Dominance Drawing of Non-Planar Graphs

Evgenios M. Kornaropoulos

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Knossou Av., P.O. Box 2208, Heraklion, GR-71409, Greece

Thesis Advisor: Prof. *Ioannis G. Tollis*

This work has been performed at the **University of Crete, School of Sciences and Engineering, Computer Science Department.**

The work is partially supported by the **Foundation of Research and Technology Hellas, Institute of Computer Science.**

University of Crete
Computer Science Department

Dominance Drawing of Non-Planar Graphs

Thesis submitted by
Evgenios M. Kornaropoulos
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Evgenios M. Kornaropoulos

Committee approvals: _____
Ioannis G. Tollis
Professor, Thesis Supervisor

_____
Menelaos I. Karavelas
Assistant Professor, Committee Member

_____
Yannis Tzitzikas
Assistant Professor, Committee Member

Departmental approval: _____
Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, June 2012

Abstract

In information technology, big data is a term used to describe data sets so large and complex that they become awkward to work and analyze. Difficulties include capture, storage, search, sharing, analysis, and visualization. In this work we focus in the latter part, specifically we study the visualization of graphs. The need for simple and intuitive visualization techniques is more imminent than ever.

In this thesis we propose a new framework for visualizing graphs. We extend the so-called Dominance Drawing method, by relaxing some constraints in order to include the much broader family of directed acyclic graphs. We introduce a new computational problem called *Weak Dominance Placement*, which we prove NP-complete using notions from the field of Order Theory. We offer simple bounds and properties as well as three heuristic algorithms in order to obtain locally optimal solutions.

Using a weak dominance placement of the vertices, we propose a new graph drawing model for visualizing directed acyclic graphs called *Overloaded Orthogonal*. In order to simplify our drawings we use an overloading technique for edge routing. All algorithms are simple and easy to implement and can be applied to directed acyclic graphs of arbitrary degree, planar, and non-planar graphs. We also present bounds on the number of bends and the layout area of a graph. Overloaded Orthogonal drawings present several interesting properties such as efficient visual edge confirmation as well as simplicity and clarity of the drawing.

Furthermore, we propose the *DAGView* framework which handles not only directed acyclic graphs, but also graphs with cycles and undirected graphs. Our approach combines the readability and scalability of a matrix based approach, with the intuitiveness of a node-link approach in spotting a node within the layout and in following an edge to find its destination. We have implemented the DAGView framework in Java and the results are very encouraging. Finally, we believe that DAGView visualizations will be well accepted in user studies, since several criteria that users identified as important in a layout are met: underlying grid, crossings that appear perpendicular, easy check for the existence of an edge and/or path, preservation of the mental-map.

Περίληψη

Η εκθετική αύξηση του όγκου δεδομένων στην εποχή μας χαρακτηρίζεται από σύνολα δεδομένων τόσο μεγάλα και περίπλοκα, που η επεξεργασία και η ανάλυσή τους είναι σχεδόν αδύνατη. Μια σειρά από δυσκολίες που παρουσιάζονται είναι η συλλογή, η αποθήκευση, η αναζήτηση, η ανάλυση και η απεικόνιση των δεδομένων. Σε αυτή την εργασία μελετάμε το τελευταίο απο τα παραπάνω προβλήματα, την απεικόνιση γράφων. Η χρήση απλών και διαισθητικών τεχνικών οπτικοποίησης κρίνεται πιο αναγκαία από ποτέ.

Επεκτείνουμε την γνωστή τεχνική Κυριαρχικής Απεικόνισης (Dominance Drawing) χαλαρώνοντας ορισμένους περιορισμούς, έτσι ώστε να συμπεριλάβουμε την ευρύτερη οικογένεια κατευθυνόμενων άκυκλων γράφων (DAG). Παρουσιάζουμε ένα νέο υπολογιστικό πρόβλημα το οποίο ορίζουμε ως Τοποθέτηση Υπό Ασθενή Κυριαρχία (Weak Dominance Placement), και αποδεικνύουμε την NPπληρότητά του χρησιμοποιώντας έννοιες από τον κλάδο της Θεωρίας Διάταξης. Αναφέρουμε ορισμένα φράγματα και ιδιότητες αλλά και τρείς ευριστικούς αλγορίθμους για την εύρεση τοπικά βέλτιστων λύσεων.

Χρησιμοποιώντας μια τοποθέτηση υπό ασθενή κυριαρχία, προτείνουμε μια νέα τεχνική απεικόνισης κατευθυνόμενων άκυκλων γράφων που ονομάζεται Επικαλυπτόμενη Ορθογωνική Απεικόνιση (Overloaded Orthogonal Drawing). Για να απλοποιηθούν οι οπτικοποιήσεις χρησιμοποιούμε μια τεχνική υπερφόρτωσης για την δρομολόγηση των ακμών του γράφου. Οι προτεινόμενοι αλγόριθμοι είναι απλοί και εύκολα υλοποιήσιμοι σε κατευθυνόμενους άκυκλους γράφους ανεξάρτητα από το βαθμό των κόμβων, σε επίπεδους αλλά και σε μη-επίπεδους γράφους. Ακόμη παρουσιάζουμε φράγματα για τον αριθμό σημείων ανάκαμψης (bends) αλλά και για τον καταλαμβανόμενο χώρο της απεικόνισης. Η Επικαλυπτόμενη Ορθογωνική Απεκόνιση παρουσιάζει ορισμένα πολύ ενδιαφέροντα χαρακτηριστικά όπως την αποτελεσματική οπτική επιβεβαίωση ακμής, την απλότητα και την σαφήνεια οπτικοποιήσης.

Επιπρόσθετα, εισάγουμε το πλαίσιο οπτικοποιήσης DAGView το οποίο μπορεί να απεικονίσει όχι μονο κατευθυνόμενους άκυκλους γράφους αλλά και γράφους με κύκλους, όπως και μη-κατευθυνόμενους γράφους. Η προσέγγισή μας συνδυάζει την αναγνωσιμότητα και την κλιμάκωση των μεθόδων απεικόνισης με πίνακα (matrix representation), με τη διαισθητικότητα που παρουσιάζουν οι γνωστές τεχνικές απεικόνισης. Υλοποιήσαμε την μέθοδο οπτικοποίησης DAGView σε Java και τα αποτελέσματα είναι πολύ ενθαρρυντικά. Τέλος, πιστεύουμε ότι το πλαίσιο οπτικοποίησης DAGView θα είναι καλώς αποδεκτό σε Μελέτες Χρηστών, δεδομένου ότι οι οπτικοποιήσεις παρουσιάζουν αρκετά κριτήρια που προσδιορίζονται από τους ίδιους τους χρήστες ως σημαντικά οπως: η απεικόνιση σε υποκείμενο πλέγμα, τα κάθετα σημεία τομής, ο εύκολος έλεγχος για την ύπαρξη ακμής ή / και μονοπατιού, η διατήρηση του νοητικού χάρτη.

Acknowledgements

There are so many people that I would like to thank, each one helped me with their own special way. First of all, I would like to thank my advisor Professor Ioannis G. Tollis. He has been a tireless source of inspiration, encouragement and ideas during my graduate studies. His office door was always open for advice and help. For all that and much more, I am grateful.

I would also like to thank Professor Menelaos Karavelas and Professor Yannis Tzitzikas for contributing as members of my Masters committee. Their comments on my work helped me to form the final version of my thesis. Furthermore, I would like to thank Professor Panagiotis Tsakalides for being my advisor during my undergraduate years. His positive attitude and his academic guidance were more than helpful. Also, I would like to thank Professor Ioannis Tsamardinos for his inspiring classes and for our discussions on academic matters.

I need to express my gratitude to the University of Crete and the Department of Computer Science for providing me with proper education; as well as the Institute of Computer Science of the Foundation for Research and Technology (ICS-FORTH) for supporting me. Special thanks go to the secretaries of the department Rena Kalaitzaki and Gelly Kosma for assisting me in so many procedural tasks during my studies.

Moreover, I would like to thank my graduate colleagues for being there for me: Giorgos Chinis, Dimitra Emmanouilidou, George Kafentzis, Evangelia Maniadi, Ioannis Manousakis, Fotis Nikolaidis, Yannis Pantazis, Vasilis Tsiaras and Miltiadis Vasilakis. I would like to give my appreciation to several dear friends from my years as an undergrad: Yannis Giokas, Melina Lyraki, Giwta Piggou, Ioannis Pistolas, Viola Saveta, Mariza Tsalla, Spiros Tsatouchas. I would also like to thank Mavra Drakouli for supporting me in so many ways, she really made everything much simpler for me.

I am grateful to all the colleagues that helped me on important academic decisions about my future, Dimitris Achlioptas, Panagiotis Achlioptas, Aris Anagnostopoulos, Foteini Baldimtsi, Markos Katsoulakis, Kostas Kollias, Charalampos Papamanthou, Alexandra Papoutsaki, Petros Perselis, Matteo Riondato, Manolis Tsangaris, Charalampos Tsourakakis. I am also grateful to Professor Asish Goel (Stanford University) and Professors Roberto Tamassia and Eli Upfal (Brown University) for believing in me.

Last but not least I am grateful to my family, Manos, Asteropi, Sotiria, Freek, Polly, Kimon, for the encouragement and the support in every single aspect of my life.

Heraklion-Crete,
June 2012

Evgenios M. Kornaropoulos

Contents

List of Figures

Chapter 1

Introduction

In information technology, big data is a term used to describe data sets so large and complex that they become awkward to work and analyze. Difficulties include capture, storage, search, sharing, analysis, and visualization. In this work we are focusing in the latter part, specifically we study the visualization of graphs. Graphs are mathematical structures used to model pairwise relations between objects. In this thesis we propose a new framework for visualizing graphs. We extend the so-called Dominance Drawing technique, by relaxing some constraints in order to include the much broader family of directed acyclic graphs. The result is the formulation of an interesting computational problem called Weak Dominance Placement. Using a weak dominance placement of the vertices, we propose a new model for visualizing directed acyclic graphs called Overloaded Orthogonal. Furthermore, we propose an even more general framework called DAGView that is proposed in order to include graphs with cycles as well as undirected graphs. Finally we present our Java implementation of the DAGView framework for visualizing graphs.

Dominance Drawing is a well-known technique for visualizing planar *st*-graphs [4]. The resulting drawings present useful features such as: small number of bends, small area, detection and display of symmetries, and geometric characterization of the transitive closure by means of the dominance relation between the points associated with the vertices. An example of a dominance drawing is depicted in Figure 1.1.

Therefore, a natural direction is to extend the dominance drawing in order to be applicable not only to planar *st*-graphs, but to the general family of directed acyclic graphs as well. A direct extension of the Dominance Drawing method is not possible, and the reason lies in the notion of the so-called dimension of a graph. Thus, we relax some constraints, and form a new general approach. But in order to get useful mathematical insights of the problem, we need to use elements from the field of Order Theory.

Order Theory studies various types of binary relations between elements. In particular, we are interested in partial orders and linear extensions. An extensive study of this area can be found in [47]. A partially ordered set (or poset) is a pair (A, P) where A is a set of elements and P is a reflexive, antisymmetric, and transitive binary relation on the elements of A. Clearly, not every pair of elements

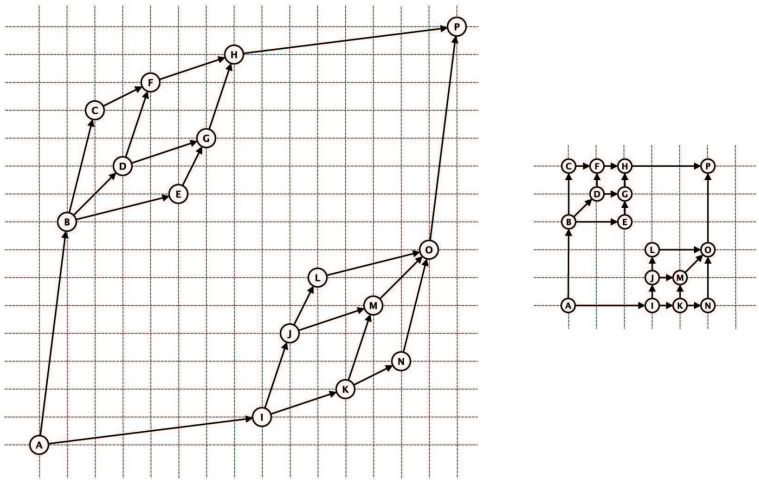


Figure 1.1: Left: A drawing made according to the Dominance-Straight-Line algorithm [4]. Right: The coordinate assignment in the 'Compaction' phase of the same algorithm.

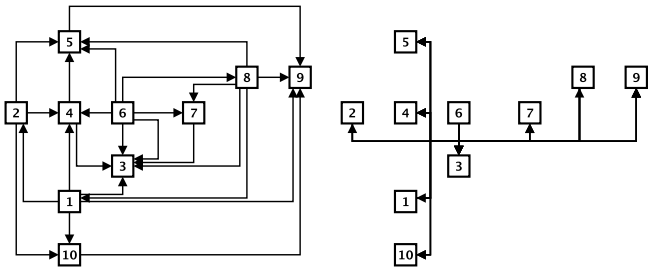


Figure 1.2: Left: A Kandinsky drawing of a high-degree directed acyclic graph. Right: The edges of the same drawing are routed according to the orthogonal bus-style edge routing. Even though this routing style was formed to enhance readability, there are several ambiguities due to the overlapping of this form.

in a partial order needs to be related.

We formulate a computationally interesting problem, namely *Weak Dominance Placement*, that lies in the very core of our generalization. Using the mathematical insights from partial orders, we are able to prove that the Weak Dominance placement problem is NP-complete. In the Weak Dominance problem, planar *st*-graphs is a subclass of directed acyclic graphs which we can optimally solve in linear time. Moreover, we propose three heuristic algorithms in order to obtain locally optimal solutions.

Using the Weak Dominance Placement problem we introduce a new graph drawing model called *Overloaded Orthogonal*. It can be confirmed in Figure 1.3 that a straight line edge routing wouldn't highlight the benefits of the weak dominance placement. Instead, the edges are routed in an orthogonal manner using the underlying grid.

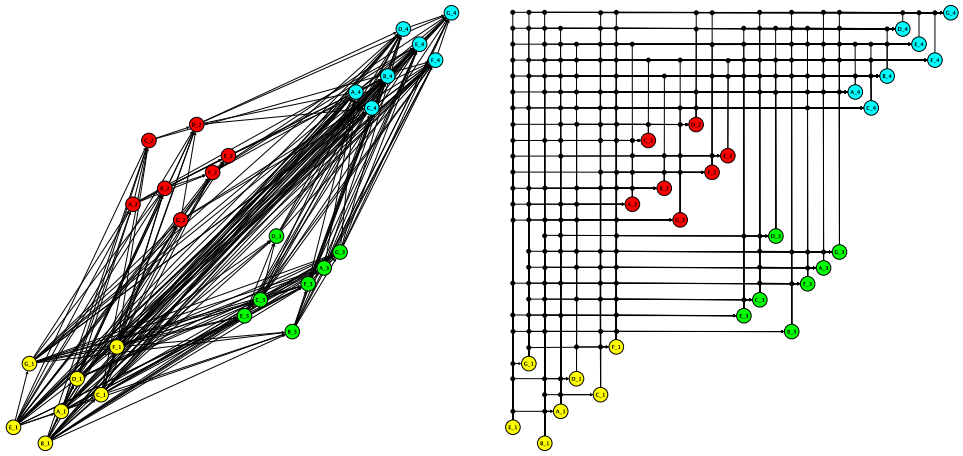


Figure 1.3: Left: Vertices are placed according to the weak dominance, but edges are routed in straight-line manner. Right: The visualization according to the Overloaded Orthogonal model.

The Orthogonal Drawing model is useful in many applications due to the high clarity of its drawings. A thorough treatment of this subject can be found in [3, 15]. One emerging limitation when we are constructing orthogonal grid embeddings is the presence of vertices with degree higher than four. In the traditional orthogonal drawing model if a vertex has degree higher than four, we cannot avoid overlapping the edges. A great number of works are trying to avoid this situation and several techniques have been proposed to overcome such an outcome [6, 21, 37, 38]. A mainstream technique in this direction, know as Kandinksy [21], is presented in Figure 1.2.

Recent works such as the Confluent Model [12, 18], indicate the overlapping of edges as a promising approach. On the other hand tools such as [51] try, not always successfully, to simplify the resulting visualization by overloading the edges and creating orthogonal bus-style visualization. This layout algorithm routes the edges of a diagram using only vertical and horizontal line segments while the position of each vertex is not altered. The algorithm yields long line edge segments (the so-called backbones). Ideally, all but the first and the last segments of all edge paths are drawn on top of each other, with short connections branching off to the nodes. This characteristic is claimed to combine the confusing mass of edges of a drawing in a tree-like structure. An example is depicted in Figure 1.2.

The Overloaded Orthogonal model handles the previously mentioned ambiguities efficiently. Edges have one vertical and one horizontal segment and flow from bottom-to-top and from left-to-right. In order to simplify these drawings we use an overloading technique to route the edges. As a result, we can check if any two vertices are connected by inspecting only a single point of the grid i.e., in $O(1)$ time. All algorithms are simple and easy to implement and can be applied to directed acyclic graphs of arbitrary degree, planar, and non-planar graphs.

Finally, we propose the *DAGView* framework which handles not only the di-

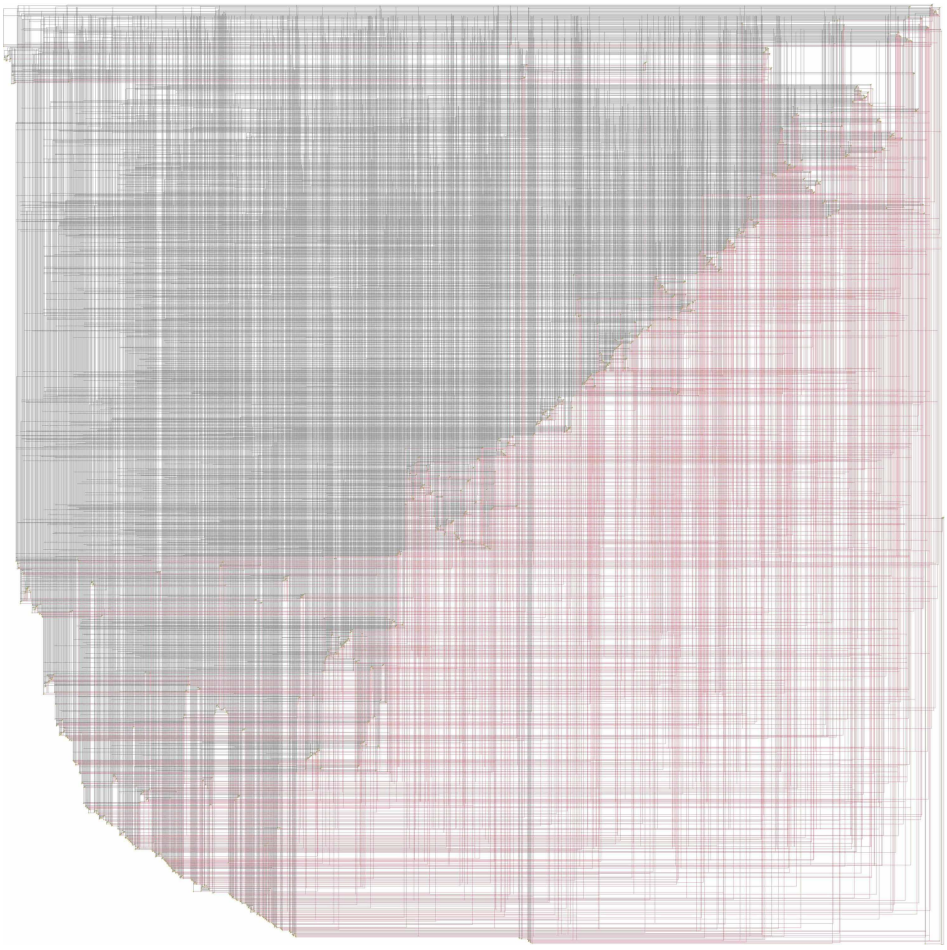


Figure 1.4: This DAGView visualization depicts a directed graph with cycles that was created from Wikipedia articles. The 1.515 nodes represent Wikipedia articles about composers and the 10.528 edges represent links between these articles.

rected acyclic graphs but also graphs with cycles and undirected graphs. We believe that this framework will be well accepted in user studies too. DAGView combines the readability and scalability of a matrix based approach, with the intuitiveness of a node-link approach in spotting a node within the layout and in following an edge to find its destination. With our proposed framework users can easily navigate through large graphs as the one produced by our Java implementation in Figure 1.4.

Chapter 2

Weak Dominance Placement

2.1 Introduction

The dominance drawing technique is an important approach for visualizing planar *st*-graphs. This method presents some very practical features such as: the planarity preservation of the given planar embedding, the geometric characterization between the vertices, an easy to implement algorithm, the characteristic that edges flow in the same direction (e.g. from bottom-to-top), and the display of symmetries and isomorphic components of the graph.

Before describing the technique, let us give some fundamental definitions. A *planar graph* is a graph that can be embedded in the plane, i.e. it can be drawn in such a way that no edges cross each other. Such a drawing is called plane graph or *planar embedding of the graph*. It is a well studied family of graphs that has attracted a lot of attention not only in the graph drawing community, but also in the broader graph theoretical community [8, 9, 30]. Graphs that have exactly one source and one sink are called *st-graphs*. An edge (u, v) of a directed graph is *transitive* if there is a directed path from u to v that does not contain the edge (u, v). A *reduced directed graph* is a directed graph with no transitive edges.

A *dominance drawing* of a directed acyclic graph G is a drawing Γ of G, such that for any two vertices u and v, there is a directed path from u to v in G, if and only if $X(u) \leq X(v)$ and $Y(u) \leq Y(v)$ in Γ. A strong characteristic of this technique from a topological perspective, is the following: for a given vertex v, we have to visually check only the lower-left quadrant of v in order to identify the vertices that reach v. Similarly, if we want to visually explore the vertices that can be reached from vertex v, we can focus only to the upper-right quadrant. Therefore, this characteristic can be used to visually explore the graph and answer reachability queries.

The algorithm that constructs a dominance drawing consists of three phases. The first Phase is the 'Preprocessing step' that sets up a linked data structure according to a planar embedding that is given as an input. If not already given, a planar embedding of G can be constructed in linear time using variations of well known planarity testing algorithms [26]. We also assume that G is embedded in the plane with s and t on the external face. The second phase, 'Preliminary Layout',

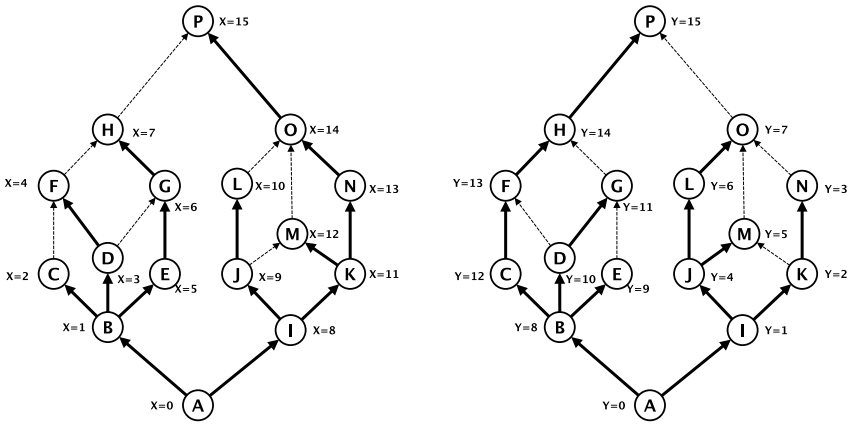


Figure 2.1: Left: The X-coordinate assignment of a planar st-graph according to the 'Preliminary Layout' phase. Successors are scanned from left-to-right. Right: The Y-coordinate assignment of the 'Preliminary Layout' phase. Successors are scanned from right-to-left.

assigns to each vertex v a distinct $X-$ and $Y-$ coordinate in the range $[0, n-1]$. In this phase the algorithm performs two topological sortings of the vertices of G, which scan the successors of each vertex from left-to-right (e.g. clockwise) and from right-to-left (e.g. counterclockwise), respectively. Figure 2.1 presents an example of the coordinate assignment of the Preliminary Layout phase. The third phase, 'Compaction', adjusts the position of the vertices to reduce the area of the drawing. The algorithm scans the vertices according to the order given by the preliminary $X-$ and $Y-$ coordinates. Figure 1.1 depicts the dominance drawing of a graph according to the preliminary layout coordinate assignment, as well as the dominance drawing according to the compaction phase.

In [4] the authors prove that any straight line dominance drawing Γ of a reduced planar st-graph G is planar. In other words, the dominance drawing technique preserves the planarity of the embedding. In addition, the Dominance-Straight-Line algorithm produces minimum area straight-line dominance grid drawing of G.

Theorem 2.1.[4] Let G be a reduced planar st-graph with n vertices. Algorithm Dominance-Straight-Line constructs in $O(n)$ time a planar straight-line dominance grid drawing Γ of G with $O(n^2)$.

An asymptotically optimal algorithm to obtain a two-dimensional dominance drawing of a bipartite graph whenever such a drawing exists was presented in [16].

2.2 Preliminaries: Linear Extension Diameter

A *linear order* (or total order) L is a binary relation on some set A, with the properties of (a) antisymmetry, (b) transitivity and also (c) the property that every

pairing of elements of A must be related by L. A *partially ordered set* (or poset) is a pair (A, P) where A is a set of elements and P is a reflexive, antisymmetric, and transitive binary relation on the elements of A. We call A the *ground set* while P is a *partial order* on A. Let $x, y \in A$ be elements such that $x \leq y$ or $y \leq x$, then x, y are *comparable*, otherwise they are *incomparable*. The *incomparability graph* of P is an undirected graph $I(P)$ with A as its set of nodes and with edges connecting the pairs of incomparable elements (such a graph is sometimes called a comparability graph in bibliography).

A total order L is a *linear extension* of a partial order P if, whenever $x \leq y$ in P it also holds that $x \leq_L y$ in L. To put it briefly, linear extensions are permutations that do not violate the given binary relations between pairs of P. The distance between two linear extensions L_i, L_j of P, denoted by $dist(L_i, L_j)$, is the number of pairs of elements that are in opposite order between the two linear extensions. Given a poset $(A, \leq)$, a pair L_1, L_2 of linear extensions is called a *diametral pair* if it maximizes the distance among all pairs of linear extensions of P. The maximal distance is called *linear extension diameter of* P and is denoted by $led(P)$.

A set R of linear extensions of a partial order P is a *realizer* of P provided that for all $x, y \in A$, $x \leq_P y$ if and only if x is below y in every member of R. The *dimension*, $dim(P)$, of a partial order P is the smallest realizer of P [13]. It has been proved that every partial order P is the intersection of a family of linear orders.

The computational complexity of whether a partial order has dimension at most k was an open question when Garey and Johnson wrote their book on NP-completeness [22]. Formally, the problem was presented in [22] as:

PARTIAL ORDER DIMENSION
INSTANCE: Directed acyclic graph $G = (V, E)$ that is transitive, i.e., whenever $(u, v) \in E$ an $(v, w) \in E$, then $(u, w) \in E$, and a positive integer $K \leq |V^2|$.
QUESTION: Does there exist a collection of $k \leq K$ linear orderings of V such that $(u, v \in E)$ if and only if u is less than v in each of the orderings?

As proved later in [50] by Yannakakis, it is NP-complete to determine if the dimension of a partial order is at most 3, and consequently the same holds also for any fixed $k \geq 3$. In [13] the authors prove that P has dimension 2 if and only if the incomparability graph $I(P)$ is transitively orientable. In other words, if the edges of $I(P)$ can be oriented so that the resulting directed graph is transitive, then $dim(P) = 2$. This condition combined with a linear time algorithm for the recognition of transitively orientable graphs [33], gives an efficient algorithm to test if a partial order has dimension at most 2.

The *linear extension graph* $G(P)$ is the undirected graph that has a vertex for every different linear extension of P [20]. Two vertices L_i, L_j of $G(P)$ are connected by an (undirected) edge if the linear extensions differ only by a single adjacent transposition. An example is depicted in Figure 2.2. The problem of counting the number of vertices of a linear extension graph belongs to the complexity class #P, due to the fact that counting linear extensions also belongs to #P [11]. The

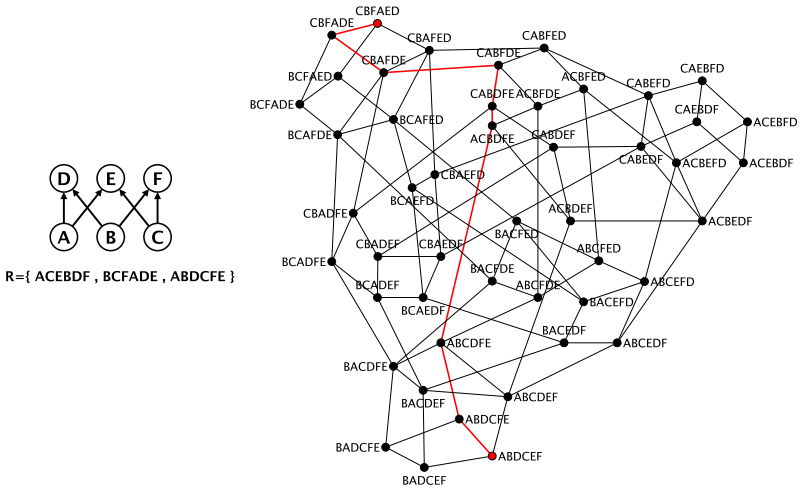


Figure 2.2: Left: This partial order P, which can also be derived from the crown C_3 graph, has 6 elements,6 binary relation, and 9 incomparable pairs. The dimension is $dim(P) = 3$, and a realizer is R is shown. Right: The linear extension graph with 48 vertices and 96 edges, of the partial order derived from the crown C_3. A diametral pair of linear extensions is shown in red L_1,L_2. The distance is $dist(L_1, L_2) = 8$ out of $inc(C_3) = 9$ incomparable pairs.

distance between two linear extensions L_i, L_j of P, is equal to the shortest path distance between the corresponding vertices in $G(P)$ [34].

If P has dimension 2 there are two linear extensions L_i, L_j such that every pair a, b of elements that cannot be compared, appear as $a \leq_{L_i} b$ in linear extension L_i, and as $b \leq_{L_j} a$ in L_j. Due to the fact that every incomparable pair of the elements is in the opposite order between L_i and L_j, the distance $dist(L_i, L_j)$ is equal to the number of incomparable pairs. Since we have already considered every incomparable pair, the $dist(L_i, L_j)$ takes its maximum value. This implies that L_i, L_j is a diametral pair. In this case, the linear extension diameter is equal to the number of pairs of incomparable elements.

The search of a diametral pair of linear extensions is formulated as follows:

(LED) LINEAR EXTENSION DIAMETER
INSTANCE: A finite poset (A, P), and a natural number k.
QUESTION: Are there two linear extensions of P with distance at least k?

In their work Brightwell and Massow [10] proved that LINEAR EXTENSION GRAPH problem is NP-complete with a reduction from the BALANCED BIPARTITE INDEPENDENT SET problem. Therefore:

Theorem 2.2. [10] *LINEAR EXTENSION DIAMETER problem is NP-complete.*

In [19] there is a sentence that connects diametral pairs to an optimal drawing of P. Notice that there is an evident connection between partial orders from the

field of Order Theory, and directed acyclic graph from the field of Graph Theory. Specifically, the vertex set of a directed acyclic graph ordered by the reachability relation, is a partially ordered set. On the other hand, every partially ordered set is a directed acyclic graph. In this work we will use notions from the area of partially ordered sets to prove results for directed acyclic graphs.

2.3 Complexity of the Problem

The notion of dominance drawing *dimension* of a directed acyclic graph G (denoted as $dim(G)$) is defined as the value of the smallest k for which a k-dimensional dominance drawing of G can be obtained [16]. The family of planar st-graphs is a subclass of directed acyclic graphs for which a (2-dimensional) dominance drawing can be efficiently computed in linear time [3, 4]. This is possible because planar st-graphs have dimension 2.

We define as $\Gamma = (t_1, t_2)$ the dominance drawing that is constructed using the topological sorting t_1 for the assignment of the first coordinate (equiv. X), and the topological sorting t_2 for the assignment of the second coordinate (equiv. Y). If a graph G has $dim(G) > 2$ then there is at least one pair of vertices $u, v \in V$ such that $t_1(u) \leq t_1(v)$ and $t_2(u) \leq t_2(v)$ in Γ, while neither u can reach v, nor v can reach u. Thus, a relaxed condition needs to be introduced in order to obtain a dominance drawing for any directed acyclic graph (dag).

Consider two vertices u and v in a dag G such that there is no path from u to v (or else vertex v is unreachable from u) but $t_1(u) \leq t_1(v)$ and $t_2(u) \leq t_2(v)$ in Γ. Then the implied path (u, v) in Γ will be called a *falsely implied path* (or simply fip) and will be included in the set of fips implied by drawing Γ. The number of falsely implied paths of drawing Γ will be denoted by $fip(\Gamma)$ or $fip(t_1, t_2)$. The existence of falsely implied paths is the trade off in drawing graphs with $dim(G) > 2$ in the two dimensional plane using the concept of dominance.

In the dominance drawing framework:

Dominance condition: *A dominance relation between the coordinates of vertices in* Γ *indicates the existence of a directed path in* G*, while if a path exists in* G *then there is a dominance relation between vertex coordinates in* Γ*.*

We introduce the concept of *weak dominance placement* by relaxing the necessity of the existence of a path [32], that is:

Weak Dominance condition: *A dominance relation between the coordinates of vertices in* Γ *does not necessarily indicate the existence of a directed path in* G*, while if a path exists in* G *then there is a dominance relation between vertex coordinates in* Γ*.*

Formally, a weak dominance placement Γ of a directed acyclic graph G is constructed, such that for any two vertices u and v if there is a directed path from u to v in G then $t_1(u) \leq t_1(v)$ and $t_2(u) \leq t_2(v)$ in Γ.

The challenge in this problem is to minimize the number of fips, $fip(\Gamma)$. A topological sorting of a dag $G = (V, E)$ is defined as a bijective function t that

maps each vertex of G to a number in $[1, |V|]$, with the additional property that for every edge $(u, v) \in E$, $t(u)$ is smaller than $t(v)$. The *intersection* of two topological sortings t_1, t_2 of vertices in V is the set $I = \{(u, v)|t_1(u) < t_1(v), t_2(u) < t_2(v)\}$. We can formulate the corresponding decision problem as follows:

(WDP) WEAK DOMINANCE PLACEMENT
INSTANCE: A directed acyclic graph $G = (V, E)$ and a positive integer C such that $|\mathrm{E}| \leq C \leq \frac{|V|(|V|-1)}{2}$.
QUESTION: Does there exist a collection of two topological sortings t_1, t_2 of V such that their intersection has cardinality C or less?

A simple example for a crown graph C_3 (can also be found as S_3^0 in bibliography) is shown in Figure 2.3. This graph cannot be drawn so that its drawing Γ has $fip(\Gamma) = 0$, due to the fact that $dim(C_3) = 3$. As depicted in Figure 2.3, there is at least one path that is implied by the coordinate assignment of vertices and does not belong to C_3. From the above definition of the Weak Dominance Placement problem it can be easily derived that if a directed acyclic graph G admits a dominance drawing, then G admits a weak dominance placement Γ such that $fip(\Gamma) = 0$.

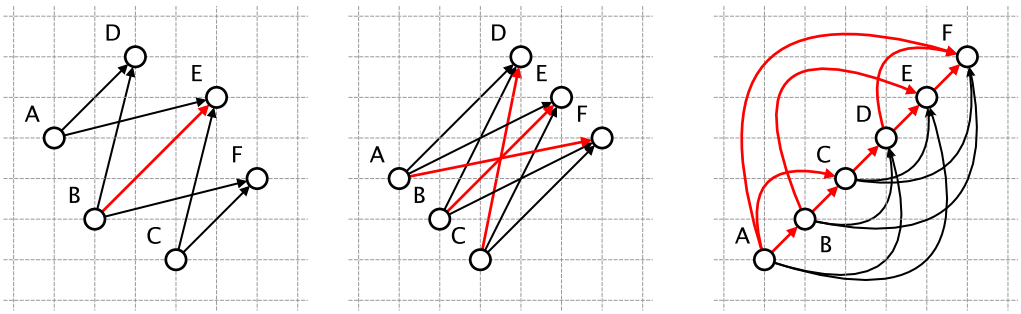


Figure 2.3: Three different weak dominance placements of crown graph C_3. Falsely implied paths appear with red. In the left drawing there is an optimal weak dominance placement with one falsely implied path, in the middle there is a weak dominance placement with three falsely implied paths, while in the right there is a weak dominance placement with nine falsely implies paths.

An *incomparable pair* of vertices u, v has the property that neither u is reachable from v, nor v is reachable from u. We will denote as $inc(G)$ the number of incomparable pairs of a dag G. The transitive closure of G is the graph $G^* = (V, E^*)$, where E^*=$\{(\ i\ ,\ j\ )$: if there is a path from vertex i to vertex j in $G\}$.

Fact 1. *The number of incomparable pairs of a dag G is:*

$$inc(G) = \frac{|V|(|V|-1)}{2} - |E^*|.$$

Clearly, according to the definition of fips, we have the following:

Fact 2. *Let* Γ *be a weak dominance placement of dag* G*. Then:*

$$fip(\Gamma) \leq inc(G).$$

The equality in the above bound is achieved for weak dominance placements where all vertices are placed on the diagonal, i.e., $\forall u \in V, t_1(u) = t_2(u)$. A general upper bound is given in the following:

Lemma 2.1. If G is a directed acyclic graph and Γ is a weak dominance placement of G, then the following inequality holds:

$$min_{\Gamma} fip(\Gamma) \leq inc(G) - (dim(G) - 2).$$

Proof. Let $T = \{t_1, t_2, ..., t_d\}$ be a set of topological sortings such that $d = dim(G)$. We choose two topological sortings t_i, t_j from T in order to construct a weak dominance placement Γ, such that Γ has the minimum number of fips comparing to any other pair of topological sortings in T. Since each topological sorting from $T - \{t_i, t_j\}$ decreases the number of fips in the intersection by at least one, drawing Γ will have at most $inc(G) - (d - 2)$ fips. Notice that although pair t_i, t_j is such that it yields the minimum number of fips out of the topological sortings in T, there could be another pair, not in T, that has an even lower number of fips. Thus, the inequality $min_{\Gamma} fip(\Gamma) \leq inc(G) - (dim(G) - 2)$ holds. □

Another upper bound for the minimum number of fips for directed acyclic graphs is:

Lemma 2.2. Let G be a directed acyclic graph and Γ be a weak dominance placement of G, then the following inequality holds:

$$min_{\Gamma} fip(\Gamma) \leq inc(G) - \lceil \frac{2inc(G)}{dim(G)} \rceil$$

Proof. Let $T = \{t_1, t_2, ..., t_d\}$ be a set of topological sortings such that $d = dim(G)$. Let us randomly choose two t_i, t_j from T in order to construct weak dominance placement Γ. The probability that an incomparable pair of vertices u, v is expressed as a fip in Γ is at least $(d-1)/\binom{d}{2}$. Thus, the expected number of fips is $\sum_{i=1}^{inc(G)} (d-1)/\binom{d}{2} = 2inc(G)/d$. Therefore in an optimal drawing $min_{\Gamma} fip(\Gamma)$ will be less than $inc(G) - \lceil \frac{2inc(G)}{dim(G)} \rceil$. □

Lemma 2.3. Let G be a directed acyclic graph and t_1, t_2 two topological sortings of G such that $fip(t_1, t_2) = 0$. Let also T be the set of all topological sortings of G. Then the following inequality holds:

$$fip(t_1, t_2) < fip(t_1, t_3) \ ,\forall t_3 \in T$$

Proof. Since $fip(t_1, t_2) = 0$ and the minimum value of $fip(t_1, t_3)$ is zero, we only have to prove that there is no topological sorting t_3 such that $fip(t_1, t_3) = 0$.

Assume for a contradiction that there is a topological sorting t_3, such that $t_3 \neq t_2$ and $fip(t_1, t_3) = 0$. Let us also assume that topological sortings t_2 and t_3 are identical up to a position k, where $0 \leq k < n$. So the vertex in position $k+1$ is the first vertex that t_2 and t_3 differ. Notice that if $k = 0$ then t_2 and t_3 differ in position $k + 1 = 1$.

The vertex in the $(k+1)$-th position of the topological sortings t_2 and t_3, will be denoted as $t_2[k+1]$ and $t_3[k+1]$ respectively. Then $t_2[k+1] = u$ and $t_3[k+1] = v$, where $u \neq v$. It its easy to prove that since t_2, t_3 are identical up to position k and they are both topological sortings, the vertices u and v are incomparable. There are two cases concerning the order of u and v in the topological sorting t_1.

Case $u <_{t_1} v$ *:* Since t_2 and t_3 are identical up to position k and $t_2[k+1] = u$, the vertex v will be placed after u in t_2. In this case $u <_{t_1} v$ and $u <_{t_2} v$, which implies that since $u - v$ are incomparable we have $fip(t_1, t_2) \geq 1$. This contradicts the fact that $fip(t_1, t_2) = 0$.

Case $v <_{t_1} u$ *:* Since t_2 and t_3 are identical up to position k and $t_3[k+1] = v$, the vertex u will be placed after v in t_3. In this case $v <_{t_1} u$ and $v <_{t_3} u$, which implies that since $u - v$ are incomparable we have $fip(t_1, t_3) \geq 1$. This contradicts our assumption that there is a topological sorting t_3 such that $fip(t_1, t_3) = 0$. □

A partially ordered set can be viewed as a transitive directed acyclic graph with a set of nodes A, and an edge $(u, v) \in E$ for each pairwise relation between $u <_P v$ in P. In these terms, a linear order is a complete dag. Linear extensions can be interpreted as topological sortings in graph theory. Therefore we can prove the following theorem:

Theorem 2.3. The LINEAR EXTENSION DIAMETER problem reduces to the WEAK DOMINANCE PLACEMENT problem.

Proof. In order to reduce the Linear Extension Diameter (LED) problem to the Weak Dominance Placement (WDP) problem we construct a dag G from a given a partial order P on the set of elements A, as described above. For every element u of A we have a vertex u in G, and for every relation $u <_P v$ between two elements, we have a directed edge (u, v).

First we prove that an optimal solution to LED is an optimal solution to WDP. Let L_i, L_j be two linear extensions of the given partial order P, that form a diametral pair of P, which implies that their distance is $dist(L_i, L_j) = k$. Thus the linear extension diameter of this partial order is $led(P) = k$. This means that out of $\binom{|A|}{2}$ possible pairs of elements of linear extension L_i, k of them are in the opposite order in L_j. Let us now choose these two linear extensions as the topological sortings for WDP and denote them as t_i, t_j. Then we will have $C = \binom{|A|}{2} - k$ pairs in the intersection, due to the fact that these C pairs appear in the same order on both topological sortings. Since k takes its maximum value for partial order P and $\binom{|A|}{2}$ is fixed, C is minimum. Thus, the minimum cardinality of the intersection set is C.

Next, we prove that an optimal solution to WDP is an optimal solution to LED. Let us assume that t_i and t_j are two topological sortings such that their intersection has minimum cardinality C. There are $\binom{|A|}{2}$ possible pairs of vertices that may appear in the intersection of t_i, t_j. Let k be the number of pairs that do not appear in the intersection of t_i, t_j. Then $C = \binom{|A|}{2} - k$. Since C is minimum, it implies that the value of k is maximum. Therefore, we will interpret t_i, t_j as linear extensions L_i, L_j in the LINEAR EXTENSION DIAMETER problem. Since we have k pairs that appear in the opposite order between t_i and t_j, we have $dist(L_i, L_j) = k$. Considering that k is the maximum value that any two topological sortings can differ in their intersection, it is also the maximum distance between any pair of linear extensions, therefore $led(P) = k$.

Concluding, the LINEAR EXTENSION DIAMETER problem has a solution if and only if the WEAK DOMINANCE PLACEMENT problem has a solution. □

Corollary 2.1. The WEAK DOMINANCE PLACEMENT problem is NP-complete.

2.4 Heuristic Algorithms

The problem of finding a pair of topological sortings t_1, t_2 of dag G such that the number of $fip(t_1, t_2)$ is minimum is NP-hard. In this section we propose three heuristic algorithms that seem to perform well in practice. In all of the following algorithms the topological sorting t_1 is given as an input and it remains fixed throughout the algorithm. Therefore, the goal is to construct a topological sorting t_2 such that the quantity $fip(t_1, t_2)$ is minimal. The complexity of each algorithm is discussed but we do not provide any performance guarantee.

2.4.1 Local Search

Local search algorithms make a sequence of decisions that optimize some local choice, though these local choice might not lead to the best overall solution. In this algorithm we start with the given fixed topological sorting t_1 as the initial choice of t_2. Then the algorithm tests if a local swap between two incomparable and consecutive in t_2 vertices, results in a decrease of the number of fips. If that is the case, the swap is made and the process is repeated until we reach a locally optimal solution. The swap between two incomparable and consecutive vertices in t_2 is performed only if the number of fips strictly decreases. We refer to the i-th element of t_2 as $t_2[i]$.

Algorithm Local-Search (G, t_1)

1. $t_2 \leftarrow t_1$
2. **while** there is a swap between incomparable vertices
3. $\quad t_2[i]$, $t_2[i+1]$ that decreases the $fip(t_1, t_2)$ **do**
4. $\quad\quad t_2 \leftarrow t_2.swap(i, i+1)$
5. **end**
6. **return** t_2

The maximum number of incomparable pairs is $\binom{n}{2}$ and occurs in a graph with isolated vertices. So the algorithm reaches a local optimum in a polynomial number of steps, since we require a decrease in every step and there can be $\binom{n}{2}$ improvement steps at most. A run of the algorithm is depicted in Figure 2.4.

In order to check if a potential swap of consecutive vertices decreases the number of fips, a data structure can be implemented. We will use an array $n \times n$ that contains a binary value at each cell. The value $'1'$ will appear in row u and column v in order to resemble the falsely implied path (u, v), and $'0'$ otherwise. This data structure will initially contain an ordered pair (u, v) for every incomparable pair $u - v$, such that $u <_{t_1} v$. Thus, the array initially contains $inc(G)$ cells of $'1'$ value, since we use the same topological sorting for t_1 and t_2. For every swap, a $'1'$ value is replaced with a $'0'$. By using an array we need $\Theta(n^2)$ for the initialization, $O(1)$ for the removal of an ordered pair, and $O(1)$ for checking if there is a falsely implied path between a pair of vertices. Therefore, the algorithm will run in $O(n^2)$ time.

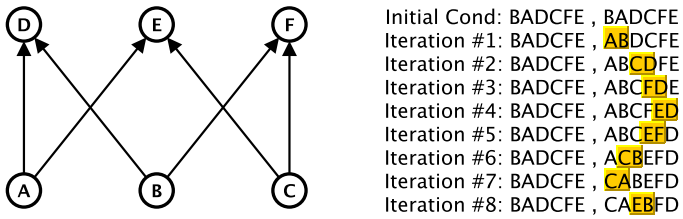


Figure 2.4: A run of the local search algorithm on crown graph C_3 with t_1 = BADCFE. The resulting pair of topological sortings t_1 = BADCFE and t_2 = CAEBFD has one falsely implied path (namely A-F).

2.4.2 A greedy approach: Maximum Rank

In the algorithm Max-Rank we follow a greedy approach in order to construct t_2 given a topological sorting t_1 as an input. The solution here is formed step by step, by making a decision that is locally optimal.

One mainstream algorithm for constructing a topological sorting [29] runs the following procedure: choose any source u, put u in the under-construction topological sorting, remove u from the graph (as well as its incident edges) and update

the set of sources. Then repeat this procedure until every vertex of the directed acyclic graph is in the topological sorting. It is easy to prove that when there are more than one sources during an iteration of the above procedure, these vertices are incomparable per two.

In our approach, the way we choose the next source is not random. Our algorithm chooses the source u with the highest position in t_1 (also called rank of the element), in order to avoid the falsely implied paths between u and the sources at the current iteration.

Let S_G be the set of sources of directed acyclic graph G. We will refer to the vertex of set S_G with the maximum rank in the topological sorting t_1, as $maxRank_{t_1}(S_G)$. A run of the algorithm is depicted in Figure 2.5.

Algorithm Maximum-Rank (G , t_1)
1. Initialize S_G
2. $t_2 \leftarrow NIL$
3. **for** $i \leftarrow 1$ **to** n
4. $u \leftarrow maxRank_{t_1}(S_G)$
5. $G \leftarrow G - \{u\}$
6. Update S_G
7. $t_2[i] \leftarrow u$
8. **end**
9. **return** t_2

The choice of our algorithm is locally optimal because among the $|S_G|$ vertices, we select the one that generates zero falsely implied paths between pairs of vertices from S_G. Specifically, when we put u in t_2, we avoid all the falsely implied paths that involve u and a source from S_G because u has the highest rank in t_1 among vertices of S_G, and the lowest rank in t_2 among vertices of S_G. Maximum-Rank removes overall n vertices and m edges from the graph while keeping a sorted data structure (for example max-heap) for the sources. In particular, the complexity of the insertion operation in a max-heap is $O(logn)$. Therefore the complexity of the algorithm Max-Rank is $O(nlogn + m)$.

We have already shown in Lemma 2.3 that if a dag G has two topological sortings t_1, t_3 such that $fip(t_1, t_3) = 0$, then there is no other topological sorting that can pair with t_1(resp. t_3) in order to result zero fips. In other words, there is only one topological sorting that 'pairs optimally' with t_1 in order to result zero fips.

In the following theorem we prove that if we are given a topological sorting t_1 for G such that $fip(t_1, t_3) = 0$, then Max-Rank outputs t_1's unique and optimal pairing.

Theorem 2.4. Let $G = (V, E)$ be a directed acyclic graph. Let t_1, t_3 be a pair of topological sortings of G such that $fip(t_1, t_3) = 0$. Let also t_2 be the resulting topological sorting of the algorithm Max-Rank(G, t_1) . Then the topological sorting t_3 is identical to t_2.

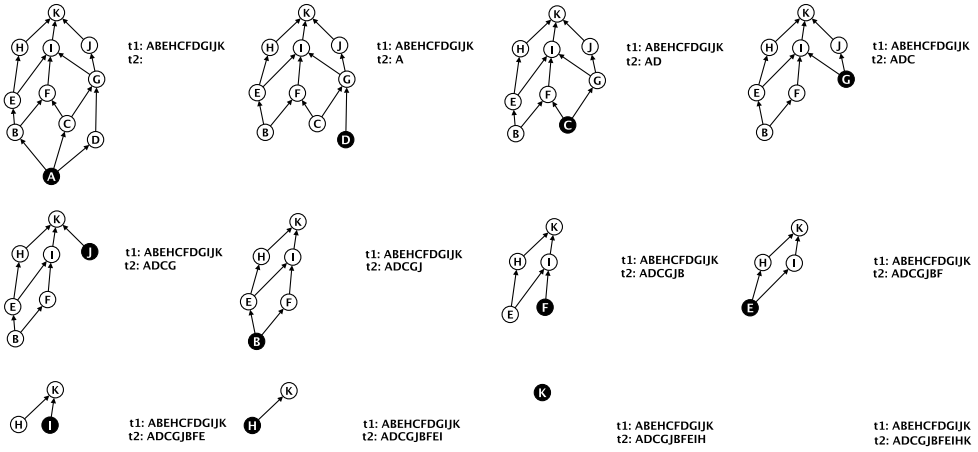


Figure 2.5: A run of the Maximum-Rank Algorithm on a planar *st*-graph with $t_1 =$ ABEHCFDGIJK. The resulting pair of topological sortings $t_1 =$ ABEHCFDGIJK and $t_2 =$ ADCGJBFEIHK has zero falsely implied paths.

Proof. The vertex in the i-th position of the topological sortings t_2 and t_3, will be denoted as $t_2[i]$ and $t_3[i]$ respectively.

Let us assume that the topological sortings t_2 and t_3 are identical up to position k, where $0 \leq k < n$. Furthermore, assume for a contradiction that the algorithm Max-Rank choose a different vertex at the $(k+1)$-th iteration from the vertex in $t_3[k+1]$. Then $t_2[k+1] = u$ and $t_3[k+1] = v$, where $u \neq v$. So we have to prove that the topological sortings remain identical in the position $k+1$ too.

A fact that we will use later in this proof is that all the predecessors of u (resp. v) have rank in the range of $[1, k]$ in t_2, otherwise it contradicts the fact that t_2 is a topological sorting. But since t_2 and t_3 are identical up to position k, all the predecessors of u (resp. v) have rank in the range of $[1, k]$ in t_3 too. Therefore, not only u but also v is a source in the $(k+1)$-th iteration of algorithm Max-Rank. It is easy to prove that if v and u are both sources in a given iteration, they are incomparable.

Furthermore, because Max-Rank chose u over v at the $(k+1)$-th iteration, the relation $v <_{t_1} u$ holds. It is also implied that $v <_{t_3} u$, since $t_3[k+1] \neq u$ and u does not appear in the construction of t_3 up to the k-th position.

By combining the relations $v <_{t_1} u$ and $v <_{t_3} u$ with the fact that v and u are incomparable, we can conclude that $fip(t_1, t_3) \geq 1$. But this contradicts the fact that $fip(t_1, t_3) = 0$. So our assumption that the algorithm Max-Rank choose a different vertex at the $(k+1)$-th iteration from the vertex in $t_3[k+1]$, does not hold. □

The algorithm Dominance-Straight-Line [3] constructs a topological sorting X and a topological sorting Y for a reduced planar *st*-graph. These topological sortings are interpreted as coordinates in order to place the vertices in an integer grid. In this drawing the strict dominance condition holds which implies that $fip(X, Y) = 0$. An interesting corollary from the Theorem 2.2, is that if we use the

topological sorting X (resp. Y) as the fixed topological sorting t_1 in the algorithm Max-Rank, then the output is Y(resp. X). This corollary indicates the connection of algorithm Max-Rank with the construction of a topological sorting according to an embedding, that is used in Dominance-Straight-Line algorithm.

Corollary 2.2. Let $G = (V, E)$ be a reduced directed acyclic planar *st*-graph. Let X be the topological sorting constructed by the procedure LabelX of algorithm Dominance-Straight-Line(G). Let also t_2 be the resulting topological sorting of the algorithm Max-Rank(G, X) . Then the topological sorting Y constructed by the procedure LabelY of Dominance-Straight-Line(G), is identical to t_2.

Corollary 2.3. Let $G = (V, E)$ be a reduced directed acyclic planar st-graph. Let Y be the topological sorting constructed by the procedure LabelY of algorithm Dominance-Straight-Line(G). Let also t_2 be the resulting topological sorting of the algorithm Max-Rank(G, Y) . Then the topological sorting X constructed by the procedure LabelX of Dominance-Straight-Line(G), is identical to t_2.

2.4.3 An approach using a Path-cover

In this approach the problem of constructing t_2 is broken down into simpler subproblems. The algorithm takes some decision in order to solve a subproblem, and then combines the solutions of the subproblem in order to form an overall feasible answer. Our approach is similar to a Dynamic Programming approach but the optimal substructure property does not hold. So the resulting t_2 is not necessarily optimal.

First compute in polynomial time a set of k paths covering all elements of the dag G. One option is to use a Breadth First Search algorithm, or a Depth-First-Search algorithm. An even better approach is to minimize the number of paths as described in [36], because the term k appears in the time complexity of our algorithm. The paths are not necessarily vertex-disjoint. We refer to these k paths as $P_1, ..., P_k$; each path P_i is a linear order of a subset of the elements, and we write P_i as $u_{i1}, u_{i2}, ..., u_{is_i}$. Let also $B = (b_1, ..., b_k)$ be an integer vector satisfying $0 \leq b_i \leq s_i$ for every i. Vector B is a border that has a member b_i from the corresponding path P_i. Since we have a fixed topological sorting t_1 there are exactly $inc(G)$ ordered pairs that may appear as fips in the final solution, we call them *potential fips*.

Following the terminology of dynamic programming, the problem will be divided into stages and each stage will be divided into states. There are n stages (one for each position of t_2) and k states (one for each path). We will refer to $t_2(j, i)$ as our solution at the j-th stage of the i-th state. Formally, $t_2(j, i)$ is a linear order of j vertices and the j-th vertex belongs to the path P_i. In order to solve the subproblem $t_2(j, i)$ we have to choose a subproblem $t_2(j-1, i')$ where $1 \leq i' \leq k$ of the $(j-1)$-th stage, such that the linear order $t_2(j-1, i').addLast(b_i)$ has the least number of potential fips. If the vertex b_i of the current border B is reachable

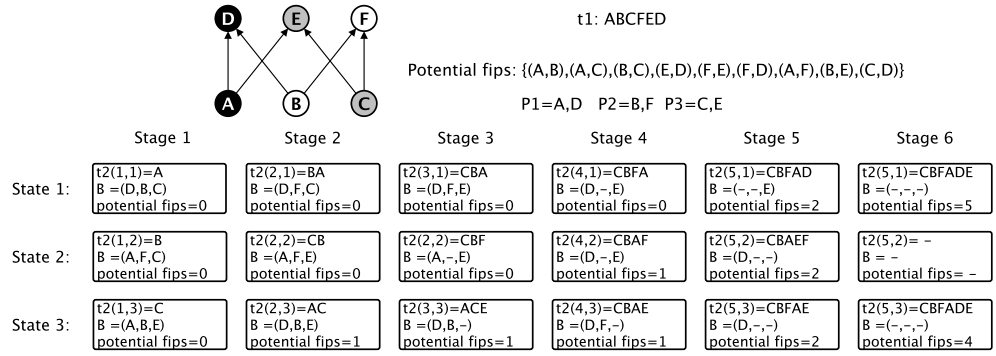


Figure 2.6: A run of the Path-Cover Algorithm on crown C_3 with fixed t_1. The resulting pair of topological sortings t_1 and t_2 has four falsely implied paths. It is not optimal since the topological sorting CBADEF, combined with t_1, results in three fips.

from another vertex of the border, then b_i cannot be considered as a valid choice for solving $t_2(j,i)$. In other words, $\forall b_q \in B$ we have that $b_q \nrightarrow b_i$. If the vertex b_i of the current border B is already in $t_2(j-1,i')$, we will not consider $t_2(j-1,i')$ either. After deciding which subproblem $t_2(j-1,i')$ to augment for solving $t_2(j,i)$, we update the corresponding border of $t_2(j-1,i')$ by replacing the i-th element of B, with the next vertex in P_i that does not appear in the construction. We follow a bottom up approach. A run of the algorithm is depicted in Figure 2.6.

Algorithm Subproblems (G , t_1)

1. $P \leftarrow pathsegmentation$
3. **for** $j \leftarrow 1$ **to** n
3. **for** $i \leftarrow 1$ **to** k
4. $t_2(j,i) \leftarrow$ Find a subproblem i', where $1 \leq i' \leq k$, such that $t_2(j-1,i').addLast(b_i)$ has the minimum number of potential fips, $b_i \notin t_2(j-1,i')$, and $\forall b_q \in B$ we have $b_q \nrightarrow b_i$.
5. $B(j,i) \leftarrow$ Duplicate $B(j-1,i')$ and augment the element in the i-th position according to P_i
6. **end**
7. **end**

Every subproblem can be solved in $O(nk)$, since we have k different states to check and we need $O(n)$ time to compute the potential fips. There are $n \times k$ subproblems, the overall time of the algorithm is $O(n^2k^2)$.

Chapter 3

Overloaded Orthogonal Model

3.1 An Introduction to Orthogonal Drawings

Orthogonal drawings are useful in many applications due to the high clarity of the model. A thorough treatment of this subject can be found in [3, 15]. An *orthogonal drawing* maps each edge into a chain of horizontal and vertical line segments. Formally, an *orthogonal grid embedding* Γ of a graph $G = (V, E)$ is a mapping into the plane, which maps vertices $v \in V$ to integer grid points $\Gamma(v)$ and edges in $(v, w) \in E$ to non-overlapping paths in the grid such that the images of their endpoints $\Gamma(v)$ and $\Gamma(w)$ are connected. A grid embedding is simple if its number of bends is zero. A simple embedding induces a partition of the edge set E into a horizontal set E_h and a vertical set E_v .

There is a lot of work on orthogonal drawings, but there are two main structural graph characteristics that are of great interest to the research community. The first is whether the graph is planar. In this case the bounds on the area and the number of bends greatly differ from the general case. The second is whether the graph has vertices with degree higher than four (simply referred as high degree graphs). There are several approaches to handle the vertices of a high degree graph in the orthogonal framework.

In planar orthogonal drawings the minimization of bends is an important aesthetic. The minimization of bends in planar orthogonal drawings, given a planar embedding, can be solved in quadratic time as proposed in the landmark paper [45]. In general, the problem of constructing an orthogonal drawing while minimizing several aesthetic criteria such as area, bends, maximum edge length, and total edge length is an NP-hard problem [3]. Therefore most algorithms employ heuristics that try to layout the graph in a manner which is good for some set of aesthetics. Experimental studies have been conducted where various proposed algorithms were tested on their performance on area, bends, crossings, edge length, and time [49]. Various algorithms have been introduced to produce orthogonal drawings of planar graphs [3, 6, 46]. For biconnected planar graphs, Bertolazzi et al. presented [5] a branch and bound algorithm that computes an orthogonal representation with the minimum number of bends. Figure 3.1 presents an orthogonal drawing of a planar graph. For drawings of non-planar graphs, the required area

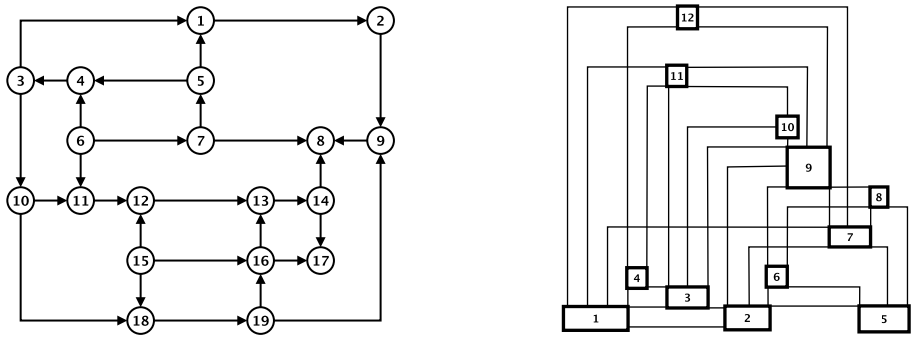


Figure 3.1: Left: An orthogonal representation with rectangular faces of a directed acyclic planar graph with 19 nodes and 28 edges. Right: A box-orthogonal illustration of a high degree graph with 12 nodes and 20 edges.

can be as little as $0.76n^2$ [38], the total number of bends is no more than $2n + 2$, and each edge has at most two bends [6, 38].

One emerging limitation when we are constructing orthogonal grid embedding of graphs is the presence of vertices with degree higher than four. In the traditional orthogonal drawing model if a vertex has degree higher than four, we cannot avoid overlapping the edges. In order to avoid this situation, several techniques have been proposed. One technique is to expand each vertex of high degree into a chain of vertices of degree four [6]. Then the proposed techniques for graphs with degree less than four, can be applied to the augmented graph in order to construct an orthogonal drawing.

On the same track, Fößmeier and Kaufmann introduced a model called the Kandinsky model [21]. In such a drawing, there are two different types of grid lines. First, the grid-lines of a coarse grid are used to only place the nodes. Second, the grid-lines of a finer grid serve to allow edges to attach on each side of the node. Here, all vertices are given as identical $k \times k$ squares, with the size k determined appropriately. In other words, in this drawings nodes have uniform node dimensions. Another technique is to present vertices with degree higher than four as a rectangular box (not necessarily as a square) [37]. Each box has a top, right, bottom and left side. A side may have more than one connector, on which an edge that has integer coordinate can be attached. In this framework the box of each vertex grows in proportion with its degree. This framework is referred as the proportional-growth model. An illustrative example is presented in Figure 3.1. One advancement of the proportional growth model is described in [7].

More recent results on orthogonal drawings is a necessary and sufficient condition for a plane graph with maximum degree three to have an orthogonal drawing without bends [44]. Another interesting result is that an outerplanar graph G with maximum degree at most three has an orthogonal drawing with no bends if and only if G contains no triangles [35].

3.2 The Overloaded Orthogonal Model

In this model, we propose to place the vertices in the grid so that edges flow from bottom-to-top and from left-to-right [31]. Each vertex u is placed on a point in the grid with coordinates $X(u)$ and $Y(u)$. Dominance drawings achieve this vertex placement for *st*-planar graphs. A dominance drawing Γ of a graph $G = (V, E)$ has the following property: for any two vertices $u, v \in V$ there is a directed path from u to v in G, if and only if $X(u) \leq X(v)$ and $Y(u) \leq Y(v)$ in Γ. Since dominance drawings are not applicable to all directed acyclic graphs, we will use the more general weak dominance condition:

Weak Dominance Condition: Let $G = (V, E)$ be a directed acyclic graph. For any two vertices $u, v \in V$ if there is a directed path from u to v in G, then $X(u) \leq X(v)$ and $Y(u) \leq Y(v)$.

Notice that the conditions on X and Y cannot be simultaneously satisfied with equality since distinct vertices must be placed at distinct points. Thus if v is in the upper-right quadrant of u, then v is not necessarily reachable from u. As discussed in the previous chapter, a path that is implied by the vertex coordinates but does not exist in G is called a falsely implied path (or fip). Following the footsteps of the algorithm for dominance drawing for (reduced) planar *st*-graphs presented in [4], we formulate an algorithm for vertex placement that respects the weak dominance condition and is applicable to any dag.

The main algorithm for planar *st*-graphs described in [4] consists of three phases. In the first phase, called 'Preprocessing Phase', a linked data structure is constructed in order to efficiently calculate coordinates. During the second phase called 'Preliminary Layout' distinct X, Y coordinates are given to each vertex. In the third and final phase, a compaction procedure is applied to reduce the area of the drawing.

We will construct a similar data structure as in 'Preprocessing step', but for general directed acyclic graphs. Let W be a representation of a dag G such that the incoming edges for each vertex u appear consecutively around u. Representation W will be called a *representation in consecutive form*. The representation in consecutive form is a method to force a left-to-right order in the incoming as well as outgoing edges of every vertex of G. Without loss of generality we assume that there is only one source, s. If not, then we insert an artificial super-source s and connect it to all sources of G. The algorithm performs two topological sortings on the vertices of G. Successors of each vertex are scanned in clockwise order for the X coordinate assignment, and in counterclockwise order for the Y coordinate assignment. The order is imposed according to the representation in consecutive form that is given as an input. We will present the algorithm for clockwise scan, that computes the X-coordinate assignment.

Algorithm TOPOLOGICAL-SORTING(Adj(G))

```
1. for each vertex v ∈ V
2.         X[v]← ∞
3. X[s]← 0
4. time←1
5. VISIT-CLCK(s)
6. return X
```

Algorithm VISIT-CLCK(u)

```
1. for each vertex v ∈ Adj(u) such that (u, v)
                is the leftmost outgoing edge of u do
2.          if in-degree(v)=1
3.                  X[v]← time
4.                  time←time+1
5.                  remove edge e=(u, v)
6.                  VISIT-CLCK (v)
7.          else
8.                  remove edge e=(u, v)
```

Algorithm TOPOLOGICAL-SORTING scans the outgoing edges of a vertex u in clockwise order (leftmost outgoing edge) and visits a direct successor v only if v has in-degree one. Otherwise, it removes edge (u, v) from the list. Analogously, we formulate an algorithm for the Y-coordinate assignment that performs a counter-clockwise scan, by replacing VISIT-CLCK with VISIT-COCLCK. The difference between the two VISIT algorithms is Line 1, where instead of leftmost outgoing edge we now have rightmost outgoing edge. The two topological sortings are used by WDP algorithm for assigning X and Y coordinates to the vertices of G.

Algorithm (WDP)WEAK DOMINANCE PLACEMENT (W)

```
1. X coordinates ← TOPOLOGICAL SORTING(W) using VISIT-CLCK
2. Y coordinates ← TOPOLOGICAL SORTING(W) using VISIT-COCLCK
```

We denote the number of vertices in G by n, and the number of edges in G by m. Since both topological sorting algorithms run in linear time $O(n + m)$, algorithm WDP also runs in linear time $O(n + m)$.

In the rest of this section we will see how the Algorithm WDP creates a natural separation between pq-components. A pq-component $G_{pq} = (V', E')$ of G is a maximally induced subgraph of G with a single source p and a single sink q that contains at least two edges and that is connected with the rest of G only through vertex p and vertex q. Thus, vertex p is a dominator of every vertex $v \in V'$ and q is a post-dominator of every vertex $v \in V'$.

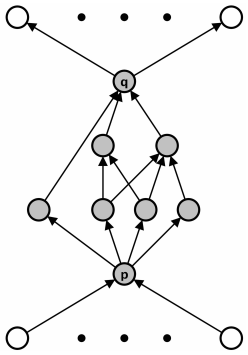


Figure 3.2: A pq-component

Lemma 3.1. Let a dag $G = (V, E)$ and a pq-component $G_{pq} = (V', E') \subseteq G$. Let also $X(), Y()$, be the coordinates assigned by the algorithm WDP. Then, $X(q) = X(p) + |V'| - 1$ and $Y(q) = Y(p) + |V'| - 1$.

Proof. As $orig(e)$ and $dest(e)$ we denote the vertex that is origin and destination of edge e, respectively. Vertex q is reachable from p, because q is a postdominator of p. The fact that vertex p is a dominator of vertices in V' does not permit edges (u, v) such that $u \in V - V'$ and $v \in V' - \{p\}$. Thus, for all the incoming edges e of $v \in V' - \{p\}$, we have $orig(e) \in G'$.

Furthermore, the fact that vertex q is a post-dominator of vertices in V' does not permit edges (u, v) such that $u \in V' - \{q\}$ and $v \in V - V'$. Thus, for all the outgoing edges e of $u \in V' - \{q\}$, we have $dest(e) \in G'$.

At the time algorithm VISIT-CLCK visits vertex p, all the vertices $v \in \{|V'| - p\}$ of G' will be visited (and consequently assigned X coordinate) within the next $|V'| - 1$ calls of method VISIT-CLCK. The same holds for method VISIT-COCLCK. Thus, $X(q) = X(p) + |V'| - 1$ and $Y(q) = Y(p) + |V'| - 1$. □

Corollary 3.1. Let a dag $G = (V, E)$ and a pq-component $G_{pq} = (V', E') \subseteq G$. Let also $X(), Y()$, be the coordinates assigned by the algorithm WDP. Then for every vertex $u \in G_{pq}$, $X(p) \leq X(u) \leq X(p) + |V'| - 1$ and $Y(p) \leq Y(u) \leq Y(p) + |V'| - 1$.

Let $X()$ and $Y()$ be the coordinates constructed by WDP algorithm. Also let $G' = (V', E')$ be a component where $V' \subseteq V$ and $E' \subseteq E$. A component G' is said to be *separated*, if the following property holds for $X()$ and $Y()$:

$$\forall u \in V', v \in V - V' \Rightarrow (X(u) \leq X(v) \wedge Y(u) \leq Y(v)) \vee (X(u) \geq X(v) \wedge Y(u) \geq Y(v))$$

This property is a guarantee that every vertex $v \in V - V'$ that is not a member of a component G' will not appear between the vertices of G'. We refer to this as the *separation property*.

Theorem 3.1. Vertex placement $X()$ and $Y()$ constructed by algorithm WEAK DOMINANCE PLACEMENT respects the separation property for every pq-component.

Proof. Let $G_{pq} = (V', E') \subseteq G$ be a pq-component. Then algorithm TOPOLOGICAL - SORTING for G, numbers the vertices of G_{pq} from $X(p)$ to $X(p) + |V'| - 1$. Also holds for Y-coordinates, i.e., numbers vertices of G_{pq} from $Y(p)$ to $Y(p)+|V'|-1$. Thus, no vertex from $V-V'$ can be drawn inside a pq-component. □

Lemma 3.2. Let $X(), Y()$, be the coordinates assigned by the algorithm WDP to a dag $G = (V, E)$. Let u and v be a pair of vertices of G such that $X(v) = X(u) + 1$. Then $Y(u) < Y(v)$ if and only if G has an edge $(u, v) \in E$.

Proof. The 'if' part is trivial as if a vertex v is reachable from u through (u, v), then the inequality $Y(u) < Y(v)$ must hold. As for the 'only if', let's assume that $Y(u) < Y(v)$. The relation among vertices u and v can be an edge (direct connection), a path (indirect connection) or a falsely implied path.

If there is a path from u to v, there would be a vertex w distinct from u and v such that $u \to w \to v$. Thus, $X(u) < X(w) < X(v)$ a contradiction since $X(v) = X(u) + 1$.

Let us now assume for a contradiction that there is a falsely implied path from u to v. When algorithm TOPOLOGICAL SORTING(W) using VISIT-CLCK assigns X(u), it does not visit any of the adjacent node of u since $X(v) = X(u)+1$ and v is incomparable. Then, the algorithm back-tracks until it reaches a predecessor w of u (even if it is the artificial super-source s') that has an edge $e = (w, v) \in E$ such that e is the leftmost outgoing edge of w. This would imply that algorithm VISIT-COCLCK visits vertex v before vertex u. In this case we would have $Y(u) > Y(v)$, contradiction. Hence there is an edge $(u, v) \in E$. □

Lemma 3.3. Let $X(), Y()$, be the coordinates assigned by the algorithm WDP to a dag $G = (V, E)$. Let u and v be a pair of vertices of G such that $Y(v) = Y(u) + 1$. Then $X(u) < X(v)$ if and only if G has an edge (u,v).

Proof. Can be similarly constructed as proof of Lemma 3.2.

Our proposed framework contains the term 'overloaded' because all outgoing edges of a vertex, use the same column in order to reach their corresponding destination vertex. We will first discuss how a single edge is routed, and then we will focus on unambiguously visualizing the edges of the drawing.

Edge routing is automatically implied by the coordinates of the vertices. Each edge (u, v) consists of a vertical edge segment from $(X(u), Y(u))$ to $(X(u), Y(v))$ and a horizontal segment from $(X(u), Y(v))$ to $(X(v), Y(v))$. Because various edges reuse segments of rows and columns we introduce e-points to resolve ambiguities, see Figure 3.3. Given an edge (u, v) an *e-point* is defined as an unlabeled point that is placed on point $(X(u), Y(v))$ to indicate a direct connection from u to v.

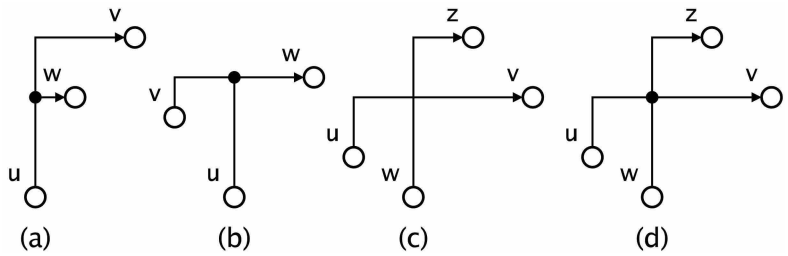


Figure 3.3: (a) the vertical segment of (u, v) is overloaded by the vertical segment of (u, w). To visualize the edge from u to w, an e-point is placed at $(X(u), Y(w))$. (b) the horizontal segment of (v, w) is overloaded by the horizontal segment of (u, w). To visualize the edge from u to w, an e-point is placed at $(X(u), Y(w))$. If there is no e-point then $(w, v) \notin E$ (c), whereas if there is an e-point in $(X(w), Y(v))$ then $(w, v) \in E$ (d).

We describe an algorithm that receives the vertex coordinates as an input, and outputs an overloaded orthogonal drawing. It routes the edges according to the given coordinates and places e-points where needed.

In order to construct an overloaded orthogonal drawing a linked data structure for G will be constructed. Each vertex $u \in V$ of G, points to the list of its direct successors sorted in decreasing order according to their Y-coordinate. This single linked list of u, can be traversed by means of pointer *next(u)*. It can also be accessed by pointer *getFirst(u)*, that is u's direct successor with the highest Y-coordinate (hence first in the list). In case of a tie, we can arbitrarily order vertices with the same coordinate without affecting the overall result.

Algorithm (OOD) OVERLOADED ORTH. DRAWING(Adj(G) , X() ,Y())

```
1. for each vertex u ∈ V
2.         visited[u]← 0
3. for each vertex u ∈ V in increasing order of X-coordinate
4.         v ← next(u)
5.         while v ≠ nil
6.                 Draw edge segment from (X(u),Y(u)) to (X(u),Y(v))
7.                 Draw edge segment from (X(u),Y(v)) to (X(v),Y(v))
8.                 if getFirst(u)≠v OR visited[v]≠0
9.                         New e-point ← (X(u),Y(v))
10.                visited[v] ← 1
11.                v ← next(v)
12. end
```

Theorem 3.2. Algorithm OOD produces an overloaded orthogonal drawing Γ of G with vertex coordinates computed by algorithm WDP. Γ has at most $n - 1$ bends, $O(n^2)$ area and is constructed in $O(n + m)$ time.

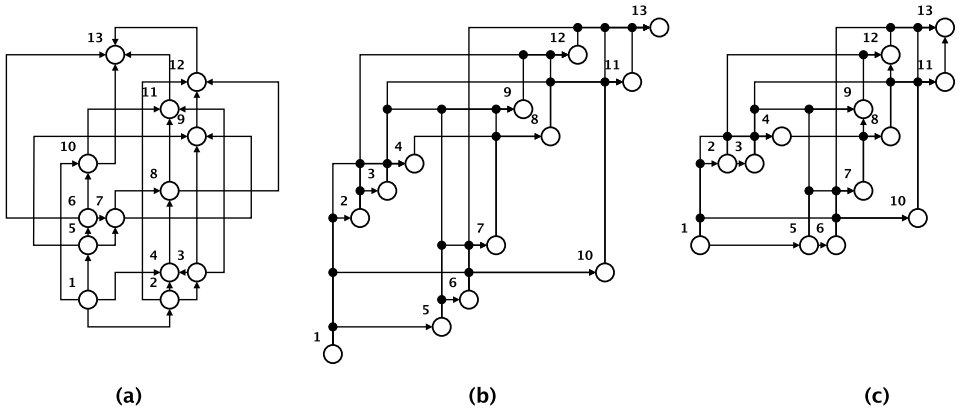


Figure 3.4: Three different drawings of a regular degree four graph with 13 vertices and 26 edges. (a) an orthogonal grid drawing is depicted, the graph and the drawing are taken from [3]. (b) an overloaded orthogonal drawing of the same graph. (c) a compacted overloaded orthogonal drawing.

3.3 Compaction

Compaction is applied as a post-processing step in an overloaded orthogonal drawing in order to reduce the X- and Y-coordinates. Our compaction follows the steps of the Algorithm in [3, 4]. However since our graphs are not planar, and therefore we do not have planar embeddings, we need to be extra careful in order to produce a valid drawing. In this step we allow equality between vertex coordinates under the following conditions: (a) The compaction is performed between vertices $u, v \in V$ such that there is an edge $(u, v) \in E$. (b) Two distinct vertices cannot coincide in the same point. (c) Compaction on the X- or Y- coordinates will not be performed if an edge is forced to pass over u or any other vertex.

Algorithm Y-COMPACTION(Adj(G) , X() ,Y())

1. **for** each vertex $u \in V$ in *increasing* order of Y-coordinate
2. $v \leftarrow$ next(u)
3. **while** $v \neq nil$
4. **if** predecessLowestX(v)=u AND $(u, v) \in E$ AND X(v)$\neq$X(u)
5. remove the bend in $(X(u), Y(v))$
6. $Y(v) \leftarrow Y(u)$
7. Move the horizontal segments of incoming edges of v, the e-points and the bends from $Y(v)$ to $Y(u)$
8. **else**
9. $Y(v) \leftarrow Y(u) + 1$
10. Move the horizontal segments of incoming edges of v, the e-points and the bends from $Y(v)$ to $Y(u) + 1$
11. **end**

Lemma 3.4. Let Γ be an overloaded orthogonal drawing of a directed acyclic graph G such that every vertex has distinct coordinate in X and Y-coordinates. Let also Γ' be the overloaded orthogonal drawing where algorithm Y-COMPACTION was performed as a post-processing step. Then, no edge is forced to pass over any vertex in Γ'.

Proof. We will prove that a compaction between vertices u, v in coordinate Y as described in Y-COMPACTION, does not force any edge to pass over a vertex. Let us assume for a contradiction, that a Y-compaction between u, v performed in Line 6 of algorithm Y-COMPACTION, is the first compaction step that results in an edge passing over a vertex.

According to Line 4 of the same algorithm, there must be an edge $(u, v) \in E$. Furthermore, since we are only considering vertices in increasing order of Y-coordinate, $Y(v) < Y(u)$. Taking into consideration that u and v are consecutive in the increasing Y order, $Y(v) = Y(u)+1$. So, the Y-compaction between u and v alters edge $e_1 = (u, v)$ as follows: the vertical segment of edge e_1 from $Y(u)$ to $Y(v)$ is eliminated, while the horizontal segment of e_1 from $X(u)$ to $X(v)$ is moved in the $Y(u)$-th row. Therefore, the vertices (resp. edges) that have Y-coordinate less than $Y(u)$ or greater than $Y(v)$ are not affected by this change. Thus, the only case which an edge passes over a vertex is when $\exists z \in V$ such that $e_2 = (z, v) \in E$ and $X(z) < X(u)$. In this case e_2 passes over vertex u. But from Line 4 we know that vertex u is the direct predecessor of v with the lowest X-coordinate, contradiction.

Finally, the displacement of e-points, bends, and incoming edges of v that occurs in Line 7 does not force any edge to pass over a vertex since the horizontal grid segment $((X(u), Y(u)), (n, Y(u))]$ is empty. □

Algorithm X-COMPACTION(Adj(G) , X() ,Y())

```
1.  for each vertex u ∈ V in increasing order of X-coordinate
2.      v ← next(u)
3.      while v ≠ nil
4.          if succesorHighestY(u)=v AND (u,v)∈ E AND Y(v)≠Y(u)
5.              if indegree(v)>1
6.                  remove the e-point in (X(u),Y(v))
7.              else
8.                  remove the bend in (X(u),Y(v))
9.              X(v) ← X(u)
10.             Move the vertical segments of outgoing edges of v, the
                e-points and the bends from column X(v) to column X(u)
11.         else
12.             X(v) ← X(u) + 1
13.             Move the vertical segments of outgoing edges of v, the
                e-points and the bends from column X(v) to column X(u) + 1
14. end
```

Lemma 3.5. Let Γ be an overloaded orthogonal drawing of directed acyclic graph G such that every vertex has distinct coordinate in X and Y-coordinates. Let also Γ' be the overloaded orthogonal drawing where algorithm X-COMPACTION was performed as a post-processing step. Then, no edge is forced to pass over any vertex in Γ'.

Proof. We will prove that a compaction between vertices u, v in coordinate X as described in X-COMPACTION, does not force any edge to pass over a vertex. Let us assume for a contradiction, that a X-compaction between u, v performed in Line 9 of algorithm X-COMPACTION, is the first compaction step that results in an edge passing over a vertex.

According to Line 4 of the same algorithm, there must be an edge $(u, v) \in E$. Furthermore, since we are only considering vertices in increasing order of X-coordinate, $X(v) < X(u)$. Taking into consideration that u and v are consecutive in the increasing X order, $X(v) = X(u) + 1$. So, the X-compaction between u and v alters edge $e_1 = (u, v)$ as follows: the vertical segment of edge e_1 from $Y(u)$ to $Y(v)$ remains the same, while the horizontal segment of e_1 from $X(u)$ to $X(v)$ is eliminated because v is placed in column $X(u)$. Therefore, the vertices (resp. edges) that have X-coordinate less than $X(u)$ or greater than $X(v)$ are not affected by this change. Also the vertices (resp. edges) that are placed in column $X(u)$ and have Y-coordinate strictly less than $Y(v)$, are not affected either.

Thus, the only case which an edge passes over a vertex is when $\exists z \in V$ such that $e_2 = (u, z) \in E$ and $Y(z) > Y(v)$. In this case e_2 passes over vertex v. But from Line 4 we know that vertex v is the direct successor of u with the highest Y-coordinate, contradiction.

Finally, the displacement of e-points, bends, and outgoing edges of v that occurs in Line 10 does not force any edge to pass over a vertex since the vertical grid segment $((X(u), Y(v)), (X(u), n)]$ is empty. □

3.4 Results on Area and Bends

In this section we present several properties and bounds of overloaded orthogonal drawings for directed acyclic graphs. Certain conditions must hold in order to form a bend.

Lemma 3.6. Let $(u, v) \in E$ be an edge where $u, v \in V$. As S_u and P_v are denoted the vertex sets of direct successors of u and direct predecessors of v, respectively. Edge (u, v) yields a bend if and only if the following three conditions hold:

- $u = arg\ min_{w \in P_v} X(w)$
- $v = arg\ max_{w \in S_u} Y(w)$
- $X(u) \neq X(v)$ and $Y(u) \neq Y(v)$

Proof. Let us assume that the above three conditions hold, we will prove that edge (u, v) yields a bend. According to the third condition, u cannot be on the

same column as v due to $X(u) \neq X(v)$. Also, u cannot be on the same row as v due to $Y(u) \neq Y(v)$. Thus, point $(X(u), Y(v))$ contains either a bend or an e-point. If there was an e-point on $(X(u), Y(v))$, there would be either (a) a vertex $w \in P_v$ such that $X(w) < X(u)$, (b) a vertex $z \in S_u$ such that $Y(v) < Y(z)$, or (c) both. Case (a) and (c) cannot hold because of the first condition, case (b) cannot hold due to the second condition. Thus, edge (u, v) yields a bend.

Let's assume that edge (u, v) yields a bend, we will prove that the above three conditions hold. Vertex u is the direct predecessor of v with the minimum X-coordinate, or else there would be an e-point instead of a bend in $(X(u), Y(v))$. Similarly, vertex v is the direct successor of u with the maximum Y-coordinate, or else there would be an e-point instead of a bend in $(X(u), Y(v))$. Finally since there is a bend $X(u) \neq X(v)$, and $Y(u) \neq Y(v)$. Therefore, the above three conditions hold. □

If both $X(u) \neq X(v)$, and $Y(u) \neq Y(v)$ hold for every pair of vertices $u, v \in V$, then every edge has a 'step'-like form and consequently produces either a bend or an e-point. Therefore we have:

Lemma 3.7. Let Γ be an overloaded orthogonal drawing of dag G, where each vertex is placed in a distinct X, Y coordinate. Then $bends(\Gamma) + ePoints(\Gamma) = m$.

If a compaction is performed on drawing Γ, then the sum $bends(\Gamma)+ePoints(\Gamma)$ would be less than the number of edges. Additionally, every vertex can have at most one bend on its row. That bend is produced from its direct predecessor with the lowest X-coordinate. Taking into consideration that sources do not have incoming edges, we have the following lemma:

Lemma 3.8. Let Γ be any overloaded orthogonal drawing of a dag G. Let also n_s be the number of sources of G. Then $bends(\Gamma) \leq n - n_s$.

Proof. Notice that a vertex can have at most one bend, and that every additional incoming edge will be an e-point on its row. Taking into consideration that sources do not have incoming edges, every overloaded orthogonal drawing will have at most $n - n_s$ bends. □

The upper bound of the above lemma is tight as shown by the following theorem.

Theorem 3.3. There exists a family of planar n-vertex graphs G_n, for $n \geq 3$, such that any overloaded orthogonal drawing Γ of G_n requires at least $n - 2$ bends, and $(n - 2) \times (n - 2)$ area.

Proof. Consider the graph G_n shown in Figure 3.5. Each vertex u_i has two outgoing edges, (u_i, u_{i+2}) and (u_i, u_{i+1}). The transitive closure of this family of graphs is a complete directed acyclic graph, therefore the topological sorting for this graph is unique. It can be constructed by placing the vertices according to their subindices in increasing order. Thus, for every vertex $u_i \in G$, the following equality holds $X(u_i) = Y(u_i)$. Their drawings admit a single compaction in Y-coordinate

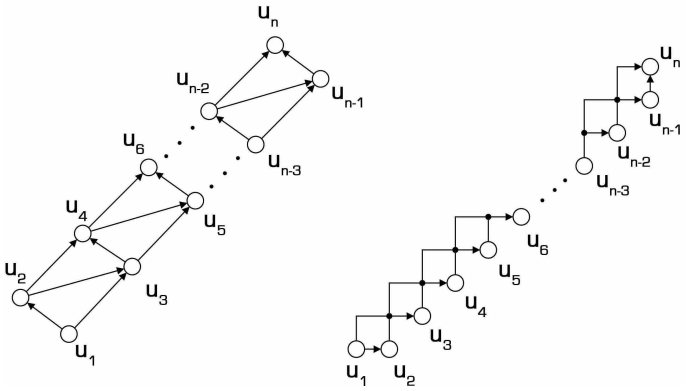


Figure 3.5: An explanatory construction of Theorem 3.3.

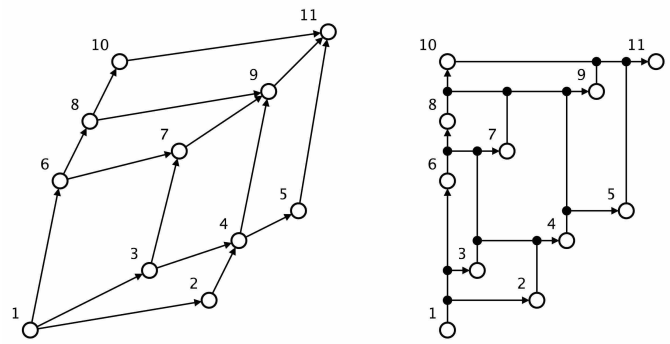


Figure 3.6: Left: The straight-line dominance drawing of a reduced planar *st*-graph as described in [3]. Right: A compacted overloaded orthogonal drawing of the same graph with zero bends.

between vertex u_1 and vertex u_2, and a single compaction in X-coordinate between vertex u_{n-1} and vertex u_n. Therefore an overloaded orthogonal drawing of this family of graphs has optimal area $(n-2) \times (n-2)$, and has at least $n-2$ bends. □

The dominance drawing technique was applied to reduced planar *st*-graphs in [4]. If we apply the edge routing technique using the vertex coordinates produced by the dominance drawing algorithm presented in [4], the resulting overloaded orthogonal drawing has zero bends.

Theorem 3.4. Given a reduced planar st-graph $G = (V, E)$, an overloaded orthogonal drawing Γ with zero bends can be constructed in linear time, $O(n)$.

Proof. Consider a reduced planar *st*-graph G. Assume that the vertex coordinates were obtained by Dominance Drawing algorithm [4], or else has Γ zero falsely implied paths. Then a vertex v is reachable from u if and only if $X(u) \leq X(v)$ and $Y(u) \leq Y(v)$. Let an edge $(u, w) \in E$ such that it forms a bend that cannot be removed by the compaction phase. In order to depict a bend all three conditions

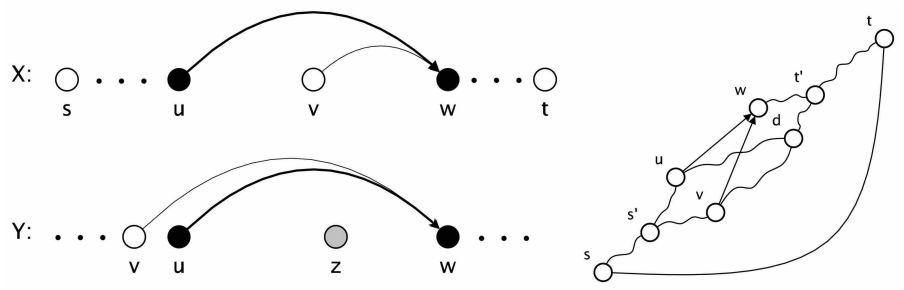


Figure 3.7: Proof of Theorem 3.4. Left: The difference between $z1$-case and $z2$-case . Right: A drawing of a $K_{3,3}$ that exists in $z2$-case.

of Lemma 3.6 must hold. Recall that G is embedded in the plane with s and t on the external face. We will denote by $u \to v$ the existence of such a path from u to v. By following the above assumptions, we will conclude that such a construction is not possible.

Let us assume that all conditions of Lemma 3.6 hold for edge (u, w). According to the third condition $X(u) \neq X(w)$ and $Y(u) \neq Y(w)$. If vertices u, w were consecutive in X-coordinate or Y-coordinate, then we could eliminate the bend performing a compaction step on X, Y respectively. But that contradicts the assumption that the bend cannot be removed by the compaction phase. Thus, vertices u, w cannot be consecutive neither in X nor in Y coordinate. Also the set of direct predecessors of w must contain more than one vertex, otherwise u, w would be consecutive.

Let $v \in V$ be another direct predecessor of w, we discuss the possible coordinates of v. Taking into consideration that the first condition holds, the inequality $X(u) < X(v) \leq X(w)$ must also hold. As for Y-coordinate, $Y(v)$ cannot be in the range $[Y(u), Y(w)]$ because in that case edge (u, w) would be transitive, contradiction. Therefore $Y(v) \leq Y(u)$. In X-coordinate, vertices u, w are not consecutive but according to our construction so far u, w are consecutive in Y-coordinate. If u, w were consecutive in Y, then we could eliminate the bend by a Y-compaction, therefore there must be a vertex z such that $Y(u) < Y(z) \leq Y(w)$.

Let us now analyze the X coordinate of z. Vertex z cannot be in the range $X(u) < X(z) < X(w)$, because in that case edge (u, w) would be a transitive edge. Thus, we have two cases to study, let $z1$ be the case where $X(z) < X(u)$ and $z2$ the case where $X(z) > X(w)$.

Case z1 ($X(z) < X(u)$): In that case we have $X(z) < X(w)$ and $Y(z) < Y(w)$, therefore vertex w is reachable from vertex z, $z \to w$. But, in order to follow condition one, we have already assumed that vertex u is the direct predecessor of w with the minimum X-coordinate. Hence, z is not a direct predecessor of w. Also z cannot reach u because we already assumed $Y(u) < Y(z)$. Thus, there must be a direct predecessor of w named f such that, f is reachable from z, $z \to f$. As for the coordinates of f, $X(z) < X(u) < X(f) \leq X(w)$ and $Y(u) < Y(z) < Y(f) < Y(w)$. Considering that we have zero falsely implied paths, vertex f must also be reachable from u, $u \to f$. But f is a direct predecessor of w,

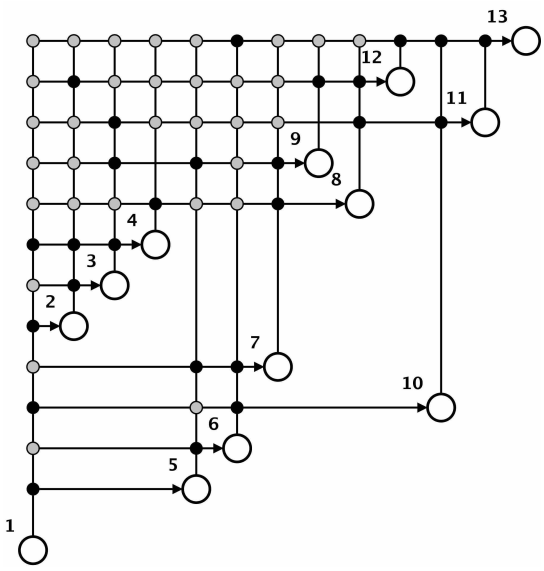


Figure 3.8: An overloaded orthogonal drawing of the transitive closure. Reachability of any pair of vertices u-v can be confirmed by looking at point $(X(u), Y(v))$. By the the color of the e-point we can determine if there is an edge or a path between the vertices.

so combining the fact $u \to f$ with the existence of edge (f, w), we conclude that edge (u, w) is transitive, contradiction.

Case z2 ($X(z) > X(w)$): In that case vertex z is reachable from u, since $X(z) > X(w) > X(v) > X(u)$) and $Y(w) > Y(z) > Y(u) > Y(v)$. According to the coordinates, vertex z is also reachable from v. Consider paths $s \to u$ and $s \to v$, and let s' be the last (farthest from s) vertex common to both paths. Likewise, let t' be the first (farthest from t) vertex common to paths $w \to t$ and $z \to t$. By the above definitions and the dominance property, G has the following paths

$$s' \to u \ , \ s' \to v \ , \ w \to t' \ , \ z \to t' \ ,$$

$$u \to w \ , \ u \to z \ , \ v \to w \ , \ v \to z \ .$$

Since s and t are on the external face, we can add the edge (s, t) to G, while preserving planarity of the embedding. One can easily verify that the paths listed above, plus edge (s, t) form a graph that is homomorphic to $K_{3,3}$. This fact contradicts the planarity of G. □

3.5 Clarity and readability of the Model

In this section we outline some advantages of the overloaded orthogonal model.

• *Meaningful relation between vertex coordinates*: The weak dominance condition implies that: if there is a path from u to v then vertex v will appear in the

upper right quadrant of vertex u.

• *Works for any pair of topological sortings as X,Y coordinates*: Since every pair of topological sortings respects the weak dominance condition, we can take any pair of topological sorting as X, Y coordinates.

• *Efficient Visual Confirmation of an Edge*: We can visually confirm the existence of an edge (u, v) by checking if there is an e-point or a bend on point $(X(u), Y(v))$. If a compaction is performed, u or v replace the e-point at the location $(X(u), Y(v))$. In contrast, in the regular orthogonal model we would have to visually follow every outgoing edge of u successively, until we reach v. Consequently, the size of a graph does not affect the readability of an overloaded orthogonal drawing, as we can check if any two vertices are connected by inspecting only a single point i.e., in $O(1)$ time.

• *Efficient Visual Confirmation of Reachability*: An interesting extension of this graph drawing technique occurs when we use the transitive closure of a graph as input. In that case every possible path along the original directed acyclic graph $G = (V, E)$ will be represented by an edge in the transitive closure $G^* = (V, E^*)$. By applying the overloaded orthogonal model we can check if a vertex v is reachable from a vertex u by examining point $(X(u), Y(v))$ in the drawing. As shown in Figure 3.8, the corresponding dots for transitive edges are colored grey and called p-points (p for path). Notice that there is no e-point at $(X(4), Y(9))$, despite the fact that the coordinates of vertex 9 dominate the coordinates of vertex 4. In this context, crossings indicate the existence of falsely implied paths.

Chapter 4

Information Visualization via DAGView

In this chapter, we propose a novel visualization framework called *DAGView* which contains a family of algorithms based on the Overloaded Orthogonal model for directed acyclic graphs. DAGView visualizations present extraordinary clarity as well as several characteristics of matrix representations. Our approach combines the readability and scalability of a matrix based approach, with the intuitiveness of a node-link approach in spotting a node within the layout and in following an edge to find its destination. Several criteria that users identified as important in a layout are met, such as underlying grid, crossings that appear perpendicular, easy check for the existence of an edge and/or path.

The DAGView framework visualizes the directed acyclic graphs, as described in Chapter 3. Our main goal in this chapter is to extend the techniques described for directed acyclic graphs in order to handle directed graphs with cycles and undirected graphs. In doing so the algorithms take into account user preferences and/or constraints.

4.1 An Introduction to Matrix Representations

Matrix-based representation of graphs offer an alternative to the traditional node-link diagrams. It is used in several user-studies and many different implementations have been proposed.

Matrix-based representation is a visualization technique where a boolean-valued n-by-n connectivity matrix is used. Columns denote the origin-node of the edge, while the rows denote the target-node of the edge. If there is an edge (u, v) the cell at the intersection of column u and row v, is colored black. Some interesting questions were posed in the work of Ghoniem et al. [23], where the authors describe a user study comparing the matrix representation over the node link representation of graphs. Users were asked to give fast and correct answers on several tasks such as counting the number of nodes and edges, finding the most connected node, finding if there was an edge between two specified nodes as well as finding a path between nodes. The crossing of edges and the lack of topology were addressed by the users

as the two main factors that caused difficulties in reading node-link representations.

Several tools implement the matrix representation technique. MatrixExplorer [24] is a network visualization system that uses both node-link diagrams and matrices. The user can hop from one representation to the other as well as interacting with the matrix representation. Zoomable Adjacency Matrix Explorer (ZAME) [17] is a tool that takes advantage of the scalability of matrix representation and enables exploration of graphs with millions of nodes and edges. MatLink [25] is a tool implementing a composite representation, where links overlay on the border of the matrix visualization. Another recent paper [2] partially uses the matrix-based representation to produce hybrid visualizations.

4.2 Handling Cycles in Directed Graphs

As mentioned before, in case the input graph is a dag the methodology described in Chapter 2 is applied. In this section we discuss how to handle directed graphs with cycles. We describe how to handle this graphs in order to be able to visualize them with the DAGView model. Specifically, we temporarily remove some edges in order to gain an directed acyclic subgraph. The removed edges are routed in separately and colored red. Furthermore, we describe how to visualize the strongly connected components in order to simplify the visualization.

4.2.1 Presenting Feedback Arcs

By following the weak dominance property (described at the previous chapters), we cannot visualize all the directed edges that are contained in a cycle. Besides, we cannot place the vertices in the grid since the existence of a cycle does not allow the construction of a topological sorting. Therefore, we will temporarily invert the direction of a set of edges such that the graph will become a directed acyclic graph. This set of edges is also known as feedback arc set. Computing the minimum feedback arc set is NP-hard, thus we use heuristics that compute a minimal feedback arc set. One such algorithm is the Greedy-Cycle-Removal algorithm [3].

Let $G = (V, E)$ be a directed graph with cycles. First we compute the minimal feedback arc set and denote it as E'. Next, we invert the direction of the edges in E'. The DAGView model is used to visualize the new directed acyclic graph since it contains no cycles. Then we visualize the edges of the graph $G = (V, E - E')$ according to the DAGView model. We focus on how to visualize an edge $(v, u) \in E'$ that belongs to the minimal feedback arc set. Notice that even though u is reachable from v, node u is placed in the lower-left quadrant of v. Also notice that all the incoming edges of node u in Γ are assembled in the row $\mathrm{Y}(u)$ of the grid. Thus, the $\mathrm{Y}(u)$ row has no edge segment to the right of point $(X(u), Y(u))$. Similarly, all the outgoing edges of node v are assembled in the column $\mathrm{X}(v)$ of the grid. Thus, the $\mathrm{X}(v)$ column has no edge segment below the point $(X(v), Y(v))$. We use these empty segments a) below v and b) to the right of u, to draw the feedback arc (v, u).

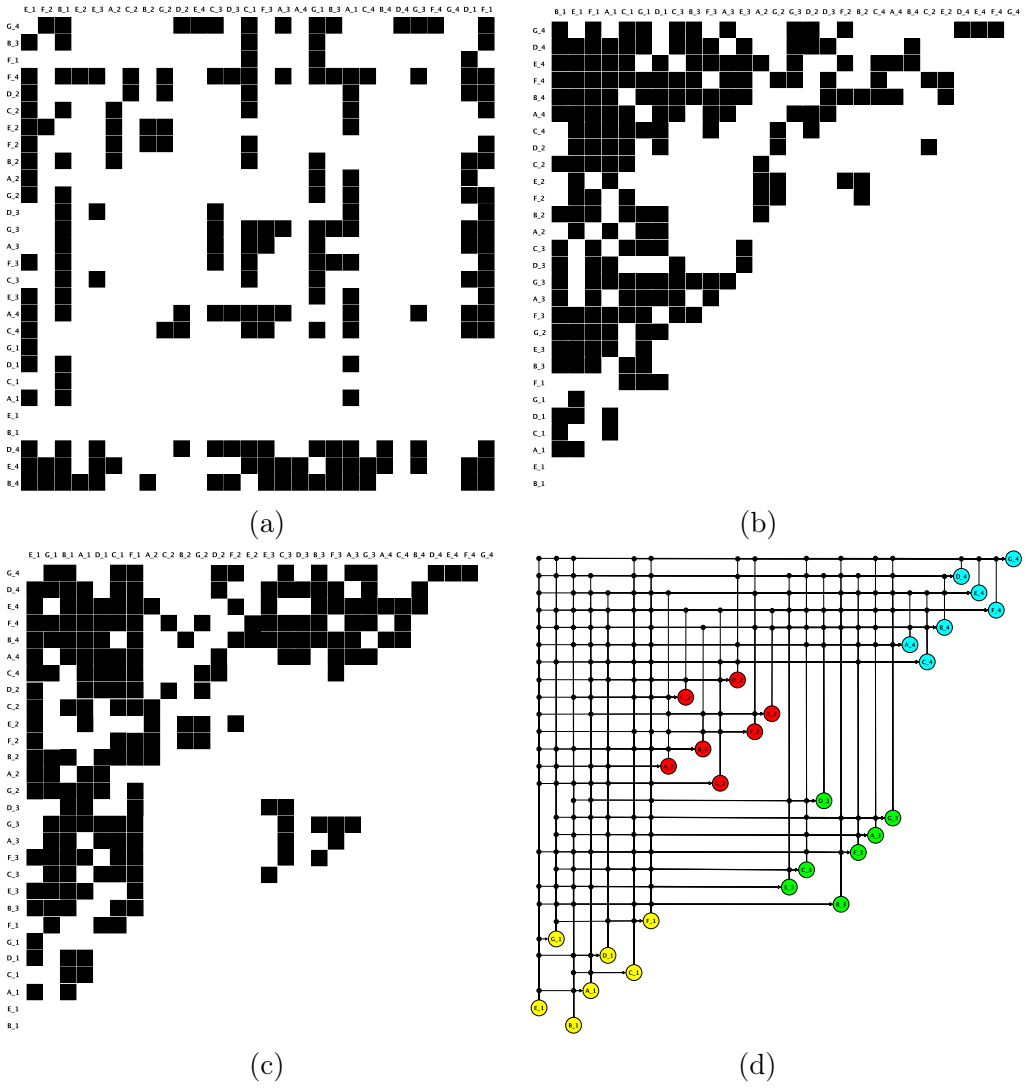


Figure 4.1: Four visualizations of the same directed acyclic graph containing 28 nodes and 200 edges. The diagram (a) depicts a matrix representation where columns and rows are randomly ordered. Diagram (b) depicts a matrix representation where columns appear in decreasing order according to their out-degree. The order of rows and columns in diagram (c) is the same as the order of topological sortings in the DAGView visualization of (d).

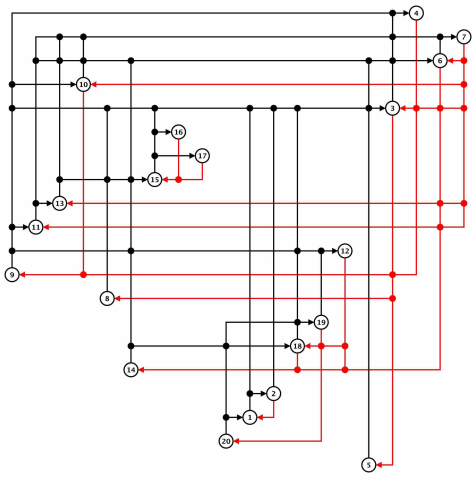


Figure 4.2: A DAGView visualization of directed graph with cycles that has 20 nodes and 59 edges.

But with this edge visualization of (v, u) we violate the weak dominance property. In order to highlight this violation, the edges in E' are colored red and are oriented from top-to-bottom and from right-to-left. Formally, each edge $e' = (v, u) \in E'$ will have a vertical segment from point $(X(v), Y(v))$ to point $(X(v), Y(u))$, and a horizontal segment from point $(X(v), Y(u))$ to point $(X(u), Y(u))$.

In this framework we also need to introduce the corresponding red e-points for the feedback arcs. Given a feedback arc (v, u) the e-point is defined as an unlabeled red point that is placed on point $(X(v), Y(u))$ to indicate a direct connection from v to u. An example is depicted in Figure 4.2.

4.2.2 User-chosen feedback arcs

In the previous subsection we chose the feedback arc set according to the output of an algorithm. Here we discuss the scenario where the user chooses a number of edges that (s)he wants to be a part of the feedback arc set. An example may come from the model of Resource Description Framework (or RDF). Several techniques visualize RDF graph ontologies from top-to-bottom. But there are classes of edges that represent different relation between the nodes and are directed from bottom-to-top. In this case the user might have a preference on which edges to route as feedback arcs. The visualization of graph ontologies has been extensively studied [48].

In this approach the user chooses an edge set E_u that will be a subset of the feedback arc set. If the set E_u is not a feedback arc set, the Greedy-Cycle-Removal algorithm is applied on the directed graph with edge set $E - E_u$. At the output of Greedy-Cycle-Removal, edges of E_u are added resulting in an augmented feedback arc set. Then we temporary inverse the direction of the edges of the augmented feedback arc set. A DAGView visualization is produced according to the new directed acyclic graph. Finally the edges of the augmented feedback arc set are

visualized in red following the orientation from top-to-bottom and from right-to-left, according to the technique described before.

4.2.3 Handling Strongly Connected Components

Under certain circumstances users might want to inspect a more high-level picture of a directed graph with cycles. In other words, the user might want to hide some information from the visualization, in order to gain a more general picture. A clustering scheme for strongly connected components with the DAGView model is discussed. An example is depicted in Figure 4.3.

First we have to give the definition of edge contraction. The process of removing an edge (u, v) while simultaneously merging nodes u and v is called edge contraction, or simply contraction. This method can also be generalized for a component, in this case all the edges of the component are contracted resulting in a new artificial node (also called super-node) in order to simplify the visualization.

Strongly connected components can be computed in linear time by using standard algorithms. Next, components are contracted resulting in a new directed acyclic graph G' that has a super-node per each strongly connected component. The DAGView model is applied in order to visualized G'. The visualization of each strongly connected component can be computed separately following the technique described in the previous subsection. A strong advantage of our framework is that the detailed visualization of a strongly connected component can be easily added within the already drawn visualization of G'. An interactive feature of the DAGView model can be the following: by clicking on the contracted strongly connected component $G_i = (V_i, E_i)$ of G', a square $|V_i|$-by-$|V_i|$ will be extended where the detailed visualization of G_i will appear. All the nodes that reach G_i have been placed in the lower-left quadrant of G_i, whereas the nodes that can be reached from the nodes of G_i appear in the upper-right quadrant of G_i. Thus, the rest of the visualization does not change and we preserve the user's mental map of G'.

Algorithm Strongly Connected Clustering ( )
1. Compute strongly connected components $S = \{S_1, ..., S_k\}$ of G
2. Contract each component S_i into a super-node s_i, obtaining G'
3. Compute X,Y coordinates of G' and construct the visualization according to the DAGView model
4. For each strongly connected component S_i compute a minimal feedback arc set, construct and save separately the DAGView of S_i.
5. Replace the node s_i of the DAGView with its detailed visualization whenever the user clicks on s_i

4.3 Undirected Graphs

In this section we propose methods to visualize undirected graphs within the DAGView model. One mainstream technique that has been widely used in the

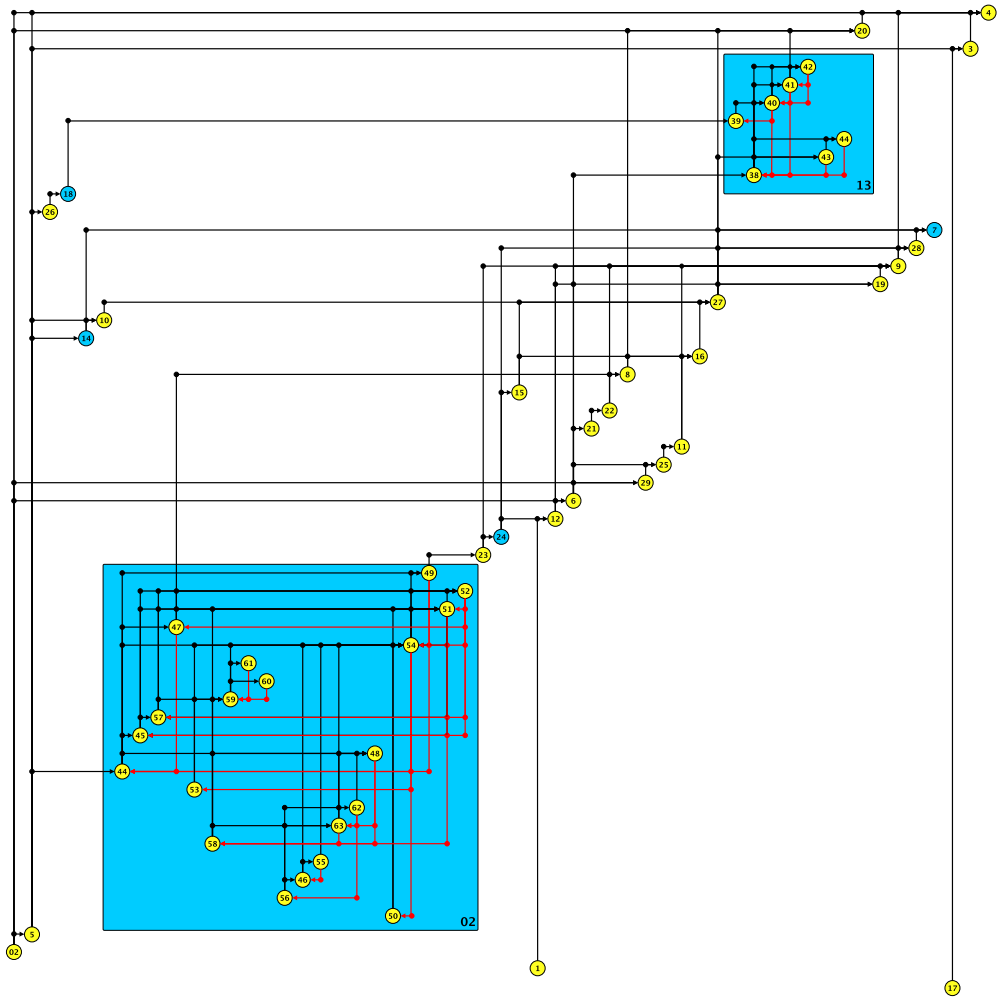
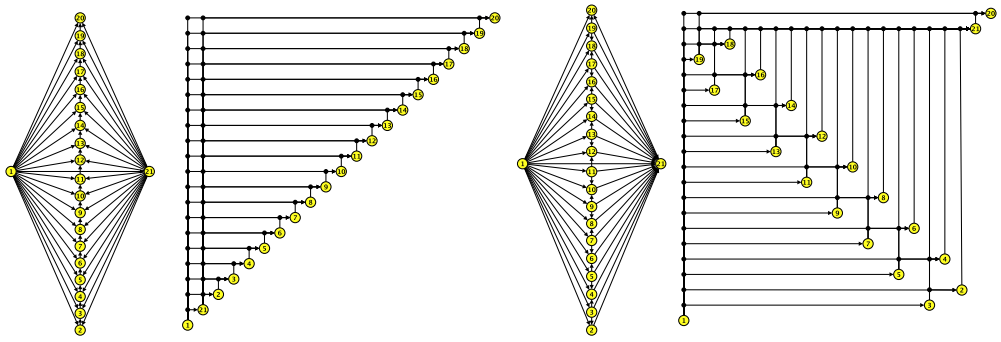


Figure 4.3: A directed graph with 55 nodes and 135 edges, out of which 31 are marked as feedback arcs with red color. Nodes that are colored blue are strongly connected components that have been contracted into a single node. Inside the blue frames are depicted in detail two out of the six contracted strongly connected components.



(a) (b)

Figure 4.4: In (a) a *st*-orientation of the corresponding undirected graph that constructs a long longest path ($p = 1$), and its DAGView visualization. In (b) there is the same undirected graph, the edge orientation constructed a short longest path ($p = 0$), and its DAGView visualization.

graph drawing area, is to orient the undirected edges in order to obtain a directed graph. For an undirected graph with n nodes and m edges there are 2^m ways to orient the edges.

There are two main approaches for orienting edges in a undirected graph. The first is to orient edges in order to avoid cycles. The computation of an *st*-orientation is in this direction. Specifically, we use a parametrized technique in order to control the length of the longest path in the resulting *st*-oriented directed graph. The second approach is to construct a directed graph with cycles. We discuss the scenario in which the user gives a set of cycles as an input. In case the graph is not connected one can apply these techniques on each connected component individually.

4.3.1 Controlling the length of the Longest Path

The problem of orienting an undirected graph such that it has one source, one sink, and no cycles is called *st*-orientation. This is always possible if the undirected graph is biconnected. For our work we deploy a parametrized technique [39] in which we are able to control the length of the longest path of the resulting directed acyclic graph.

The main idea of this algorithm is based on the successive removal of nodes and the simultaneous update of the t-rooted block-cutpoint tree. The authors proposed an algorithm that takes a constant $p \in [0, 1]$ and computes a directed acyclic graph of length of longest path that is a function of p. For small values of p the algorithm computes an *st*-orientation with small longest path, while for high values of p it computes an *st*-orientation with long longest paths. These algorithms run in $O(mlog^5n)$ time for general graphs and in $O(mlogn)$ time for planar graphs, for more information see [39]. By controlling the *st*-orientation of the undirected graph, we influence the DAGView visualization.

Different *st*-orientations give different DAGView visualizations as can be seen in

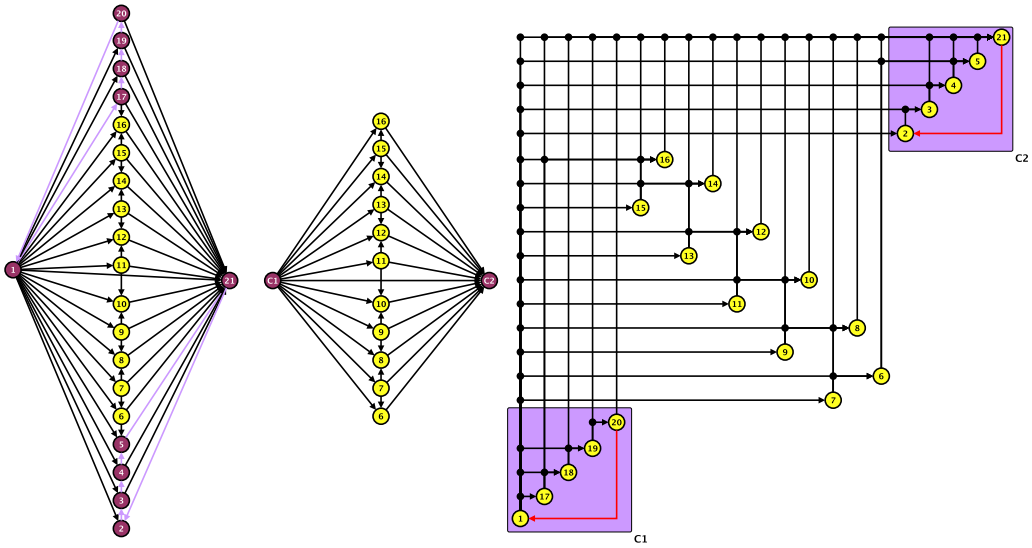


Figure 4.5: In this Figure the scenario of user-chosen clusters for undirected graphs is illustrated. The undirected graph is the same as the one of Figure 4.4. With purple are depicted the two different cycles chosen by the user. In the first figure the orientation of the edges is described. The second figure depicts the contracted directed graph and the third figure depicts the final DAGView visualization.

Figure 4.4. The directed acyclic graph that was constructed with the long longest path algorithm, Figure 4.4-(a) , has a Hamiltonian path. Therefore it has only one topological sorting, which is used for both X and Y coordinate. Due to this fact, nodes are placed in the diagonal from the lower-left corner to the upper-right corner of the grid. Whereas, the directed acyclic graph that was constructed with the short longest path algorithm, Figure 4.4-(b), produced a different DAGView visualization.

4.3.2 User-chosen Clusters

In this scenario the user gives as an input a set of node-disjoint cycles $C = \{C_1, ..., C_k\}$ of an undirected graph $G = (V, E)$. The goal is to orient the edges of each cycle C_i in order to form a strongly connected component c_i which can be easily contracted into a super-node as described in the previous section.

We will discuss how to handle cycle C_i, the other cycles are treated similarly. Let (w, z) be an undirected edge of C_i. Let us also assume that the user chose a direction for the edges of the cycle, in which the undirected edge (w, z) is oriented from w to z. Each undirected chordal edge of C_i is oriented so as to conform with the direction from z to w. After we have oriented all the edges of cycle C_i and its chordal edges, we temporarily invert the direction of (w, z) and put it in the edge set E'. The set E' is be formed in order to visualize the feedback arc set. We repeat the above process for every cycle in C.

Next, we compute an st-orientation of the remaining undirected edges of G,

considering each cycle C_i as a contracted super-node c_i. The reason we handle cycles as super-nodes is because we want to have consecutive X and Y coordinates for the vertices of each cycle. The directed acyclic graph $G = (V, E - E')$ is visualized according to the DAGView model. The direction of edges of E' is inverted back to the original direction and are drawn in red as feedback arcs. In the final visualization the strongly connected components can be visualized either in their original form or as super-nodes. An illustrative example is presented in Figure 4.5.

4.4 Visualizing Large & Disconnected Graphs

In this section we will address some issues on the topic of visualizing graphs with many nodes. The scalability of our technique will be discussed as well as techniques to handle different connected components of the same graph.

4.4.1 Scalability of the model

Massive graph models tend to appear in many scientific fields, from gene interaction to World Wide Web. These graphs are dense, highly connected and have tens of thousands of edges and vertices. Therefore there is an emerging need for visualization techniques that are scalable and can produce readable layouts of large graphs.

Even though DAGView visualizations are categorized as node-link representations, they have the advantages of matrix-based representations. In DAGView the number of crossings is small and their layout is orthogonal. Moreover, columns and rows are reused in order to hide as much redundant information as possible. Large graphs are modularly displayed in order to obtain clear an readable visualizations. The DAGView model aims at providing a scalable visualization technique for navigation and easy exploration through large scale graphs. An example is depicted in Figure 1.4 and in Figure 4.8.

4.4.2 Connected components & Tiles

In this section we discuss how to visualize graphs that have a number of different connected components. Let G be a directed graph and $D = \{D_1, ..., D_k\}$ be the set of connected components that partitions G. We will study each connected component as a separate graph and consider its visualization as a tile (or rectangle). Therefore the weak dominance property will be applied for each component individually. If we handled the graph as a whole we would need a size $(n-1) \times (n-1)$ grid to visualize G. Whereas, with the assumption that each connected component is treated independently, we reduce the area of visualization. For instance one can visualize components separately and then place them horizontally in decreasing size order. A consequence of this approach is that exactly one node from each of the k components, will have Y coordinate equal to 1. In this case the graph is expanded only towards the X axis. So, the weak dominance property does hold within each connected component, but it does not hold when comparing nodes from different components.

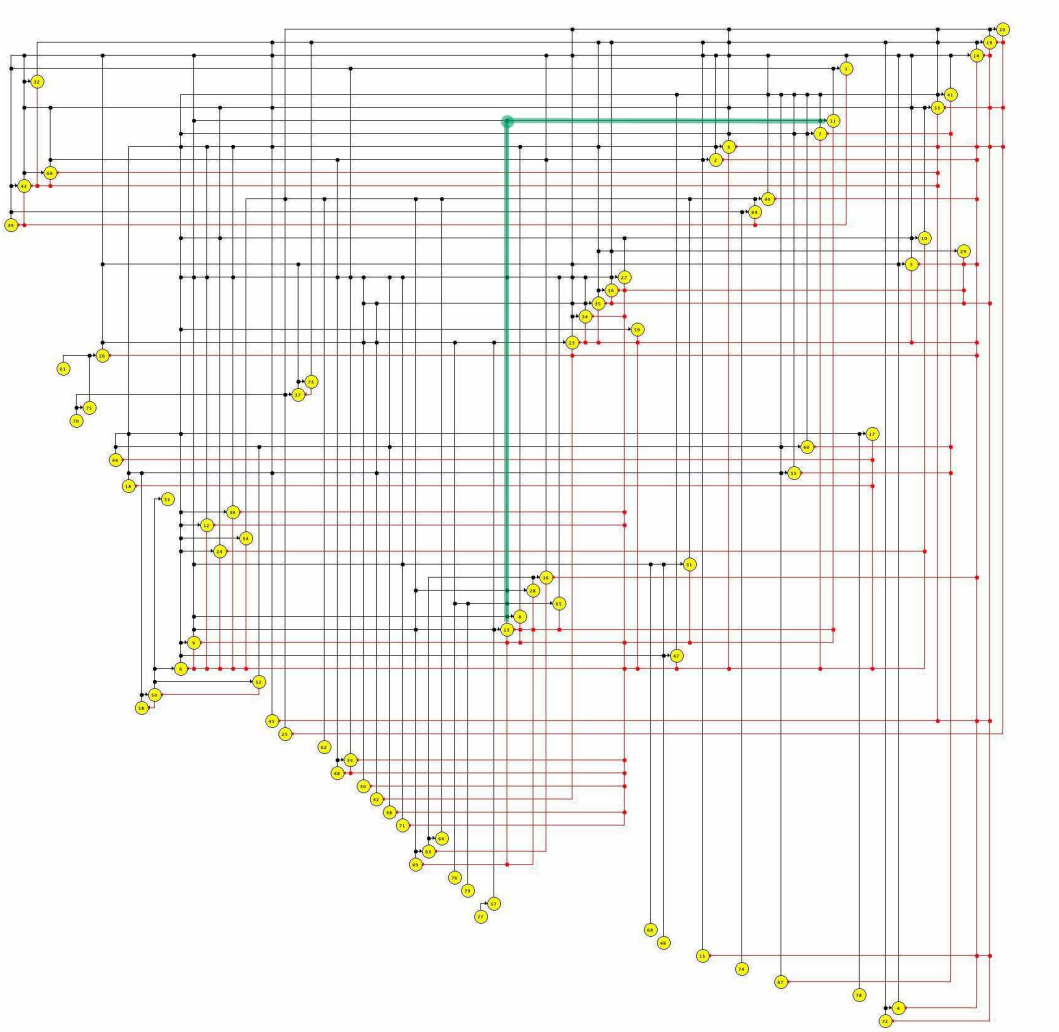


Figure 4.6: In order to check if there is an edge from node u to node v in a DAGView visualization, we have to inspect only one point of the grid $(X(u), Y(v))$. If there is an e-point, then there is an edge. In this visualization the edge is highlighted with a green color.

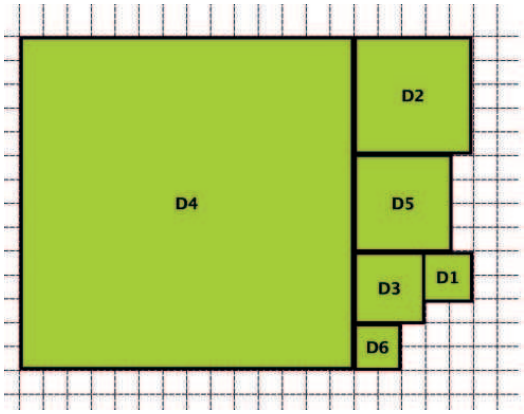


Figure 4.7: A graph with six connected components $D = \{D1, ..., D6\}$. The connected components(rectangles) are placed so as to reduce the overall area of the visualization.

Thus, the visualization of the graph G is performed in two steps, in the first we visualize the connected component and in the second we place the k tiles according to an algorithm. The placement of tiles can be described as a rectangle packing problem. The rectangle packing problem consists of finding an enclosing rectangle of smallest area that can contain a given set of rectangles without overlap [27]. It is known that this optimization problem is NP-hard. Therefore, heuristics and approximation algorithms must be used in order to visualize G in a compact manner.

4.4.3 Edge highlight

Highlighting an edge is a common feature in graph drawing tools, especially when dealing with large graphs. This feature can be implemented as follows in the DAGView model: when the user places the mouse over an e-point, the edge is highlighted. This approach assists the user in locating nodes that are placed in a great distance by inspecting the e-point that connects them. This simple but effective interactive enhancement helps users focus on graph elements. In most node-link layouts the user has to visually follow a highlighted edge which changes orientation at every bend. In our case the user anticipates a certain geometry in the edge that will be highlighted, a vertical segment followed by a horizontal. The feature that we propose will be of great help in experimental studies where users are answering an extensive series of questions and they may need assistance in performing more complicated tasks. An illustrative example is depicted in Figure 4.10.

4.5 Comparing with User-Studies

In recent years there is a trend to study graph representation from a cognitive point of view. Experimental studies were deployed in order to evaluate the aesthetics and the readability of different graph layouts [42, 43]. Grid drawings is an important goal for graph drawing techniques, this is confirmed even recently in [41]. Purchase

et al. note that besides the minimization of edge crossings, aligning nodes and edges to an underlying grid is equally important.

In [14] the authors conduct an experiment to compare user-generated and automatic graph layouts. As it is indicated, studies of the human factors in readability of a layout, should inform the algorithm design process in order to optimize the most valuable heuristics. In this experiment users chose graphs based on general aesthetics, most commonly a symmetric, ordered, or clean look. In addition to this, in several graphs crossings are not removable. Therefore they will be part of the visualization regardless of the layout algorithm that the designer will choose. In [28] an eye tracking experiment was conducted in order to study the effects of crossing angles as well as the geometric-path tendency on users's eye movement. Among the findings, authors concluded that when edges cross at 90 degrees, eye movement may be slightly slowed down, but it remains smooth.

The DAGView model combines several important features from the traditional node-link representation and the matrix-based representation. According to the results in [23], for small graphs node link diagrams are always more readable and more familiar than matrices. On the contrary, for larger graphs the performance of node link diagrams deteriorates quickly while matrices remain readable with comparable answer time. Here we discuss the features of the DAGView model in terms of the purely relational method which is extensively used in user studies. In the relational method, the primary concern is the accuracy and efficiency with which people can read the graph structure and answer questions about it. The tasks that we discuss are posed to users at several user studies [23, 28, 43].

Find an edge. In a DAGView visualization in order to confirm the existence an edge (u, v) the user has to focus on only three points of the visualization. The node u which is located in $(X(u), Y(u))$, the node v which is located in $(X(v), Y(v))$ and the point $(X(u), Y(v))$ of the underlying grid. If there is an e-point at $(X(u), Y(v))$, then edge (u, v) is part of the graph. Note that if node v is in the lower-left quadrant of node u then the edge is a backedge and it is depicted in red. In the traditional node-link representation with a straight-line edge routing, the user has to locate nodes u and v and then follow one-by-one the outgoing edges of u until (s)he finds v. If the routing of the edges is not a straight line then the user has to follow edges with bends which might be confusing due to the lack of topology. Whereas, in the matrix representation the user has to locate the column that corresponds to node u, and the row that corresponds to node v. Then if the point in the intersection of the column of u and the row of v is black, the edge (u, v) belongs to the edge set of G. So, the task of finding an edge in a DAGView visualization is as effective as in a matrix-based representation.

Locate nodes. The DAGView model is a node-link representation. Therefore the task of locating a node is very intuitive. The user has to inspect the plane in order to locate the node, a common procedure to users that are familiar with graphs. However in a matrix-based representation, nodes are depicted as linearly ordered labels at the column/row. In such a scenario users might have difficulty recalling the exact position of the column that they spotted at a previous task, since there isn't any distinctive feature (such as location on the grid or topology)

to assist their memory. Hence, the task of locating a node in the DAGView model is similar to the node-link model.

Count nodes. Since each node has a distinct coordinate, columns and rows of a DAGView visualization represent nodes of the graph. This fact reduces the task of counting nodes into the task of counting the number of rows in the underlying grid of the DAGView visualization. In a node-link representation, as studies have pointed out [23], users have difficulty when confronting this task in large graphs. A possible reason for this behavior is that the user has to scan the two-dimensional plane of the visualization while counting nodes. On the other hand in the matrix representation model (as in the DAGView model), the problem is reduced to counting in one dimension instead of two.

Find a path. Paths can be easily visualized in the DAGView model for directed acyclic graphs. Let Γ be a DAGView visualization of a directed acyclic graph G. If the user is aware of the existence of a path from u to v and has located node u, then due to the Weak Dominance property (s)he has to inspect only the upper-right quadrant of u in order to locate v. This property reduces the area of the two dimensional plane that the user has to check for node v. Moreover, the feature of the transitive closure extension offers all the reachability information of a directed acyclic graph with the touch of a button. With this technique the user can efficiently answer any question about path existence, while preserving the mental map of the visualization. Because of the way edges are drawn in a DAGView visualization, a path is a sequence of vertical and horizontal edge segments. Intuitively, a path from u to v can be described as a 'stairway' of varied size steps, that begin from point $(X(u), Y(u))$ and end at point $(X(v), Y(v))$. In a matrix-based representations this task is complex since nodes are represented twice (once on both axes of the matrix), which forces the user's eye to move from the row representing a node to its associated column back and forth. Whereas in a DAGView visualization, the user has both the row and the associated column once (s)he locates the node in the plane.

Find the most Connected node. In a DAGView visualization the outgoing edges of a node u are located in the column $X(u)$, and the incoming edges in the row $Y(u)$. To locate the most connected node, the user has to glance at a row/column that is proportional to the degree of the node. Even though in this task we are interested only in the number of neighbors, when processing e-points of a column the name of the neighbors comes in at no cost since there can be only one node in each row. Users might have better performance on this task in a traditional node-link representation since they are inspecting locally a node in order to count the edges that have u as an origin/destination. But in a traditional node-link visualization of dense graphs, the user will have difficulties counting the edges that are associated with a node.

Besides these typical tasks, there are generic characteristics of a DAGView visualization that are helpful to users. Several of the following characteristics were addressed in experimental studies by the users themselves as 'important features' [28]. Also some of the addressed features were observed in studies [41] where the user was free to ideally visualize the graph from scratch.

Perpendicular crossings. As indicated in [28] crossings that appear perpendicular have the minimum effect on the readability of the visualization. The DAGView model not only constructs perpendicular crossings, but also 'hides' a great number of them due to the column/row reuse. This general characteristic of this model enables us to study the problem of minimizing edge crossings from a completely different perspective.

Visualize on an underlying grid. According to experimental studies [41], when users are asked to draw a graph from scratch, they tend to use a unit grid formation to place the nodes. Moreover they draw edges vertically and horizontally, this indicates that the structure that is proposed in the DAGView model follows the tendencies of users.

Scalable Model. As in a matrix-based representation [23], the readability of a DAGView visualization does not deteriorate when the size of the graph and the link density increases. The DAGView model inherits the scalability of a matrix-based representation. On the other hand traditional node-link diagrams (i.e. straight line) are sensitive to density and size variation [23].

Preserve the Mental Map. Important features such as the clustering of strongly connected components, the user-chosen cycles and the transitive closure extension, preserve user's mental map of the DAGview visualization. This is especially important [40] when the user has to perform several tasks sequentially on the same graph. In that case the user might need to recall the topology of the layout. In the clustering scheme the algorithms modify the visualization locally while preserving the relative position of the nodes. Whereas the transitive closure feature extends the visualization of the directed acyclic graph without changing the position of the already drawn features of the DAGView visualization.

Applicable to every graph. The DAGView model was originally designed to visualize directed acyclic graphs efficiently with the Overloaded Orthogonal model. In this chapter we extended the model, and described techniques in order to visualize both undirected graphs and directed graphs with cycles. Additionally, the user may take active role and give as input constraints or preferences to modify the final DAGView visualization.

4.6 A Java implementation of DAGView

The DAGView tool is an application written in Java programming language, that visualizes graphs according to the DAGView framework. The tool is designed to be modular and to run as an applet. DAGView tool can read graph/network files written in SIF format. The SIF format specifies nodes and interactions only, while other formats store additional information about network layout. The resulting visualizations can be exported in JPG image file format.

From this tool the user can inspect several properties of the graph structure and its visualization. In the right menu of our tool, we present the value of several important aspects such as: the number of nodes, the number of edges, the number of feedback-arcs, the number of self loops, the number of connected components, the number of strongly connected components as well as the number of fips.

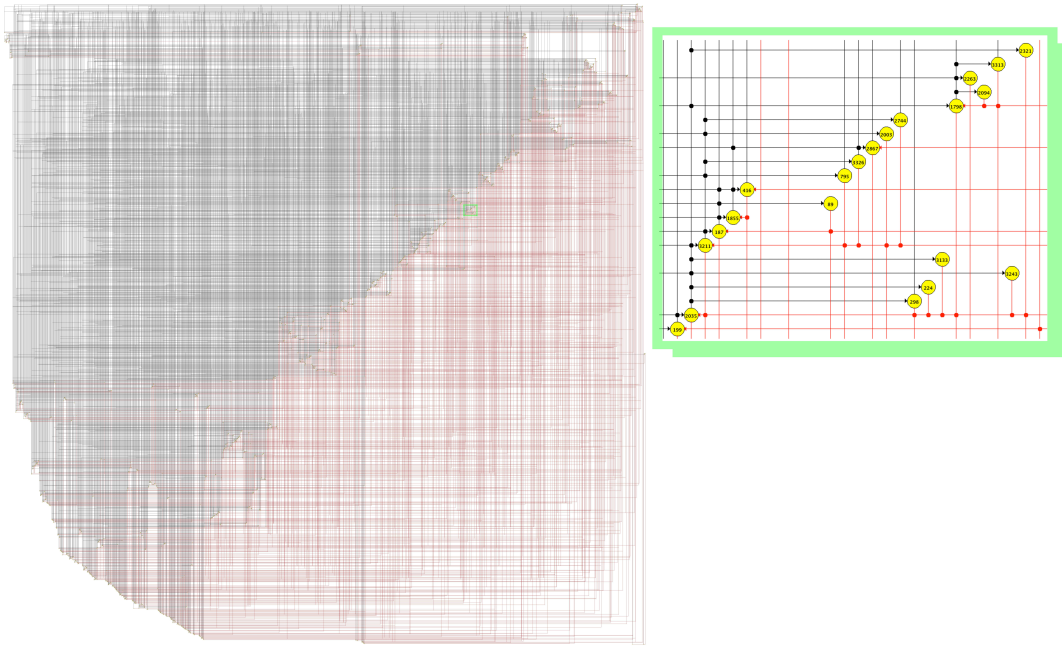


Figure 4.8: This DAGView visualization depicts a strongly connected component graph that was created from Wikipedia articles. The 1.515 nodes represent Wikipedia articles about composers and the 10.528 edges represent links between these articles. On the right there is a detailed visualization of the highlighted region.

The analysis of graphs is based on a library called FlexiGraph developed and maintained by Dimitris Andreou [1] which provides the basic graph functionality. The implementation was partially developed by Nikos Chouliaras who collaborated with our research team in the Institute of Computer Science-F.O.R.T.H. as a junior software engineer.

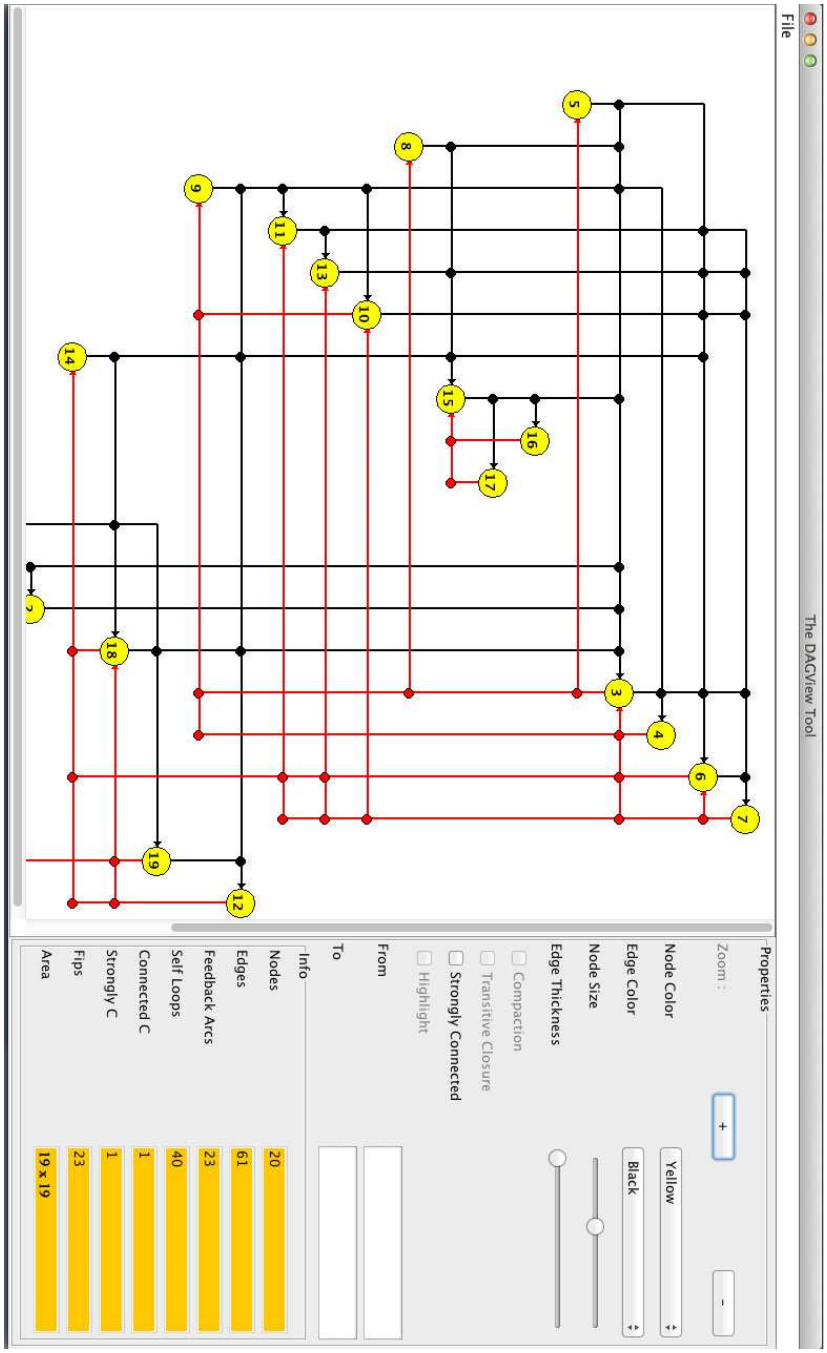


Figure 4.9: A view of the DAGView Tool.

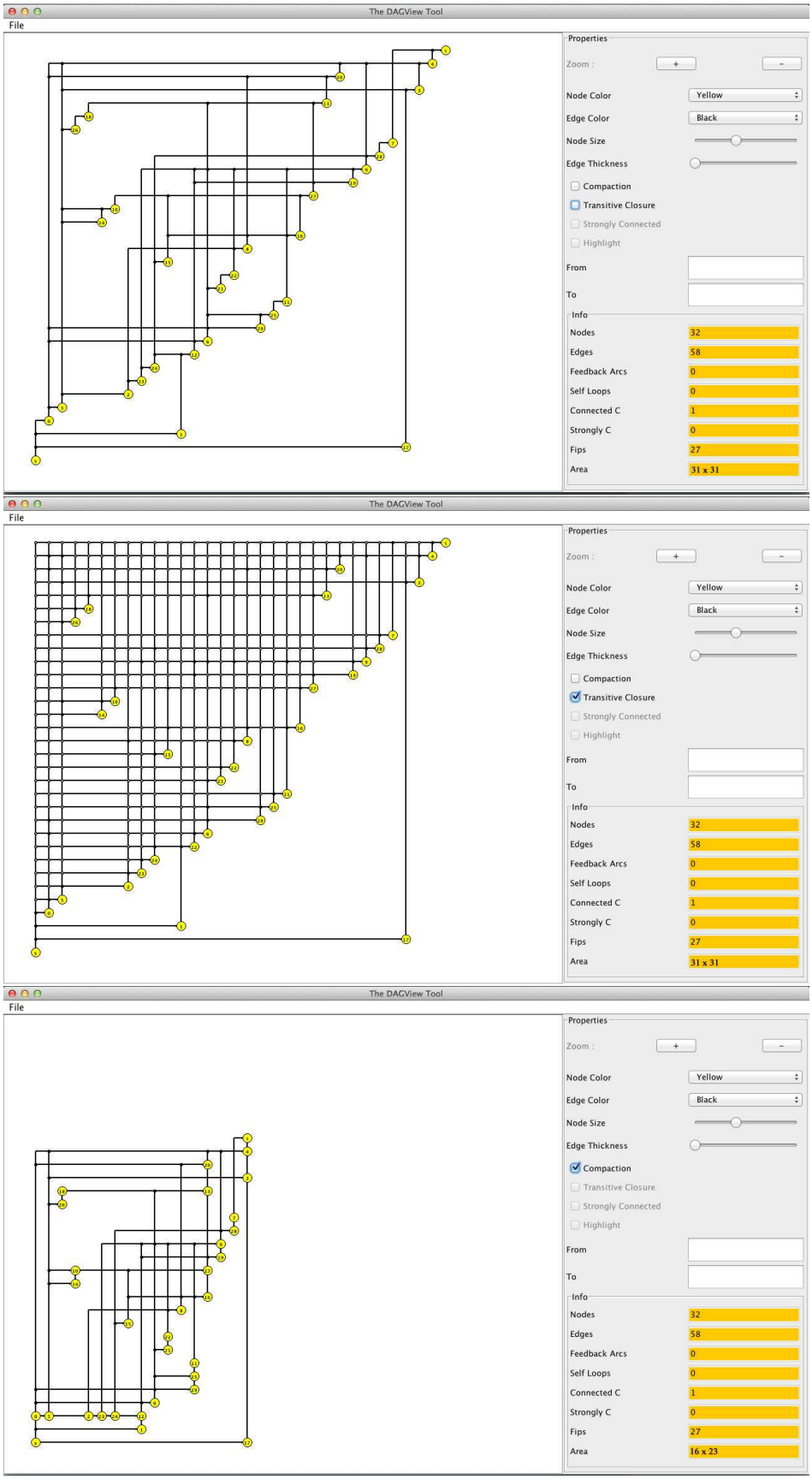


Figure 4.10: Features of the DAGView Tool.

Chapter 5

Conclusions & Open Problems

In this thesis we developed a new framework for visualizing graphs. We generalized the Dominance drawing technique that is proposed for planar *st*-graphs, in order to be able to include the broader family of directed acrylic graphs. From a theoretical perspective, we formulated a computational problem named Weak Dominance Placement which lies in the very core of our generalization. Based on this problem we proposed a graph drawing model, presented as Overloaded Orthogonal Model which can be applied to directed acyclic graphs. As a next step we extended the Overloaded Orthogonal Model by introducing the DAGView framework which can handle directed graphs with cycled as well as undirected graphs.

There are several interesting open questions related to our work. An interesting topic is the Fixed Weak Dominance Placement problem. In this problem we are given a topological sorting t_1 of a dag G as an input, and the goal is to construct a topological sorting t_2 in order to minimize the number of falsely implied paths $fips(t_1, t_2)$. We touched on this problem to form heuristic algorithms for the Weak Dominance Placement problem. The complexity of the Fixed Weak Dominance Placement problem is an open question.

Another interesting variation is to duplicate a number of vertices of a graph G in order to form a graph G' that admits a weak dominance placement with zero falsely implied paths. Ideally we would like to minimize the number of vertices that we duplicate. It is an open problem to find algorithms that minimize the number of duplications.

The problem of Weak Dominance Placement is expressed using two topological sortings, thus we are essentially referring to a two-dimensional placement. It is interesting from a theoretical perspective to further extend this problem for $k > 2$ dimensions. What would be the complexity of that problem? An advancement in this direction may have implications in the field of Order Theory too. The weak dominance placement is related to the linear extension diameter on the linear extension graph. How can one express the general problem for $k > 2$ linear extensions with notions from the linear extension graph?

There are interesting questions that are related with the overloaded orthogonal model too. One approach is to develop algorithms for vertex placement that perform well for specific families of graphs (for instance planar graphs, bipartite

graphs etc.). An even more promising problem is the following, consider the minimization of falsely implied paths as a metric for two topological sortings. What other metrics for two topological sortings can be formulated? An advancement in this direction would result in a new graph drawing model.

Furthermore it would be very interesting to measure the performance of users in the DAGView framework. How would DAGView perform comparing to a matrix-based representation and traditional graph drawing techniques? Finally, there is also room for heuristic algorithms concerning directed graphs with cycles and undirected graphs.

Bibliography

[1] D. Andreou. flexigraph library. *http://code.google.com/p/flexigraph/*.

[2] V. Batagelj, F. J. Brandenburg, W. Didimo, G. Liotta, P. Palladino, and M. Patrignani. Visual analysis of large graphs using (x, y)-clustering and hybrid visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1587–1598, 2011.

[3] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[4] G. D. Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete & Computational Geometry*, 7:381–401, 1992.

[5] P. Bertolazzi, G. D. Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transaction on Computers*, 49(8):826–840, 2000.

[6] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Computational Geometry*, 9(3):159 – 180, 1998.

[7] T. C. Biedl, B. Madden, and I. G. Tollis. The three-phase method: A unified approach to orthogonal graph drawing. *International Journal of Computational Geometry and Applications*, 10(6):553–580, 2000.

[8] G. Borradaile and P. Klein. An o(n log n) algorithm for maximum st-flow in a directed planar graph. *Journal of the ACM*, 56(2):9:1–9:30, 2009.

[9] G. Borradaile, P. Klein, and C. Mathieu. An o(n log n) approximation scheme for steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):31:1–31:31, 2009.

[10] G. Brightwell and M. Massow. Diametral pairs of linear extensions. *CoRR*, abs/0809.1828v1, 2008.

[11] G. Brightwell and P. Winkler. Counting linear extensions is #**P**-complete. In *ACM Symposium on Theory of Computing*, pages 175–181, 1991.

[12] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications*, 9(1):31–52, 2005.

[13] B. Dushnik and E. W. Miller. Partially ordered sets. 63(3):600–610, 1941.

[14] T. Dwyer, B. Lee, D. Fisher, K. I. Quinn, P. Isenberg, G. G. Robertson, and C. North. A comparison of user-generated and automatic graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):961–968, 2009.

[15] M. Eiglsperger, S. P. Fekete, and G. W. Klau. Orthogonal graph drawing. In *Drawing Graphs*, pages 121–171, 1999.

[16] P. E. H. A. ElGindy, M. E. Houle, W. Lenhart, M. Miller, D. Rappaport, and S. Whitesides. Dominance drawings of bipartite graphs. 1993.

[17] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete. Zame: Interactive large-scale graph visualization. In *Proc. of IEEE Pacific Visualization Symposium*, pages 215 –222, 2008.

[18] D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent layered drawings. *Algorithmica*, 47(4):439–452, 2007.

[19] S. Felsner and M. Massow. Linear extension diameter of downset lattices of two-dimensional posets. *SIAM Journal on Discrete Mathematics*, 25(1):112–129, 2011.

[20] S. Felsner and K. Reuter. The linear extension diameter of a poset. *SIAM Journal on Discrete Mathematics*, 12(3):360–373, 1999.

[21] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In *Proc. of Graph Drawing*, volume 1027, pages 254–266, 1995.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[23] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proc.of the IEEE Symposium on Information Visualization*, pages 17–24, 2004.

[24] N. Henry and J.-D. Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12:677–684, 2006.

[25] N. Henry and J.-D. Fekete. Matlink: enhanced matrix visualization for analyzing social networks. In *Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction - Volume Part II*, INTERACT'07, pages 288–302, 2007.

[26] J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[27] E. Huang and R. E. Korf. New improvements in optimal rectangle packing. IJCAI'09, pages 511–516, 2009.

[28] W. Huang. Using eye tracking to investigate graph layout effects. In *Proc. of International Asia Pacific Symposium on Visualization (APVIS)*, pages 97–100, 2007.

[29] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.

[30] P. N. Klein. Multiple-source shortest paths in planar graphs. In *Proc. of the sixteenth annual ACM-SIAM Symposium On Discrete Algorithms*, pages 146–155, 2005.

[31] E. M. Kornaropoulos and I. G. Tollis. Overloaded orthogonal drawings. In *Proc. of Graph Drawing*, pages 242–253, 2011.

[32] E. M. Kornaropoulos and I. G. Tollis. Weak dominance drawings and linear extension diameter. volume abs/1108.1439, 2011.

[33] R. M. McConnell and J. P. Spinrad. Linear-time transitive orientation. In *Proceedings of the eighth annual ACM-SIAM Symposium On Discrete Algorithms*, SODA '97, pages 19–25, 1997.

[34] M. Naatz. The graph of linear extensions revisited. *SIAM Journal on Discrete Mathematics*, 13(3):354–369, 2000.

[35] K. Nomura, S. Tayu, and S. Ueno. On the orthogonal drawing of outerplanar graphs. *IEICE Transactions*, 88-A(6):1583–1588, 2005.

[36] S. Ntafos and S. Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, 5:520–529, 1979.

[37] A. Papakostas and I. G. Tollis. Orthogonal drawing of high degree graphs with small area and few bends. In *Proc. Algorithms and Data Structures: 5th International Workshop (WADS'97)*, volume 1272, pages 354–367, 1997.

[38] A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry*, 9(1-2):83–110, 1998.

[39] C. Papamanthou and I. G. Tollis. Algorithms for computing a parameterized st-orientation. *Theoretical Computer Science*, 408(2-3):224–240, 2008.

[40] H. Purchase, E. Hoggan, and C. Görg. How important is the "mental map" ? – an empirical investigation of a dynamic graph layout algorithm. volume 4372, pages 184–195, 2007.

[41] H. Purchase, C. Pilcher, and B. Plimmer. Graph drawing aesthetics-created by users, not algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):81 –92, 2012.

[42] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proc. of Graph Drawing*, pages 248–261, 1997.

[43] H. C. Purchase, R. F. Cohen, and M. I. James. An experimental study of the basis for graph drawing algorithms. *Journal on Experimental Algorithmics*, 2, 1997.

[44] M. S. Rahman, T. Nishizeki, and M. Naznin. Orthogonal drawings of plane graphs without bends. *Journal of Graph Algorithms and Applications*, 7(4):335–362, 2003.

[45] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[46] R. Tamassia and I. Tollis. Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems*, 36(9):1230 –1234, 1989.

[47] W. T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, 1992.

[48] Y. Tzitzikas and J. L. Hainaut. On the visualization of large-sized ontologies. pages 99–102, 2006.

[49] L. Vismara, G. D. Battista, A. Garg, G. Liotta, R. Tamassia, and F. Vargiu. Experimental studies on graph drawing algorithms. *Software: Practice and Experience*, 30(11):1235–1284, 2000.

[50] M. Yannakakis. The Complexity of the Partial Order Dimension Problem. *SIAM Journal on Algebraic and Discrete Methods*, 3(3):351–358, 1982.

[51] yWorks GmbH. "yed graph editor". *www.yworks.com*.