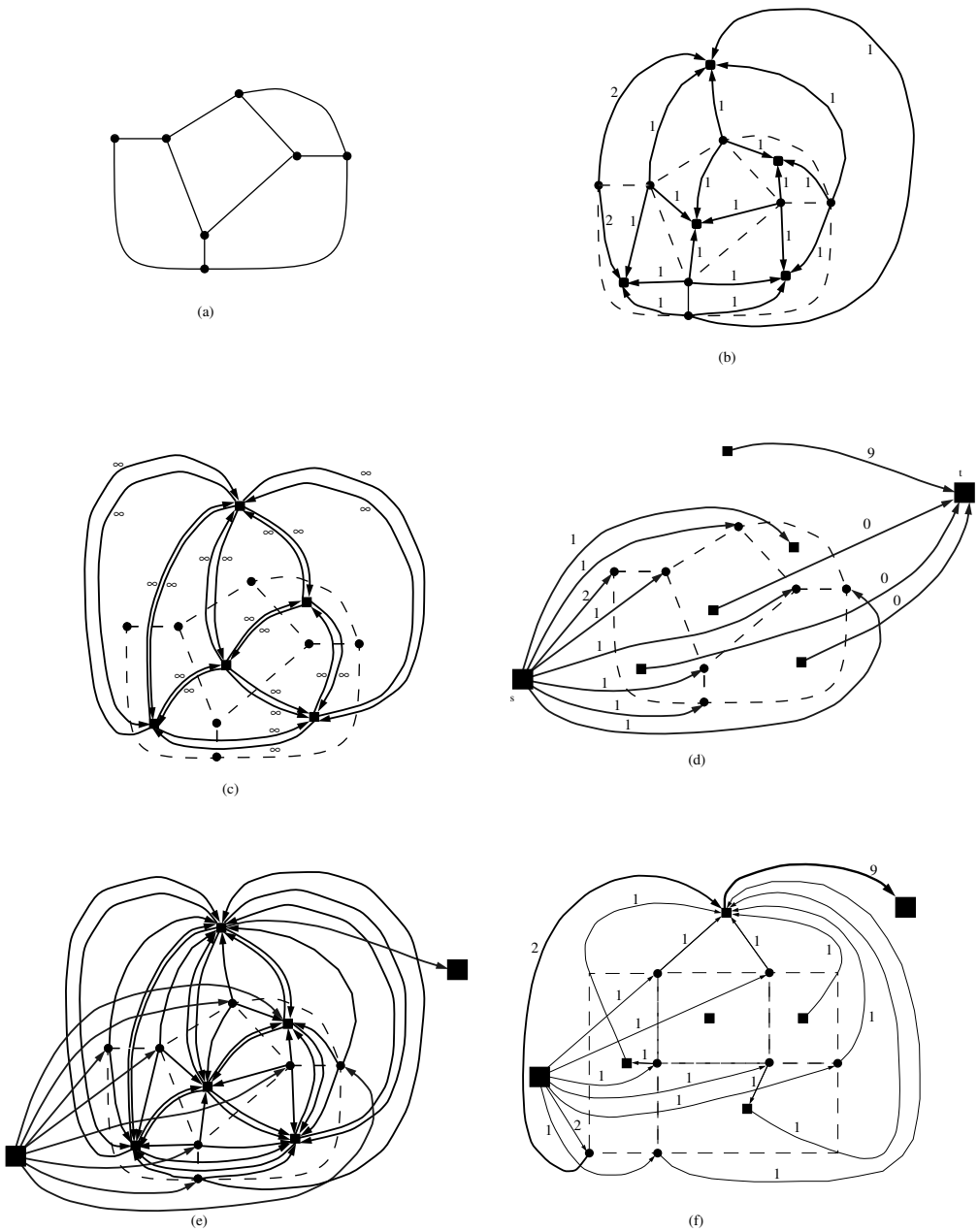

Figure 7.6: Bend-minimization and network flow: (a) An embedded degree 4 planar graph G; (b) Node-face arcs of the associated flow-network N(G) of G; (c) Face-face arcs of N(G); (d) Source-node and face-sink arcs of N(G); (e) Flow-network N(G); (f) A min-cost max-flow in N(G) and the corresponding bend-minimum planar orthogonal representation of G(shown as broken lines). Notice that each arc of N(G) has cost 1. In parts (b)-(f), we have shown the arcs of N(G) as solid lines, and the edges of G as broken lines. In parts (b)-(d), we have have labeled each arc by its capacity. In part (f), we have labeled each arc by the amount of flow in it, and have shown only the arcs with non-zero flow.

Algorithm *min-bend(G)*: /*G is an embedded degree 4 planar graph*/
begin
Orthogonalization Phase. Invoke Algorithm *Orthogonalize(G)* to construct a consistent orthogonal representation O(G) of G, with minimum number of bends;
Find-lengths Phase. Construct an orthogonal planar drawing associated with O(G).
end

Figure 7.7: Algorithm *min-bend*

Algorithm *Orthogonalize(G)*: /*G is an embedded degree 4 planar graph*/
begin
Construct the associated flow network N(G) of G;
Compute a min-cost max-flow f in N(G);
Compute a consistent orthogonal representation O(G) of G from f.
end

Figure 7.8: Algorithm *Orthogonalize*

- For every node $v \in S$, there is a *source-node* arc $e = (s, v)$ in N(G) (see Fig. 7.6(d)). e has cost 1. If v is a vertex of G, then e has capacity $4 - degree(v)$. If v is a face of G, then e has capacity $4 - length(v)$.
- For every node $h \in T$, there is a *face-sink* arc $e = (h, t)$ in N(G) (see Fig. 7.6(d)). e has cost 1. Recall from the definition of set T that, h is a face of G. If h is the external face of G then e has capacity $length(h) + 4$, otherwise e has capacity $length(h) - 4$.

Fig. 7.6(e) shows the complete flow-network N(G) (shown as solid lines) for the embedded degree 4 planar graph of Fig. 7.6(a).

The following correspondence holds between a maximum flow f in the flow-network N(G) and a consistent orthogonal representation O(G) of G:

- r units of flow on a vertex-face arc (v, f) corresponds to an angle equal to $(r+1)\pi/2$ at vertex v in face f.
- k units of flow on a face-face arc from f to g corresponds to k bends on the corresponding edge of G. Each bend has an angle equal to $3\pi/2$ in face g.

Fig 7.6(f) shows a maximum flow in the associated network of Fig. 7.6(e), and a corresponding consistent orthogonal representation (shown as broken lines).

Let $z(s)$ be the sum of the capacities of the arcs of N(G) incident on s. Clearly the maximum flow in N(G) is at most $z(s)$. Lemma 35 follows from Lemma 2 of [99].

Lemma 35 ([99], Lemma 2) *N(G) admits a maximum flow of magnitude $z(s)$.*

Notice that because each source-node, node-face, and face-sink arc has cost 1, each maximum flow of N(G) has cost at least $3z(s)$. Lemmas 36, 37, 38, 39, and 40 either paraphrase, or follow immediately from results proved in [99].

Lemma 36 ([99], Lemma 6) *N(G) has $O(n)$ nodes and arcs.*

Lemma 37 ([99], Theorem 2) *A min-cost max-flow f of N(G) corresponds to a consistent orthogonal representation O(G) of G with minimum number of bends. Moreover if f has cost ϕ, then O(G) has exactly $\phi - 3z(s)$ bends.*

Lemma 38 ([99], Lemma 7) *G admits a consistent orthogonal representation with at most 3n+2 bends.*

Lemma 39 ([99], Theorem 1) *If O(G) has b bends then Phase* find-length *constructs a planar drawing associated with O(G) in $O(n + b)$ time.*

Lemma 40 ([99], Theorem 4) *If $M(\mathcal{N}(G))$ is the time of computing a min-cost max-flow f in N(G), then the* orthogonalization *phase can be implemented such that it computes a consistent orthogonal representation O(G) of G in $O(M(\mathcal{N}(G)) + n)$ time.*

Theorem 25 *Let G be an embedded degree 4 planar graph. Then the associated flow network N(G) of G is an integer flow network with no cost free arc, has $O(n)$ nodes and arcs, and admits a min-cost max-flow with cost $O(n)$. Moreover, if $M(\mathcal{N}(G))$ is the time of computing a min-cost max-flow f in the associated flow network N(G) of G, then we can construct an orthogonal planar drawing of G with minimum bends in $O(M(\mathcal{N}(G)) + n)$ time.*

Proof: From the construction of N(G) it is clear that N(G) is an integer flow network with no cost free arc. From Lemma 36, N(G) has $O(n)$ nodes and arcs. Let b be the minimum number of bends in an orthogonal planar drawing of G. From Lemma 38, $b = O(n)$. Because each arc incident on s has capacity at most 4, $z(s) = O(n)$. Therefore, from Lemma 37, the cost of a min-cost max-flow in N(G) is equal to $b + 2z(s) = O(n)$.

Again, because $b = O(n)$, from Lemma 39 we have that the time-complexity of *find-length* phase is $O(n)$. Therefore, from Lemma 40 it follows that the total time-complexity of Algorithm *min-bend* is $O(M(\mathcal{N}(\mathrm{G})+n)$. Hence, we can construct an orthogonal planar drawing of G with minimum bends in $O(M(\mathcal{N}(\mathrm{G})) + n)$ time. □

The algorithm described in [99] uses an $O(n^2 \log n)$ algorithm for computing a min-cost max-flow in a network, resulting in a total time-complexity of $O(n^2 \log n)$ for *min-bend*. A faster algorithm for computing a min-cost max-flow in an integer network will give us a faster bend-minimization algorithm. In section 7.5, we give a new algorithm, namely Algorithm *compute-flow* which computes a min-cost max-flow f in an integer flow network with no cost free arc, in time $O(\chi^{3/4} m \log n)$, where χ is the cost of f. Because from Theorem 25 N(G) is an integer flow network with no cost free arc, has $O(n)$ nodes and arcs, and admits a min-cost max-flow with cost $O(n)$, a min-cost max-flow in N(G) can be computed in $O(n^{3/4} n \log n) = O(n^{1.75} \log n)$ time. This gives us an algorithm for bend-minimization with time-complexity $O(n^{1.75} \log n)$.

7.5 Algorithm compute-flow

Let $N = (V, E, c, u)$ be an integer flow network with n nodes and m arcs, such that N admits a max-flow with cost at most χ. Let s and t be the source and sink respectively of N. Lemma 44, which is the main result of this section, shows that a max-flow in N with cost at most χ can be computed in $O(\chi^{3/4} m \log n)$ time. This computation is done using Algorithm *compute-flow* shown in Fig 7.9. Algorithm *compute-flow* is a variation of the Algorithm *PD* described in Section 7.3. The main feature of Algorithm *compute-flow* is that, in each stage, it computes f_i by running two algorithms in parallel- Algorithm *SAP*(N_i) and Algorithm *BlockFlow*(N_i). f_i is therefore the flow computed by the algorithm terminating first.

Algorithm *compute-flow(N)*: /*N is an integer flow network with no cost free arcs*/
begin
    Let A_0 be the admissible network of N with respect to a zero-flow;
    $i \leftarrow 0$;
    while A_i admits a non-zero maximum flow (stage i)
    begin
        To find a maximum flow of A_i, run *SAP*(N_i) and *BlockFlow*(N_i) in parallel. Go to the next step when any one of them terminates after computing a maximum flow f_i;
        Let F_i be the total flow in N after establishing flow f_i in A_i;
        Let A_{i+1} be the admissible network of N with respect to flow F_i;
        $i \leftarrow i + 1$;
    end
end

Figure 7.9: Algorithm *compute-flow*.

Before we prove Lemma 44, we need to do some ground work. We use the notation of Fig 7.9. Let χ be the cost of the min-cost max-flow f computed by Algorithm *Compute-flow*, given N as input. Let ϕ_i and χ_i be the magnitude and cost respectively, of flow f_i. Let $S = \{f_i | 1 \leq \phi_i \leq \sqrt{\chi}\}$. Let $B = \{f_i | \phi_i > \sqrt{\chi}\}$. Let $\phi_S = \sum_{f_i \in S} \phi_i$. Let $\phi_B = \sum_{f_i \in B} \phi_i$. Let $\chi_S = \sum_{f_i \in S} \chi_i$. Let $\chi_B = \sum_{f_i \in B} \chi_i$.

Lemma 41 $\phi_S = O(\chi^{3/4})$.

Proof: Suppose set S consists of flows $f_{i_1}, f_{i_2}, \ldots, f_{i_l}$, where $i_1 < i_2 \ldots < i_l$. From the definition of ϕ_S, $\phi_S = \sum_{1 \leq j \leq l} \phi_{i_j}$. From the definition of flow network A_{i_j}, all the augmenting paths of A_{i_j} have the same cost c_{i_j}. Therefore, $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j}$. Since N is an integer flow network with no cost-free arcs, from Corollary 12, it follows that $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j} \geq i_j \phi_{i_j} \geq j \phi_{i_j}$. Therefore, $\chi \geq \chi_S = \sum_{1 \leq j \leq l} \chi_{i_j} \geq \sum_{1 \leq j \leq l} j \phi_{i_j}$. It is easy to see that $\partial\chi/\partial\phi_{i_j} = j$ and $\partial\phi_S/\partial\phi_{i_j} = 1$. Therefore, we have that $\partial\chi/\partial\phi_{i_j} < \partial\chi/\partial\phi_{i_{j+1}}$, whereas $\partial\phi_S/\partial\phi_{i_j} = \partial\phi_S/\partial\phi_{i_{j+1}}$. Since $\phi_{i_j} \leq \sqrt{\chi}$, it follows that for a given χ, ϕ_S is

maximum, when the values of l and the ϕ_{i_j}'s are such that $\phi_{i_1} = \phi_{i_2} = \cdots = \phi_{i_{l-1}} = \sqrt{\chi}$ and $1 \leq \phi_{i_l} \leq \sqrt{\chi}$. Because $\chi \geq \chi_S \geq \sum_{1 \leq j \leq l} j\phi_{i_j}$, it follows that when ϕ_S is maximum, l is such that $\chi > \sum_{1 \leq j \leq l-1} j\sqrt{\chi} = (l(l-1)/2)\sqrt{\chi}$. Therefore, when ϕ_S is maximum, $l^2 - l < 2\sqrt{\chi}$, and hence, $l = O(\chi^{1/4})$. Therefore, the maximum value of ϕ_S is at most $\sum_{1 \leq j \leq l} \sqrt{\chi} = l\sqrt{\chi} = O(\chi^{1/4})\sqrt{\chi} = O(\chi^{3/4})$. □

Lemma 42 *Let $k(\alpha)$ be the number of indices such that for each index i, $\phi_i > \chi^\alpha$, then $k(\alpha) < \sqrt{2}\chi^{(1-\alpha)/2}$.*

Proof: Let $f_{i_1}, f_{i_2}, \ldots, f_{i_l}$, where $i_1 < i_2 \ldots < i_l = k(\alpha)$, be the set of all the flows for which $\phi_{i_j} > \chi^\alpha$. From the definition of flow network A_{i_j}, all the augmenting paths of A_{i_j} have the same cost c_{i_j}. Therefore, $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j}$. Since N is an integer flow network with no cost-free arcs, from Corollary 12, it follows that $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j} \geq i_j\phi_{i_j} \geq j\phi_{i_j}$. We have that, $\chi \geq \sum_{1 \leq j \leq l} \chi_{i_j} = \sum_{1 \leq j \leq l} j\phi_{i_j} > \sum_{1 \leq j \leq l} j\chi^\alpha = (l(l+1)/2)\chi^\alpha$. Therefore, $2\chi^{1-\alpha} > l^2 + l \geq l^2$. Hence, $k(\alpha) = l < \sqrt{2}\chi^{(1-\alpha)/2}$. □

Because, $|B| = k(1/2)$, it follows immediately from Lemma 42 that,

Corollary 13 $|B| \leq \sqrt{2}\chi^{1/4}$.

Lemma 43 gives an upper bound on the time used by *Blockflow* for computing the max-flow f_i of A_i.

Lemma 43 *Algorithm* BlockFlow *computes the max-flow f_i of A_i in time $O(\sqrt{\chi}m_i \log n_i + m_i \log n_i)$, where n_i and m_i are the number of nodes and arcs respectively of A_i.*

Proof: Recall the description of Algorithm *BlockFlow* given in Section 7.3. From Lemma 31(3), a phase j can be executed in time $O(m_j \log n_j)$. Let r be the number of phases used by Algorithm *BlockFlow* to compute f_i. We now show that r is at most $3\sqrt{\chi}+1$. This will prove that Algorithm *BlockFlow* computes f_i in time $O(\sqrt{\chi}m_i \log n_i + m_i \log n_i)$.

If $\phi_i < \sqrt{\chi}$, then because from Lemma 31(1), every phase, except the last one, increases flow by at least one unit, r is at most $\phi_i + 1 < \sqrt{\chi} + 1$. Hence, suppose $\phi_i \geq \sqrt{\chi}$. Let l be the phase that increases the flow in A_i to at least $\phi_i - \sqrt{\chi}$. Because from Lemma 31(1), every phase, except the last one, increases the flow by at least one unit, the number of phases after phase l is at most $\sqrt{\chi} + 1$. Hence, if we show that $l \leq 2\sqrt{\chi}$, we are done. From lemma 31(2), each phase increases the depth of the layered network by at least one, hence if we show that the depth d of the layered network used for phase l is at most $2\sqrt{\chi}$, we are done. Now we show that $d < 2\sqrt{\chi}$. Our proof is based on the following idea: We first compute the costs of the flows established in A_i along paths with length at least d. Using the costs of these flows, we compute the cost α of the total flow in N at the end of the phase r. Then we show that because α is at most χ, d is less than $2\sqrt{\chi}$. We now present the proof formally.

Let f be the total flow established in N (since the beginning of the execution of Algorithm *compute-flow*) at the end of phase $l-1$. Let $R(N, f)$ be the residual network of

N with respect to flow f. Let f' be the total flow established in A_i in phases $1, 2, \ldots, l-1$. Let $R(A_i, f')$ be the residual network of A with respect to flow f'. Notice that $R(A_i, f')$ is a subgraph of $R(N, f)$. Let V_q be the set of nodes of $R(A_i, f')$ such that for each node $v \in V_q$, the length of the shortest capacitated path in $R(A_i, f')$ from s to v is exactly q. Let E_q be the set of arcs from nodes in set V_q to nodes in set V_{q+1}. Because $R(A_i, f')$ has depth at least d, Each arc-set E_q for $0 \leq q \leq d-1$ is a st-cut of $R(A_i, f')$. Also, no two distinct arc-sets E_{q_1} and E_{q_2} have any arc in common. Let ϕ^* be the total magnitude of the flows established in $R(A_i, f')$ by Algorithm *BlockFlow* in phases $l, l+1, \ldots, r$. From the definition of phase l, we have that $\phi^* > \sqrt{\chi}$. Therefore, at the end of phase r, the total flow established in $R(A_i, f)$ consists of ϕ^* flows from s to t, $g_1, g_2, \ldots, g_{\phi^*}$, where each g_j has magnitude exactly one. Because each arc-set E_q is a st-cut of $R(A_i, f')$, from Lemma 29, each g_j "flows through" at least one arc of each E_q for $0 \leq q \leq d-1$, i.e., for each pair (j, q), where $1 \leq j \leq g_{\phi^*}$, and $0 \leq q \leq d-1$, there is an arc e in E_q, such that $g_j(e) = 1$. Since, no two distinct arc-sets E_{q_1} and E_{q_2} have any arc in common, it follows that the number of arcs of $R(A_i, f')$ in which flow g_j is established is at least d.

Let $W_j = \{e \in R(A_i, f') | c(e) > 0 \text{ and } g_j(e) = 1\}$. Let $B_j = \{e \in R(A_i, f') | c(e) < 0 \text{ and } g_j(e) = 1\}$. Thus, W_j is the set of forward arcs of $R(A_i, f')$ in which flow g_j is established, and B_j is the set of backward arcs of $R(A_i, f')$ in which flow g_j is established. Let $c(W_j) = \sum_{e \in W_j} c(e)$. Let $c(B_j) = \sum_{e \in B_j} c(e)$. Therefore, the cost of flow g_j is equal to $c(W_j) + c(B_j)$. We have shown earlier that the number of arcs of $R(A_i, f')$ in which flow g_j is established is at least d. Therefore, $|W_j| + |B_j| \geq d$. Because N has no cost free arcs, $c(W_j) \geq |W_j|$ and $-c(B_j) \geq |B_j|$. Therefore, $c(W_j) - c(B_j) \geq d$. Even though the cost of a backward arc is negative, from Corollary 12, the cost of each and every augmenting path of A_i is non-negative, and hence from Lemma 32, the cost of each and every augmenting paths of $R(A_i, f')$ is non-negative. Therefore, the cost of g_j is non-negative. Since, $c(g_j) = c(W_j) + c(B_j)$, we have that $c(W_j) \geq -c(B_j)$. Therefore $2c(W_j) \geq c(W_j) - c(B_j) \geq d$. Hence, $c(W_j) \geq d/2$.

Let $T = \cup_{1 \leq j \leq \phi^*} B_j$. Let α be the cost of the total flow established in N (since the beginning of the execution of Algorithm *compute-flow*) at the end of phase r. From Corollary 12, it follows that $\chi \geq \alpha$. Therefore,

$$\begin{aligned}
\chi &\geq \alpha = c(f) + \sum_{1 \leq j \leq \phi^*} c(g_j) \\
&\geq c(f) + \sum_{1 \leq j \leq \phi^*} (c(W_j) + c(B_j)) \\
&\geq \sum_{1 \leq j \leq \phi^*} c(W_j) + c(f) + \sum_{1 \leq j \leq \phi^*} c(B_j) \\
&\geq \sum_{1 \leq j \leq \phi^*} c(W_j) + c(f) + \sum_{e \in T} f_i(e) c(e) \qquad (7.1)
\end{aligned}$$

Recall that $R(A_i, f')$ is a subgraph of $R(N, f)$. From the definition of a backward arc, for every backward arc $e = (u, v)$ in g_j, there is a forward arc $e' = (v, u)$ in $R(N, f)$ such that $f(e') = u(e)$, and $c(e') = -c(e)$. Because for each arc e in T, $f_i(e) \leq u(e)$, it follows that $c(f) + \sum_{e \in T} f_i(e) c(e) \geq 0$. Therefore from Eq 7.1, $\chi \geq \sum_{1 \leq j \leq \phi^*} c(W_j)$. We have shown earlier that for each g_j, $c(W_j) \geq d/2$. Therefore, $\chi \geq \sum_{1 \leq j \leq \phi^*} d/2 \geq \phi^* \cdot d/2$. Recall that $\phi^* > \sqrt{\chi}$. Therefore, $\chi > \sqrt{\chi} d/2$. Hence, $d < 2\sqrt{\chi}$. □

Lemma 44 *Let N be an integer flow network with n nodes and m arcs, such that N has no cost-free arc, and N admits a flow with cost at most χ. We can compute a min-cost max-flow f of N in time $O(\chi^{3/4} m \log n + m \log n)$.*

Proof: Let r be the total number of stages used by *Algorithm compute-flow* to compute f. Each stage i, for $i \leq k-1$, increases the amount of flow in N by at least 1. Therefore, $\phi_i \geq 1$ for each $i \leq r-1$. Let i be a non-negative integer less than r. Let n_i and m_i be the number of nodes and arcs respectively in N_i. Let T_i, T_{S_i}, and T_{B_i} be the time taken by algorithms *compute-flow*, *SAP* and *BlockFlow* respectively, for computing f_i in N_i. We have that $T_i = O(\min(T_{S_i}, T_{B_i}))$. Since $\phi_i \geq 1$, from Lemma 30, $T_{S_i} = O(\phi_i \cdot m_i \log n_i)$. Since $\phi_i \geq 1$, and N has no cost free arcs, we have that $\chi \geq 1$. Therefore, from Lemma 43, $T_{B_i} = O(\sqrt{\chi} m_i \log n_i)$. Therefore, $T_i = O(T_{S_i})$ if $\phi_i \leq \sqrt{\chi}$, and $T_i = O(T_{B_i})$, otherwise. In other words, if $f_i \in S$, then $T_i = O(T_{S_i})$, and if $f_i \in B$, then $T_i = O(T_{B_i})$. Let T be the total time taken by *Algorithm compute-flow* to compute f. Therefore, from Lemma 34,

$$\begin{aligned}
T &= \sum_{1 \leq i \leq r-1} T_i + O(m \log n) \\
&= \sum_{f_i \in S} T_{S_i} + \sum_{f_i \in B} T_{B_i} + O(m \log n) \\
&= \sum_{f_i \in S} O(\phi_i \cdot m_i \log n_i) + \sum_{f_i \in B} O(\sqrt{\chi} m_i \log n_i) + O(m \log n) \\
&= O(m \log n) \sum_{f_i \in S} \phi_i + (\sum_{f_i \in B} 1) \cdot O(\sqrt{\chi} m \log n) + O(m \log n) \\
&= \phi_S \cdot O(m \log n) + |B| \cdot O(\sqrt{\chi} m \log n) + O(m \log n)
\end{aligned}$$

From Lemma 41, $\phi_S = O(\chi^{3/4})$. From Corollary 13, $|B| \leq \sqrt{2}\chi^{1/4}$. Therefore, $T = O(\chi^{3/4}) \cdot O(m \log n) + \sqrt{2}\chi^{1/4} \cdot O(\sqrt{\chi} m \log n) + O(m \log n) = O(\chi^{3/4} m \log n + m \log n)$. □

7.6 Conclusion

We now give the main theorem of this chapter.

Theorem 26 *Let G be an embedded degree 4 planar graph with n vertices, then we can construct an orthogonal planar drawing of G with minimum number of bends in $O(n^{1.75} \log n)$ time.*

Proof: Let N(G) be the associated flow network of G. From Theorem 25, N(G) is an integer flow network with no cost free arcs, consists of $O(n)$ nodes and arcs, and admits a min-cost max-flow with cost $O(n)$. Therefore, from Lemma 44, we can compute a min-cost max-flow f of N(G) in time $O(n^{3/4} n \log n + n \log n) = O(n^{1.75} \log n)$. From Theorem 25, it follows that we can construct an orthogonal planar drawing of G with minimum number of bends in $O(n^{1.75} \log n)$ time. □

Chapter 8

Conclusions and Open Problems

In this thesis, we have considered the problem of constructing four different kinds of drawings: upward planar drawings, planar straight-line drawings with large angular resolution, angle-preserving drawings (i.e. drawings of angle graphs), and orthogonal planar drawings with minimum number of bends. We have also presented results that hold individually for other kinds of drawings.

Our results are summarized as follows:

- We have shown that the upward planarity testing problem is NP-complete, and hence a polynomial-time algorithm is unlikely to exist.
- We have provided matching upper and lower bounds for the area-requirements of both polyline and orthogonal drawings of rooted trees. We have presented linear-time algorithms for constructing such drawings with areas that match the respective lower bounds within a constant factor.
- We have given the first non-trivial upper bound on angular resolution, namely, we have shown that there exists an infinite family of degree-d planar graphs that requires $O(\sqrt{\log d/d^3})$ angular resolution in any planar straight-line drawing.
- We have shown a trade-off between the area-requirement and angular resolution of a planar straight-line drawing, which implies that an infinite family of planar graphs requires exponential area for any drawing with angular resolution that is independent of d and n, where n is the number of vertices in a graph.
- We have also given linear-time algorithm for constructing planar straight-line drawings of outerplanar, star-nested and series-parallel graphs with angular resolution that is a polynomial in the degree of the graph, and is independent of the number of vertices in the graph.
- We have shown that testing a consistent angle graph for planarity is NP-hard.
- We have shown that testing a bilayered consistent angle graph, in which each angle is a multiple of 90°, for biplanarity is also NP-hard.

- We have given a linear-time algorithm for testing whether a given series-parallel angle graph admits a drawing.
- We have studied the area-requirement of angle graphs, and have shown that an infinite family of angle graphs requires exponential area in any drawing.
- Using our NP-hardness result for the planarity testing of angle graphs, we have shown that the problem of constructing a planar straight-line drawing with maximum angular resolution, of a triconnected planar graph is also NP-hard.
- We have shown that the rectilinear planarity testing problem is NP-hard. In fact, we have also shown that given an n-vertex graph, even approximating the number of bends within $O(n^{1-\epsilon})$ of the optimal, for any ϵ greater than 0, is NP-hard.
- We have given a new algorithm with time-complexity $O(n^{1.75} \log n)$ for constructing a bend-minimum planar orthogonal drawing of an n-vertex embedded graph. Our algorithm is faster than the previously known fastest algorithm [99] by a factor of $n^{0.25}/\log n$.

Our results leave several problems open, and also give rise to many interesting problems:

- Characterize graphs for which upward planarity testing can be done efficiently.
- Find an algorithm for approximating angular resolution within a small factor of the optimal.
- Either show that the $O(\sqrt{\log d/d^3})$ upper bound on angular resolution is also tight by giving a matching lower bound, or give a tighter upper bound.
- Prove or disprove whether every planar graph has angular resolution that is a polynomial in its degree. Notice that the result of [83] shows that every planar graph has angular resolution that is exponential in its degree.
- Find tighter lower bound for the area-requirement of straight-line drawings, with large angular resolution, of a planar graph. Also find matching upper and lower bounds for the area-requirement.
- Devise algorithms that are parameterized for area and angular resolution, i.e., they construct drawings with large or small area, depending upon the desired angular resolution.
- Notice that the area-requirement of a graph can sometimes be reduced by allowing its edges to have bends in its drawings. Study the trade-off between area, angular resolution and the number of bends.
- Characterize angle graphs for which planarity testing can be done efficiently.
- Find an efficient algorithm for testing an angle graph for consistency, i.e., testing whether it admits a drawing. The current best approach [110] is based on linear programming, and hence has a time-complexity that is a high-degree polynomial in the number of edges and vertices in the graph.

- Characterize graphs for which rectilinear planarity testing can be done efficiently.
- Notice that rectilinear planarity testing can be done in polynomial time for degree-3 graphs [33]. Devise an algorithm whose time-complexity is exponential in the number of degree-four vertices in the graph.
- Improve the time-complexity of bend-minimization for embedded graphs, series-parallel graphs and degree-3 graphs.
- Study the problem of bend-minimization in three-dimensions, and on other surfaces such as sphere, cylinder, and torus etc.
- Study the trade-off between the area-requirement of a drawing and the number of bends in the drawing. Devise algorithms that are parameterized for area and number of bends.

Bibliography

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] R. Arrathoon, editor. *Optical Computing: Digital and Symbolic*. Marcel Dekker, Inc., 1989.

[3] C. Batini, L. Furlani, and E. Nardelli. What is a good diagram? A pragmatic approach. In *Proc. 4th Internat. Conf. on the Entity Relationship Approach*, 1985.

[4] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.

[5] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. How to draw a series-parallel digraph. In *Proc. 3rd Scand. Workshop Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 272–283. Springer-Verlag, 1992.

[6] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. How to draw a series-parallel digraph. *Internat. J. Comput. Geom. Appl.*, 4:385–402, 1994.

[7] P. Bertolazzi and G. Di Battista. On upward drawing testing of triconnected digraphs. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 272–280, 1991.

[8] P. Bertolazzi, G. Di Battista, and G. Liotta. Parametric graph drawing. Technical Report 6/67, IASI–CNR, Rome, Italy, 1992.

[9] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, to appear.

[10] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia. Optimal upward planarity testing of single-source digraphs. In *1st Annual European Symposium on Algorithms (ESA '93)*, volume 726 of *Lecture Notes in Computer Science*, pages 37–48. Springer-Verlag, 1993.

[11] S. Bhatt and S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Inform. Process. Lett.*, 25:263–267, 1987.

[12] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *J. Comput. Syst. Sci.*, 28:300–343, 1984.

[13] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. In *Proc. 2nd Annu. European Sympos. Algorithms (ESA '94)*, volume 855 of *Lecture Notes in Computer Science*, pages 24–35. Springer-Verlag, 1994.

[14] K. Booth and G. Lueker. Testing for the consecutive ones property interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[15] F. J. Brandenburg. Nice drawings of graphs and trees are computationally hard. Technical Report MIP-8820, Fakultat fur Mathematik und Informatik, Univ. Passau, 1988.

[16] R. P. Brent and H. T. Kung. On the area of binary tree layouts. *Inform. Process. Lett.*, 11:521–534, 1980.

[17] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[18] N. Chiba, T. Yamanouchi, and T. Nishizeki. Linear algorithms for convex drawings of planar graphs. In J. A. Bondy and U. S. R. Murty, editors, *Progress in Graph Theory*, pages 153–173. Academic Press, New York, NY, 1984.

[19] M. Chrobak and T. H. Payne. A linear time algorithm for drawing a planar graph on a grid. Technical Report UCR-CS-90-2, Dept. of Math. and Comput. Sci., Univ. California Riverside, 1990.

[20] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.

[21] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom. Theory Appl.*, 2:187–200, 1992.

[22] P. Crescenzi and A. Piperno. Optimal-area upward drawings of AVL trees. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 307–317. Springer-Verlag, 1995.

[23] I. F. Cruz and A. Garg. Drawing graphs by example efficiently: Trees and planar acyclic digraphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 404–415. Springer-Verlag, 1995.

[24] I. F. Cruz, R. Tamassia, and P. Van Hentenryk. A visual approach to graph drawing. In *Graph Drawing '93 (Proc. ALCOM Workshop on Graph Drawing)*, Paris, France, September 1993.

[25] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *Commun. ACM*. To appear.

[26] H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fary embeddings of planar graphs. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 426–433, 1988.

[27] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[28] E. Dengler, M. Friedell, and J. Marks. Constraint-driven diagram layout. In *Proc. IEEE Sympos. on Visual Languages (VL '93)*, pages 330–335, 1993.

[29] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. Preprint, Dept. Comput. Sci., Brown Univ., Providence, RI, November 1993. Preliminary version available via anonymous `ftp` from `wilma.cs.brown.edu`, `gdbiblio.tex.Z` and `gdbiblio.ps.Z` in `/pub/papers/compgeo`.

[30] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[31] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of three graph drawing algorithms. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995.

[32] G. Di Battista, G. Liotta, M. Strani, and F. Vargiu. Diagram Server. In *Advanced Visual Interfaces (Proceedings of AVI '92)*, volume 36 of *World Scientific Series in Computer Science*, pages 415–417, 1992.

[33] G. Di Battista, G. Liotta, and F. Vargiu. Spirality of orthogonal representations and optimal drawings of series-parallel graphs and 3-planar graphs. In *Proc. Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 151–162. Springer-Verlag, 1993.

[34] G. Di Battista, G. Liotta, and F. Vargiu. Diagram Server. *J. Visual Languages and Computing*, 6(3), 1995. (special issue on Graph Visualization, edited by I. F. Cruz and P. Eades).

[35] G. Di Battista, W. P. Liu, and I. Rival. Bipartite graphs upward drawings and planarity. *Inform. Process. Lett.*, 36:317–322, 1990.

[36] G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoret. Comput. Sci.*, 61:175–198, 1988.

[37] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete Comput. Geom.*, 7:381–401, 1992.

[38] G. Di Battista and L. Vismara. Angles of planar triangular graphs. In *Proc. 25th Annu. ACM Sympos. Theory Comput. (STOC 93)*, pages 431–437, 1993.

[39] D. Dolev, F. T. Leighton, and H. Trickey. Planar embedding of planar graphs. In F. P. Preparata, editor, *Advances in Computing Research*, volume 2, pages 147–161. JAI Press, Greenwich, Conn., 1985.

[40] P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.

[41] P. Eades and T. Lin. Algorithmic and declarative approaches to aesthetic layout. In *Graph Drawing '93 (Proc. ALCOM Workshop on Graph Drawing)*, Paris, France, September 1993.

[42] P. Eades, T. Lin, and X. Lin. Two tree drawing conventions. Technical Report 174, Department of Computer Science, University of Queensland, 1990. to appear in *Comput. Geom. Theory Appl.*

[43] P. Eades, T. Lin, and X. Lin. Minimum size h-v drawings. In *Advanced Visual Interfaces (Proceedings of AVI '92)*, volume 36 of *World Scientific Series in Computer Science*, pages 386–394, 1992.

[44] P. D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[45] S. Even and G. Granot. Rectilinear planar drawings with few bends in each edge. Technical Report 797, Computer Science Dept., Technion, 1994.

[46] L.R. Ford and D.R. Fulkerson. A primal-dual algorithm for the capacitated hitchcock problem. *Naval Research Logistics Quarterly*, 4:47–54, 1957.

[47] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

[48] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Simvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.*, 22:1035–1052, 1993.

[49] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Simvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. In *Proc. 31th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 86–95, 1990.

[50] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[51] A. Garg. On drawing angle graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 84–95. Springer-Verlag, 1995.

[52] A. Garg, M. T. Goodrich, and R. Tamassia. Area-optimal upward tree drawings. *Computational Geometry, Theory and Appln.* To appear.

[53] A. Garg, M. T. Goodrich, and R. Tamassia. Area-efficient upward tree drawings. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 359–368, 1993.

[54] A. Garg and R. Tamassia. Upward planarity testing. *Order.* To appear.

[55] A. Garg and R. Tamassia. Angular resolution of planar drawings. Technical report, Brown Univ., Dept. of Computer Science, 1993.

[56] A. Garg and R. Tamassia. Advances in graph drawing. In *Algorithms and Complexity (Proc. CIAC' 94)*, volume 778 of *Lecture Notes in Computer Science*, pages 12–21. Springer-Verlag, 1994.

[57] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. Report CS-94-10, Comput. Sci. Dept., Brown Univ., Providence, RI, 1994.

[58] A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In *Proc. 2nd Annu. European Sympos. Algorithms (ESA '94)*, volume 855 of *Lecture Notes in Computer Science*, pages 12–23. Springer-Verlag, 1994.

[59] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 286–297. Springer-Verlag, 1995.

[60] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[61] M. Himsolt. Comparing and evaluating layout algorithms within graphed. Manuscript, Fakultat fur Mathematik und Informatik, Univ. Passau, 1994.

[62] M. Himsolt. Comparing and evaluating layout algorithms within GraphEd. *J. Visual Languages and Computing*, 6(3), 1995. (special issue on Graph Visualization, edited by I. F. Cruz and P. Eades).

[63] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[64] M. D. Hutton and A. Lubiw. Upward planar drawing of single source acyclic digraphs. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 203–211, 1991.

[65] T. Kamada. *On Visualization of Abstract Objects and Relations.* PhD thesis, Department of Information Science, University of Tokyo, 1988.

[66] T. Kamada. *Visualizing Abstract Objects and Relations.* World Scientific Series in Computer Science, 1989.

[67] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica.* (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia, to appear).

[68] G. Kant. Drawing planar graphs using the *lmc*-ordering. In *Proc. 33th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 101–110, 1992.

[69] G. Kant. *Algorithms for Drawing Planar Graphs.* PhD thesis, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1993.

[70] G. Kant. A more compact visibility representation. In *Proc. 19th Internat. Workshop Graph-Theoret. Concepts Comput. Sci. (WG'93)*, 1993.

[71] G. Kant, G. Liotta, R. Tamassia, and I. Tollis. Area requirement of visibility representations of trees. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 192–197, Waterloo, Canada, 1993.

[72] Goos Kant. Hexagonal grid drawings. In *Proc. 18th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, page ??, 1992.

[73] D. Kelly. Fundamentals of planar ordered sets. *Discrete Math.*, 63:197–216, 1987.

[74] D. Kelly and I. Rival. Planar lattices. *Canad. J. Math.*, 27(3):636–665, 1975.

[75] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270–281, 1980.

[76] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: Internat. Symposium (Rome 1966)*, pages 215–232, New York, 1967. Gordon and Breach.

[77] R. Liggett and E. Mitchell. Optimal space planning in practice. *Computer Aided Design*, 13:277–288, 1981.

[78] Y. Liu, P. Marchioro, and R. Petreschi. A single bend embedding algorithm for cubic graphs. Manuscript, 1994.

[79] Y. Liu, P. Marchioro, R. Petreschi, and B. Simeone. Theoretical results on at most 1-bend embeddability of graphs. Technical report, Dipartimento di Statistica, Univ. di Roma "La Sapienza", 1990.

[80] Y. Liu, A. Morgana, and B. Simeone. General theoretical results on rectilinear embeddability of graphs. *Acta Math. Appl. Sinica*, 7:187–192, 1991.

[81] Y. Liu, A. Morgana, and B. Simeone. A linear algorithm for 3-bend embeddings of planar graphs in the grid. Manuscript, 1993.

[82] S. Malitz and A. Papakostas. On the angular resolution of planar graphs. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 527–538, 1992.

[83] S. Malitz and A. Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.*, 7:172–183, 1994.

[84] J. Manning and M. J. Atallah. Fast detection and display of symmetry in trees. *Congr. Numer.*, 64:159–169, 1988.

[85] K. Mehlhorn. *Graph Algorithms and NP-Completeness*, volume 2 of *Data Structures and Algorithms*. Springer-Verlag, Heidelberg, West Germany, 1984.

[86] A. Papakostas. Upward planarity testing of outerplanar dags. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 298–306. Springer-Verlag, 1995.

[87] A. Papakostas and I. G. Tollis. Improved algorithms and bounds for orthogonal drawings. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 40–51. Springer-Verlag, 1995.

[88] C. Platt. Planar lattices and planar graphs. *J. Combin. Theory Ser. B*, 21:30–39, 1976.

[89] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223–228, 1981.

[90] I. Rival. The diagram. In I. Rival, editor, *Graphs and Orders*, pages 103–133. Reidel Publishing, 1985.

[91] I. Rival. Graphical data structures for ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 3–31. Kluwer Academic Publishers, 1989.

[92] I. Rival. Reading, drawing, and order. In I. G. Rosenberg and G. Sabidussi, editors, *Algebras and Orders*, pages 359–404. Kluwer Academic Publishers, 1993.

[93] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.*, 1(4):343–353, 1986.

[94] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.

[95] Y. Shiloach. *Arrangements of Planar Graphs on the Planar Lattice*. PhD thesis, Weizmann Institute of Science, 1976.

[96] D.D. Sleator. *An $O(nm \log n)$ Algorithm for Maximum Network Flow*. PhD thesis, Stanford University, 1980.

[97] J. A. Storer. On minimal node-cost planar embeddings. *Networks*, 14:181–212, 1984.

[98] K. J. Supowit and E. M. Reingold. The complexity of drawing trees nicely. *Acta Inform.*, 18:377–392, 1983.

[99] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[100] R. Tamassia. A dynamic data structure for planar graph embedding. In T. Lepisto and A. Salomaa, editors, *Automata, Languages and Programming (Proc. 15th ICALP)*, volume 317 of *Lecture Notes in Computer Science*, pages 576–590. Springer-Verlag, 1988.

[101] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.*, 1(4):321–341, 1986.

[102] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. on Circuits and Systems*, CAS-36(9):1230–1234, 1989.

[103] R. Tamassia, I. G. Tollis, and J. S. Vitter. Lower bounds and parallel algorithms for planar orthogonal grid drawings. In *Proc. IEEE Symposium on Parallel and Distributed Processing*, pages 386–393, 1991.

[104] R. Tamassia and J. S. Vitter. Parallel transitive closure and point location in planar structures. *SIAM J. Comput.*, 20(4):708–725, 1991.

[105] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial Applied Mathematics, 1983.

[106] Teamwork. Cadre Technologies, Providence, RI.

[107] C. Thomassen. Planar acyclic oriented graphs. *Order*, 5(4):349–361, 1989.

[108] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.

[109] L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.

[110] G. Vijayan. Geometry of planar graphs with angles. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 116–124, 1986.

[111] G. Vijayan and A. Wigderson. Rectilinear graphs and their embeddings. *SIAM J. Comput.*, 14:355–372, 1985.

[112] C. Wetherell and A. Shannon. Tidy drawing of trees. *IEEE Trans. Softw. Eng.*, SE-5(5):514–520, 1979.

[113] D. Woods. *Drawing Planar Graphs*. PhD thesis, Department of Computer Science, Stanford University, 1982. Technical Report STAN-CS-82-943.