UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

Analysis and Visualization of Directed Graphs

by

Panagiotis Lionakis

PhD Dissertation

Presented

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Heraklion, January 2023

University of Crete
Computer Science Department

Analysis and Visualization of Directed Graphs

PhD Dissertation Presented by
Panagiotis Lionakis
in partial fulfillment of the Requirements for the
Degree of Doctor of Philosophy in Computer Science

APPROVED BY:

Author: Panagiotis Lionakis

Supervisor: Ioannis G. Tollis, Professor, University of Crete

---

Committee Member: Giuseppe Liotta, Professor, University of Perugia

---

Committee Member: Kostas Stefanidis, Associate Professor, Tampere University

---

Committee Member: Antonis Argyros, Professor, University of Crete

---

Committee Member: Ioannis Tsamardinos, Professor, University of Crete

---

Committee Member: Polyvios Pratikakis, Associate Professor, University of Crete

---

Committee Member: Constantine Manasakis, Associate Professor, University of Crete

---

Department Chairman: Antonis Argyros, Professor, University of Crete

Heraklion, January 2023

"Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution."
-Albert Einstein

The drawing of the previous page represents the following:

i) It depicts a simple tree as sketched by Leonardo Da Vinci. This schematic "Rule of Trees" was based only on observation and suggested that *"if you fold all the branches of a tree upward, they will combine to create a continuation of the trunk with the same surface area"*. Regardless of the scientific proof of this statement, this blindingly obvious observation shows that imagination and observation is the motivation to the greatest research questions.

> *"The formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill. To raise new questions, new possibilities, to regard old questions from a new angle, requires creative imagination and marks real advances in science."*

ii) Observation is essential in science. Scientists use observation to collect and interpret the data into useful information in order to extract the results. Data, is like truth. It is always from a certain point of view and not everyone can perceive it.
iii) This dissertation takes advantage of hierarchies and hierarchical drawings of directed graphs.

Στην αγαπημένη μου Σταυρούλα, στους γονείς μου Κώστα και Τασία και στον αδερφό μου Φίλιππο, για την υποστήριξή τους όλα αυτά τα χρόνια

Acknowledgements

The end of this thesis could not be typed without the help and support of my mentor, advisors, my family and my friends. First and foremost, I offer my sincerest gratitude to my supervisor and mentor, Professor Ioannis G. Tollis for providing me with knowledge, support and motivation throughout this journey. During the past years, he has devoted a lot of time and effort for helping me not only with my PhD dissertation but also by encourage me and support me until today. He has been a critic and at the same time a supportive advisor and mentor. His insights have been really inspiring and crucial to me.

Apart from my supervisor, I would like to deeply thank the other two members of my Advisory Committee, Professor Giuseppe Liotta and Professor Kostas Stefanidis, for their valuable comments and suggestions all this time until this point, which helped me to improve my PhD thesis. Moreover, I would also like to thank the rest of the examining committee, Professor Antonis Argyros, Professor Ioannis Tsamardinos, Professor Polyvios Pratikakis and Professor Constantine Manasakis for their evaluation and constructive comments on this thesis.

Additionally, I want to express my deepest thanks to the undergraduate and postgraduate students of Computer Science Department which help me with the experiments in order to finish this dissertation.

Finally, I want to thank the people that supported me all these years: My family and my friends. This work is a result of the constant support of my family, my parents Kostas and Tasia and my brother Filippos. They always believed in my abilities and supported me in any possible way.

Abstract

The display of Graphs, is widely used for the visualization of data or information. Although it is important to show all the information represented by the edges and the nodes of the graph, there are cases where this is either impossible due to the complexity and the size of the graphs or even misleading since the graph could be very dense, while other reasons may be related to privacy protection (e.g., social network graphs). To overcome this challenge, different approaches imply to hide the unnecessary or redundant information or even replace the original graph with a subgraph or a summary.

The main objective of this dissertation is to investigate and elaborate on the visualization and analysis of large graphs. More specifically, regarding vizualization we design and implement various versions of a new sophisticated framework of graph layout techniques that focus on the idea of improving the visualization aesthetics, in order to reduce the visualization complexity of the graph, on top of sophisticated graph drawing layouts. These techniques can be categorized in two main groups. The first category is with respect to bends while the second is based on edge removal. To this respect, we perform a set of experiments that show that these techniques produce progressively more abstract drawings of the input graph. No dummy vertices are introduced and the vertices of each path/channel are vertically aligned. Subsequently, the value of the introduced approach is that it also provides a generic and parameterized visualization method based on the given scenario. Towards this direction, we also elaborate on methods for grouping nodes with similar characteristics that naturally decomposes the graph based on the information derived by the edges within the graph, which can be used in order to have better visualization of complex networks.

In order to evaluate our drawing layout techniques, we assess the usability of various versions of our new layout compared with the one produced by other hierarchical

drawing techniques. As a result, we provide insights regarding the factors that affect efficiency so introduce a set of metrics in order to evaluate the performance of these techniques. Our algorithms require almost linear time. Moreover, we design and conduct a comparative task-based evaluation with users in which we ask the participants to carry them out in order to extract the user's satisfaction level and exploit any possible problems and difficulties as regards the reachability information. Generally, the drawings produced by our algorithms have lower number of bends and are significantly smaller in area. The user evaluation also reveals that the performance of the participants is slightly better in the drawings of our proposed model and that our model is preferred in overall rating compared to the other model.

Keywords: Hierarchical Graph Drawing, Directed Graphs, Crossings, Bends, Reachability, Abstraction of edges, Graph Drawing, Graph Algorithms, Experimental and User Study, Information Visualization

Supervisor: Professor Ioannis G. Tollis

Computer Science Department

University of Crete

Περίληψη

Η απεικόνιση των Γράφων, χρησιμοποιείται ευρέως για την οπτικοποίηση δεδομένων ή πληροφοριών. Αν και είναι σημαντικό να εμφανίζονται όλες οι πληροφορίες που αντιπροσωπεύονται από τις ακμές και τους κόμβους του γράφου, υπάρχουν περιπτώσεις όπου αυτό είναι είτε αδύνατο λόγω της πολυπλοκότητας και του μεγέθους του γράφου ή ακόμα και παραπλανητικό. Αυτό έγκειται στο γεγονός ότι οι εν λόγω γράφοι ενδέχεται να είναι αρκετά πυκνοί, ενώ άλλες περιπτώσεις σχετίζονται με προσωπικά δεδομένα και την ιδιωτικότητα (π.χ κοινωνικά δίκτυα). Για να ξεπεραστεί αυτή η πρόκληση, διαφορετικές προσεγγίσεις σχετίζονται με την απόκρυψη της περιττής ή πλεονάζουσας πληροφορίας ή ακόμα και τη μερική αντικατάσταση του αρχικού γράφου με υπογράφο ή με μια περίληψη αυτού.

Ο βασικός στόχος της παρούσας διατριβής είναι η λεπτομερής διερεύνηση περί οπτικοποίησης και ανάλυσης μεγάλων γράφων. Πιο συγκεκριμένα, αναφορικά με την οπτικοποίηση, σχεδιάζουμε και υλοποιούμε μια σειρά από διαφορετικές προσεγγίσεις ενός νέου μοντέλου γραφικής αναπαράστασης που εστιάζουν στην βελτίωση της γραφικής αναπαράστασης, προκειμένου να μειωθεί η οπτική πολυπλοκότητα του γραφήματος, πάνω από εξελιγμένες τεχνικές σχεδίασης για γράφους. Οι τεχνικές αυτές μπορούν να κατηγοριοποιηθούν σε δύο κύριες ομάδες. Η πρώτη κατηγορία σχετίζεται με τις γωνίες ενώ η δεύτερη με την αφαίρεση άκρων. Τα πειραματικά αποτελέσματα ανέδειξαν ότι αυτές οι τεχνικές παράγουν προοδευτικά πιο αφαιρετικά γραφήματα. Δεν εισάγονται εικονικοί κόμβοι και οι κόμβοι κάθε διαδρομής/καναλιού είναι κάθετα ευθυγραμμισμένοι. Η αξία της προσέγγισής μας είναι ότι παρέχει μια γενική αλλά παραμετροποιήσιμη μέθοδο οπτικοποίησης συναρτήσει του δοθέντος σεναρίου. Επιπροσθέτως, ασχολούμαστε με μεθό-

δους ομαδοποίησης κόμβων με παρόμοια χαρακτηριστικά, οι οποίες ομαδοποιούν τον γράφο βάσει των πληροφοριών που προέρχονται από τις ακμές, και οι οποίες μπορούν να χρησιμοποιηθούν για την καλύτερη οπτικοποίηση πολύπλοκων δικτύων.

Προκειμένου, να αξιολογήσουμε τις τεχνικές σχεδιαστικής διάταξης, συγκρίνουμε τη χρηστικότητα των αποτελεσμάτων διαφόρων εκδόσεων του νέου μας μοντέλου με τα αντίστοιχα αποτελέσματα που παράγονται από παρόμοιες τεχνικές ιεραρχικής σχεδίασης. Ως αποτέλεσμα, τονίζουμε τις γενικές κατευθυντήριες γραμμές και παρέχουμε πληροφορίες σχετικά με τους παράγοντες της αποτελεσματικότητας. Οι τεχνικές μας απαιτούν σχεδόν γραμμικό χρόνο. Επιπλέον, σχεδιάζουμε και εφαρμόζουμε μία συγκριτική αξιολόγηση βασισμένη σε Δραστηριότητες με χρήστες, προκειμένου να εξαγάγουμε το επίπεδο ικανοποίησής τους καθώς και να αναδείξουμε τα όποια πιθανά προβλήματα και δυσκολίες. Τα αποτελέσματα αναδεικνύουν ότι τα γραφήματα βάσει του προτεινόμενου μοντέλου μας, έχουν μικρότερο αριθμό γωνιών και απαιτούν σημαντικά μικρότερη γεωμετρική περιοχή ενώ το μοντέλο μας προτιμάται στη συνολική βαθμολογία σε σύγκριση με το άλλο μοντέλο.

Λέξεις κλειδιά: Σχεδίαση Ιεραρχικών Γράφων, Κατευθυνόμενοι Γράφοι, Διασταυρώσεις Ακμών, Προσβασιμότητα, Αφαίρεση Ακμών, Σχεδίαση Γράφων, Αλγόριθμοι Γράφων, Πειραματική Έρευνα και Έρευνα με Χρήστες, Οπτικοποίηση Πληροφορίας

Επόπτης: Ιωάννης Γ. Τόλλης

Καθηγητής

Τμήμα Επιστήμης Υπολογιστών

Πανεπιστήμιο Κρήτης

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and Motivation

Hierarchical graphs are very important for many applications in several areas of research and business because they often represent hierarchical relationships between objects in a structure. They are directed (often acyclic) graphs and their visualization has received significant attention recently [1–3]. In a directed graph G, a Feedback Arc Set (FAS) is a set of edges whose removal leaves G acyclic. Computing a minimum FAS is important for visualizing directed graphs in hierarchical style [3, 4]. In fact, the first step of well known frameworks for hierarchical graph drawing is to compute a minimum FAS [5, 6]. Directed Acyclic Graphs (DAGs) are often used to describe processes containing some long paths, such as in PERT applications see for example [7, 8]. The paths can be either application based, e.g. critical paths, user defined, or automatically generated paths.

In their seminal paper of 1981, Sugiyama, Tagawa, and Toda [6] proposed a four-phase framework for producing hierarchical drawings of directed graphs. This framework is known in the literature as the Sugiyama framework, or algorithm. Even though the Sugiyama framework is very popular, most problems involved in the optimization of various phases of this framework are NP-hard. The Path Based Hierarchical Drawing Framework (PBF) exploits a new approach to visualize directed acyclic graphs that focus on their reachability information [9]. This framework is orthogonal

to the Sugiyama framework in the sense that it is a vertical decomposition of G into (vertical) paths/channels. The vertices of a graph G are partitioned into paths, called a *path decomposition* and the vertices of each path are drawn *vertically aligned*. It consists of only two steps: (a) the cycle removal step (if the graph contains directed cycles) and (b) the hierarchical drawing step.

Although it is important to show all the information represented by the edges of the graphs, in several applications, such as graph databases and big data, the graphs are very large and the usual visualization techniques are not applicable due to the complexity and the size of the graphs or even misleading since the graph can be very dense. Moreover, human ability to identify patterns is inversely proportional to the size and (visualization) complexity of graphs. According to Tufte [10] the Data-Ink ratio is a concept that is used to present essential data compared to the total amount of ink used in the entire drawing. Other reasons are also related to privacy protection (e.g., social network graphs). In such cases, it makes sense to replace the original graph with a subgraph or summary (e.g., clustering, supernodes), which removes unnecessary details about the original graph topology but retains the mental map of the user. To overcome this challenge, different approaches imply to hide the unnecessary information by displaying them on demand, skip the redundant information or to apply clustering algorithms in order to reduce the complexity of the graph on top of sophisticated graph drawing layouts.

1.2 Contributions and Outline of this Dissertation

The key contributions of this thesis are the following:

- Chapter 2, focuses on the problem of finding a feedback arc set. In a directed graph G, a feedback arc set is a set of edges whose removal leaves G acyclic. The minimum FAS problem, which is NP-hard, is important for visualizing directed graphs in hierarchical style [7]. In fact, the first step of both known frameworks for hierarchical graph drawing is to compute a minimum FAS [5, 6]. To this respect, we introduce a new heuristic algorithm for computing a minimum Feedback Arc Set in directed graphs. The new technique produces solutions that are better than the ones produced by the previously best known heuristics, and is based on computing the PageRank score of the nodes of the directed line graph of the input directed graph. Our experimental results show that the size of a FAS computed by our heuristic algorithm is typically about 50% smaller than the sizes obtained by the best previous heuristics. The results have been published in [11].

- Chapter 3, introduces a new approach to visualize directed graphs and their hierarchies that departs from the classical four-phase framework of *Sugiyama framework* and computes readable hierarchical visualizations that contain the complete reachability information of a graph. It also discusses algorithms that extend the path-based hierarchical drawing framework. Our algorithms run in $O(km)$ time, where k is the number of paths and m is the number of edges of the graph, and provide better upper bounds than the original path based framework: i.e., the height of the resulting drawings is equal to the length of the longest path of G, instead of $n-1$, where n is the number of nodes. Additionally, we extend this framework, by bundling and drawing all the edges of the DAG in $O(m+nlogn)$ time, using minimum extra width per path. We also provide some comparison to the well known hierarchical drawing framework of

Sugiyama framework, as a proof of concept. The experimental results show that the drawings produced by our algorithms have significantly lower number of bends and are much smaller in area than the ones produced by $OGDF$, which is based on the Sugiyama technique, but they have more crossings for sparse graphs. Since there are advantages (and disadvantages) to both frameworks, we also designed, performed and evaluated a task-based user study. The user evaluation shows that the performance of the participants is slightly better in PBF drawings than in $OGDF$ drawings and the participants prefer PBF in overall rating compared to $OGDF$. Hence, our technique offers an interesting alternative for drawing hierarchical graphs when we visualize hierarchical graphs, since we focus on showing important aspects of the graph such as critical paths, path transitive edges, and cross edges. The results have been published in [12].

- Chapter 4, presents a set of visualization algorithms (variants) that attempt to draw DAGs hierarchically with few bends and crossings, and by abstracting edges in order to improve the clarity of the drawings. Our algorithms reduce the visual complexity of the resulting drawings by (a) drawing the vertices of the graph in some vertical lines, and (b) by progressively abstracting some transitive edges thus showing only a subset of the edge set in the output drawing. Our algorithms are based on the concepts of the path and channel decomposition and focus on showing the existence of paths clearly. Our algorithms run in $O(km)$, where k is the number of paths/channels and m is the number of edges of the graph. The nodes of each path/channel (which can be user defined) are *vertically aligned*. The process of progressively abstracting the edges gives different visualization results, but they all have the same transitive closure as the input graph. We also present experimental results that show a very interesting interplay between bends, crossings, clarity of the drawings, and the abstraction of edges. The results have been published in [13].

- Chapter 5, considers the problem of answering reachability queries in directed acyclic graphs (DAGs), which is an operation required by many applications. Our approach is based on dominance drawings of DAGs, which are important in many research areas, including graph drawing, computational geometry, information visualization and very large databases. Toward this direction, we present efficient algorithms to construct and search a space-efficient data structure in the k-dimensional space. Our algorithms construct this data structure in O(km) time while it can be stored in O(kn) space. Any reachability query is answered in constant time, since no "falsely implied paths (fips)" are introduced. We also present experimental results, that show that the number of dimensions, k, in the solutions produced by our techniques is low. Additionally, we present a new method for constructing random DAGs with prespecified structure and density which is more suitable to DAGs and their applications. In this new model, graphs are randomly generated but they are based on a number of predefined but randomly created paths. The analysis of our experimental results reveals an interesting interplay between density and structure. The results have been published in [14].

- Chapter 6, introduces graph abstraction techniques for grouping nodes with similar characteristics that naturally decompose a graph based on the set of features (relationships) applied, using the information derived by the edges. Consequently, we present the notion of a Context-Aware Graph as a semantically enriched representation of the original graph that allows the depiction of relationship between the nodes, in order to have better visualization of complex networks. Our goal is to reduce the drawing complexity using the different semantics of the graphs that can be further used as a visualization aid (preprocessing step) for our hierarchical visualization graph framework.

Additionally, we efficiently use the context of the graph to reveal hidden knowledge and discover communities and patterns underneath the original network. As a use case, the proposed techniques are applied and evaluated successfully on our Hierarchical Drawing framework based on the context of reachability (transitivity). The results show that we manage to reduce the number of nodes and edges without losing important information about the graph, while we also reveal hidden patterns that could not be detected using the original graph.

1.3 The Roadmap

The main objective of this dissertation is to offer advanced techniques and algorithms for covering the needs of visualizing directed acyclic graphs (DAGs), while focusing on displaying the reachability information. Our target is to hide (some non-essential edges) information in order to make reachability between nodes easy to visualize by reducing the visual complexity, retain the mental map of the user and improve the clarity of the produced drawings. To this respect, we investigate various approaches towards this direction. In abstract, the structure of this dissertation is organized in the following way:

As a first step, we elaborate on the problem of computing the minimum Feedback Arc Set, since in a directed graph G, a FAS is a set of edges whose removal leaves G acyclic (DAGs). Next, we introduce a new approach to visualize DAGs that focuses on their reachability information. We present a detailed general-purpose hierarchical graph drawing framework and we further extend the hierarchical graph drawing framework of [5, 9]. Moreover, we design and implement various versions (variants) of this new drawing framework that attempt to draw DAGs hierarchically with few bends and crossings, and by abstracting edges in order to further improve the clarity of the drawings. Furthermore, we exploit reachability queries in DAGs based on dominance drawings. To this respect, we construct a data structure in the k-dimensional space that is based on Graph Dominance Drawing to answer such queries. Since our

proposed drawing framework is based on the concepts of path and channel decomposition, we also introduce a new graph model, which is based on a number of predefined paths. Finally, we address the challenges in interpreting the semantics of a graph that can be used as a graph visualization aid. More specifically, we investigate approaches regarding graph abstraction techniques for grouping nodes with similar characteristics that naturally decompose a graph based on the semantics applied. Consequently, we present the notion of Context-Aware Graph, as a semantically enrinched representation of the original graph in order to have better visualizations of complex networks. Additionally, we efficiently use the context of the graph to reveal hidden knowledge and discover communities and patterns underneath the original network. As a proof of concept, the proposed techniques are applied on our Hierarchical Drawing framework based on the context of reachability (transitivity) as a use case.

Chapter 2

A Heuristic algorithm for computing a minimum Feedback Arc Set in directed graphs

In a directed graph, G, a feedback arc set (FAS) is a set of edges whose removal leaves G acyclic. The minimum FAS problem is important for visualizing directed graphs in hierarchical style [4]. In fact, the first step of both known frameworks for hierarchical graph drawing is to compute a minimum FAS [5, 6]. Unfortunately, computing a minimum FAS is NP-hard and thus many heuristics have been presented in order to find a reasonably good solution. In this chapter we present a new heuristic that uses a different approach and produces FAS that contain about half the number of edges of the best known heuristics. However, it requires superlinear time, and hence it may not be suitable for very large graphs. Finding a minimum FAS has many additional applications beyond Graph Drawing, including misinformation removal, label propagation, and many application domains motivated by Social Network Analysis [15–17].

A feedback arc set of a directed graph $G = (V, E)$ is a subset of edges F of E such that removing the edges in F from E leaves G acyclic (no directed cycles). In other words, a FAS contains at least one edge from each cycle of G. In hierarchical drawing algorithms the edges in a FAS are not removed, but instead their direction is inverted. Following the terminology of [4], a set of edges whose reversal makes the digraph

acyclic is called a feedback set (FS). Notice that a FAS is not always a FS. However, it is easy to see that every minimal cardinality FAS is also a FS. Hence it follows that the minimum FS problem is as hard as the well studied minimum FAS problem which is known to be NP-hard [18, 19]. Clearly, any heuristic for solving the minimum FAS problem can be applied for solving the minimum FS problem, as discussed in [3, 4]. A novel exact method for computing the minimum FAS Problem was recently proposed in [20]. Their proposed method uses Integer Linear Programming and enumerates simple cycles in a lazy fashion by extending an incomplete cycle matrix iteratively. Their method succeeds in finding a minimum FAS for several (small/medium) graphs but, as it is expected, it fails to produce results for dense graphs, and even for sparse, larger graphs [20].

There have been many heuristics for solving the FAS problem due to the multitude of its applications. Two of the most important heuristics/techniques are due to Eades, Lin & Smyth [21] and Brandenburg & Hanauer [22]. The first is a greedy heuristic, that will be called *GreedyFAS*, whereas the second presents a set of heuristics based on sorting. Simpson, Srinivasan & Thomo published an experimental study for the FAS problem on very large graphs at web-scale (also called *webgraphs*) [23]. They implemented and compared many FAS heuristics. According to their study, the aforementioned are the most efficient heuristics, but only GreedyFAS is suitable to run on their extra large webgraphs.

In this chapter we present a new heuristic algorithm for computing a minimum FAS in directed graphs. The new technique produces solutions that are better than the ones produced by the best previous heuristics, sometimes even reducing the FAS size by more than 50%. It is based on computing the PageRank score of the nodes of a graph related to the input graph, and runs rather fast for graphs up to 4,000 nodes. However, it is slower than GreedyFAS for webgraphs.

2.1 Existing Algorithms

In this section we summarize and give a brief description of two important heuristics that currently give the best results for the FAS problem, according to the new experimental study of Simpson, Srinivasan & Thomo [23]. They implemented and compared many heuristics for FAS, and performed experiments on several large and very large webgraphs. Their results show that two of the known heuristic algorithms give the best results.

The first of the two heuristic algorithms that currently produce the best FAS size is called *GreedyFAS* and it is due to Eades, Lin & Smyth [21]. In [23] two different optimized implementations of GreedyFAS that run in $O(n + m)$ are presented and tested. These are the most efficient implementations in their study and are able to run even for their extra large webgraphs. The second algorithm is *SortFAS* of Brandenburg & Hanauer [22]. According to [23], SortFAS, as proposed runs in $O(n^3)$ time but Simpson et al. present an implementation that runs in $O(n^2)$ time.

We will present experimental results that show that our new heuristic algorithm performs better than both of them in terms of the size of the produced FAS. On the other hand, it takes more time than both of them for large graphs. However, for graphs that are typically used for visualization purposes, the running time is acceptable whereas the produced FAS size is about half.

2.1.1 GreedyFAS

The GreedyFAS algorithm was introduced by Eades, Lin & Smyth in 1993 [21]. It efficiently calculates an approximation to the FAS problem on a graph G. In order to understand the algorithm, we first discuss the *Linear Arrangement Problem (LA)*, which is an equivalent formulation to the FAS problem. The LA problem produces an ordering of the nodes of a graph G for which the number of arcs pointing backwards is minimum. The set of backwards arcs is a FAS since removing them from G leaves

the graph acyclic.

GreedyFAS calculates a feedback arc set of a graph G by first calculating a Linear Arrangement of G. More specifically, in each iteration, the algorithm removes all nodes of G that are sinks followed by all the nodes that are sources. A node is considered a source in a graph if it has no incoming edges. Likewise, a node is considered a sink if it has no outgoing edges. It then removes a node u for which $\delta(u) = d^+(u) - d^-(u)$ is a maximum, where $d^+(u)$ denotes the out-degree of u and $d^-(u)$ denotes the in-degree of u. The algorithm also makes use of two sequences of nodes s_1 and s_2. When any node u is removed from G then it is either prepended to s_2 if it's a sink, or appended to s_1 if it's not. The above steps are repeated until G is left with no nodes, then the sequence $s = s_1 s_2$ is returned as a linear arrangement for which the backward arcs make up a feedback arc set. For more details see [3, 4]. Using the implementations of [23], GreedyFAS runs very fast, in $O(n + m)$ time, and is suitable for their extra large webgraphs. The pseudocode for GreedyFAS, as described in [4] and [23], is presented in Algorithm 1.

Algorithm 1 GreedyFAS

Input: Directed graph $G = (V, E)$
Output: Linear Arrangement A
$s_1 \leftarrow \emptyset$, $s_2 \leftarrow \emptyset$
while $G \neq \emptyset$ **do**
    while G contains a sink **do**
        choose a sink u
        $s_2 \leftarrow u s_2$
        $G \leftarrow G \backslash u$
    while G contains a source **do**
        choose a source u
        $s_1 \leftarrow s_1 u$
        $G \leftarrow G \backslash u$
    choose a node u for which $\delta(u)$ is a maximum
    $s_1 \leftarrow s_1 u$
    $G \leftarrow G \backslash u$
return $s = s_1 s_2$

2.1.2 SortFAS

The SortFAS algorithm was introduced in 2011 by Brandenburg & Hanauer [22]. The algorithm is an extension of the KwikSortFAS heuristic by Ailon et al. [24], which is an approximation algorithm for the FAS problem on tournaments. With SortFAS, Brandenburg & Hanauer extended the above heuristic to work for general directed graphs. It uses the underlying idea that the nodes of a graph can be sorted into a desirable Linear Arrangement based on the number of back arcs induced.

SortFAS is equivalent to sorting by insertion for the linear arrangement problem. In the case of SortFAS, the nodes are processed in order of their ordering $(v_1 ... v_n)$. The algorithm goes through n iterations. In the the i-th iteration, node v_i is inserted into the linear arrangement in the best position based on the first $i-1$ nodes which are already placed. The best position is the one with the least number of back arcs induced by v_i. In case of a tie the leftmost position is taken. Using the implementation of [23], SortFAS runs in $O(n^2)$ time. The pseudocode for SortFAS, as described in [23], is presented in Algorithm 2.

Algorithm 2 SortFAS

```
Input: Linear arrangement A
for each node v in A do
    val ← 0, min ← 0, loc ← position of v
    for each position j from loc − 1 down to − do
        w ← node at position j
        if arc (v, w) exists then
            val ← val − 1
        else if arc (w, v) exists then
            val ← val + 1
        if val ≤ min then
            min ← val, loc ← j
    insert v at position loc
```

2.2 Efficient Computation of a Feedback Arc Set: An approach Using PageRank

Our approach is based on running the well known PageRank algorithm [25, 26] on the directed line digraph of the original directed graph. The *line graph* of an undirected graph G is another graph $L(G)$ that is constructed as follows: each edge in G corresponds to a node in $L(G)$ and for every two edges in G that are adjacent to a node v an edge is placed in $L(G)$ between the corresponding nodes. Clearly, the number of nodes of a line graph is m and the number of edges is proportional to the sum of squares of the degrees of the nodes in G, see [27]. If G is a directed graph, its *directed line graph* (or *line digraph*) $L(G)$ has one node for each edge of G. Two nodes representing directed edges incident upon v in G (one incoming into v, and one outgoing from v), called $L(u, v)$, and $L(v, w)$, are connected by a directed edge from $L(u, v)$ to $L(v, w)$ in $L(G)$. In other words, every edge in $L(G)$ represents a directed path in G of length two. Similarly, the number of nodes of a line digraph is m and the number of edges is proportional to $\sum_{u \in V}[d^+(u) \times d^-(u)])$. Hence, the size of $L(G)$ is $O(m + \sum_{u \in V}[d^+(u) \times d^-(u)])$.

Given a digraph $G = (V, E)$ our approach is to compute its line digraph, $L(G)$, run a number of iterations of PageRank on $L(G)$ and remove the node of highest PageRank in $L(G)$. Our experimental results indicate that PageRank values converge reasonably well within five iterations.

A digraph G is *strongly connected* if for every pair of vertices of G there is a cycle that contains them. If G is not strongly connected, it can be decomposed into its *strongly connected components (SCC)* in linear time [28]. An SCC of G is a subgraph that is strongly connected, and is maximal, in the sense that no additional edges or vertices of G can be included in the subgraph without breaking its property of being strongly connected. If each SCC is contracted to a single vertex, the resulting graph is a directed acyclic graph (DAG). It follows that feedback arcs can exist only

within some (SCC) of G. Hence we can apply this approach inside each SCC, using their corresponding line digraph, and remove the appropriate edges from each SCC. This approach will avoid performing several useless computations and thus reduce the running time of the algorithm.

2.2.1 Line Graph

In order to obtain the line digraph of G, we use a DFS-based approach. First, for each edge (u, v) of G, we create a node (u, v) in $L(G)$ and then run the following recursive procedure. For a node v, we mark it as visited and iterate through each one of its outgoing edges. For each outgoing edge (v, u) of v, we add an edge in $L(G)$ from the *prev* $L(G)$ node that was processed before the procedure's call to the node (v, u). Afterwards we call the same procedure for u if it's not visited with (v, u) as *prev*. If u is visited we add an edge from (v, u) to each one of $L(G)$'s nodes corresponding from u. Since this technique is based on DFS, the running time is $O(n+m+|L(G)|)$. The pseudocode for computing a line digraph is presented in Algorithm 3.

Algorithm 3 LineDigraph

```
Input: Digraph G = (V, E)
Output: Line Digraph L(G) of G
Create a line digraph L(G) with every edge of G as a node
v ← random node of G

procedure GetLineGraph(G, L(G), v, prev)
    mark v as visited
    for each edge e = (v, u) outgoing of v do
        z ← node of L(G) representing e
        create an edge in L(G) from prev to z          ▷ Given that prev is not nill
        if u is not visited then
            GetLineGraph(G, L(G), u, z)
        else
            for each node k in L(G) that originates from u do
                create an edge in L(G) from z to k
```

2.2.2 PageRank

PageRank was first introduced by Brin & Page in 1998 [25, 26]. It was developed in order to determine a measure of importance of web pages in a hyperlinked network of web pages. The basic idea is that PageRank will assign a score of importance to every node (web page) in the network. The underlying assumption is that important nodes are those that receive many "recommendations" (in-links) from other important nodes (web pages). In other words, it is a link analysis algorithm that assigns numerical scores to the nodes of a graph in order to measure the importance of each node in the graph. PageRank works by counting the number and quality/importance of edges pointing to a node and then estimate the importance of that node. We use a similar approach in order to determine the importance of edges in a directed graph. The underlying assumption of our technique is that the number of cycles that contain a specific edge e will be reflected in PageRank score of e. Thus the removal of edges with high PageRank score is likely to break the most cycles in the graph.

Given a graph with n nodes and m edges, PageRank starts by assigning an initial score of $1/n$ to all the nodes of a graph. Then for a predefined number of iterations each node divides its current score equally amongst its outgoing edges and then passes these values to the nodes it is pointing to. If a node has no outgoing links then it keeps its score to itself. Afterwards, each node updates its new score to be the sum of the incoming values. It is obvious that after enough iterations all PageRank values will inevitably gather in the sinks of the graph. In use cases where that is a problem a damping factor is used, where each node gets a percentage of its designated score and the rest gets passed to all other nodes of the graph. For our use case we have no need for this damping factor as we want the scores of the nodes to truly reflect their importance. The number of iterations depends on the size and structure of a graph. We found that for small and medium graphs, which is the case in the scenario for graph visualization, about five iterations were enough for the scores of the nodes

to converge. Depending on the implementation, PageRank can run in $O(k(n+m))$ time, where k is the number of iterations. The pseudocode for PageRank is presented in Algorithm 4.

Algorithm 4 PageRank

Input: Digraph $G = (V, E)$, number of iterations k
Output: PageRank scores of G
for each node v in G **do**
    $PR(v) \leftarrow \frac{1}{|V|}$
for k iterations **do**
    for each node v in G **do**
        $PR(v) \leftarrow \sum_{u \in in(v)} \frac{PR_{old}(u)}{|out(u)|}$
return PR

2.2.3 PageRankFAS

The proposed algorithm is based on the concepts of PageRank and Line Digraphs. The idea behind *PageRankFAS* is that we can score the edges of G based on their involvement in cycles: For each strongly connected component $(s_1, s_2, ..., s_j)$ of G, it computes the line digraph $L(s_i)$ of the i-th strongly connected component, to transform edges to nodes; next it runs the PageRank algorithm on $L(s_i)$ to obtain a score for each edge of s_i in G.

We observed that the nodes of the line digraphs with the highest PageRank score correspond to edges that are involved in the most cycles of G. We also observed that the nodes of the line digraphs with lower score correspond to edges of G with low involvement in cycles. Using this knowledge, we run PageRankFAS for a number of iterations. In each iteration, we use PageRank to calculate the node scores of each $L(s_i)$ and remove the node(s) with the highest PageRank score, also removing the corresponding edge(s) from G. We repeat this process until G becomes acyclic. The pseudocode is presented in Algorithm 5.

Algorithm 5 PageRankFAS

Input: Digraph $G = (V, E)$
Output: Feedback Arc Set of G
$fas \leftarrow \emptyset$
while G has cycles **do**
  Let $(s_1, s_2, ..., s_j)$ be the strongly connected components of G
  for each strongly connected component s_i **do**
    Create a line digraph $L(s_i)$ with every edge of s_i as a node
    $v \leftarrow$ random node of s_i
    GetLineGraph$(s_i, L(s_i), v, nill)$
    PageRank$(L(s_i))$
    $u \leftarrow$ node of $L(s_i)$ with highest PageRank value
    $e \leftarrow$ edge of G corresponding to u
    Add e to fas
    Remove e from G
return fas

2.3 Experiments and Discussion

Here, we report the experimental results and describe some details of our setup. All of our algorithms are implemented in Java 8 using the WebGraph framework [29, 30] and tested on a single machine with Apple's M1 processor, 8GB of RAM and running macOS Monterey 12.

Datasets: In order to evaluate *PageRankFAS*, we used four different datasets:

1. Generated random graphs with 100, 200, 400, 1000, 2000, 4000 nodes and an average out-degree of 1.5, 3 and 5 each.

2. Three directed graphs from the datasets in graphdrawing.org, suitably modified in order to contain cycles (since the originals are DAGs).

3. Generated random graphs with 50, 100 and 150 nodes and average out-degrees of 1.5, 3, 5, 8, 10 and 15 each.

4. Two webgraphs from the Laboratory of Web Algorithmics[1], also used in [23].

[1] https://law.di.unimi.it/datasets.php

We randomly generate a total of 36 graphs using a predefined number of nodes, average out-degree and back edge percentage, and we repeat the process 10 times. The total number of edges is based on the number of nodes and average out-degree. The edges are selected with uniform probability by taking advantage of JAVA random numbers generators. Next, we add the predefined percentage of edges as back edges and the rest as forward edges. Finally we shuffle the node IDs in order to prevent any bias in the traversal of the graph. By construction, this model has the advantage that we know in advance an upper bound to the FAS size, since the number of randomly created back edges divided by the total number of edges, is an upper-bound to the size of a minimum FAS. Finally, in order to obtain more reliable results, for each case we run the three algorithms on 10 created graphs and report the average numbers. This smooths out several points in our curves.

2.3.1 FAS with Respect to the Number of Nodes

The first set of experiments gives us an idea of how PageRankFAS performs on graphs, with varying number of nodes in comparison to the other two algorithms. It is noteworthy that in most cases the FAS found by PageRankFAS is less than 50% of the FAS found by GreedyFAS and SortFAS. As a matter of fact, for large visualization graphs with 4,000 nodes and 12,000 edges the reduction in the FAS size is almost 55% with respect to the FAS produced by GreedyFAS. The execution time taken by PageRankFAS is less than one second for graphs up to 1,000 nodes, which is similar to the time of the other two heuristics. Figure 2.1 shows a visual comparison of the FAS size of PageRankFAS with respect to GreedyFAS on a graph with 1,000 nodes. For the larger graphs, even up to 4,000 nodes the time required is less than 8 seconds, whereas, the other heuristics run in about 1-2 seconds. Examples of such graph sizes can be found in a number of application domains related, but not limited, to social networks, traffic networks, citation networks and large scale chemical engineering systems.

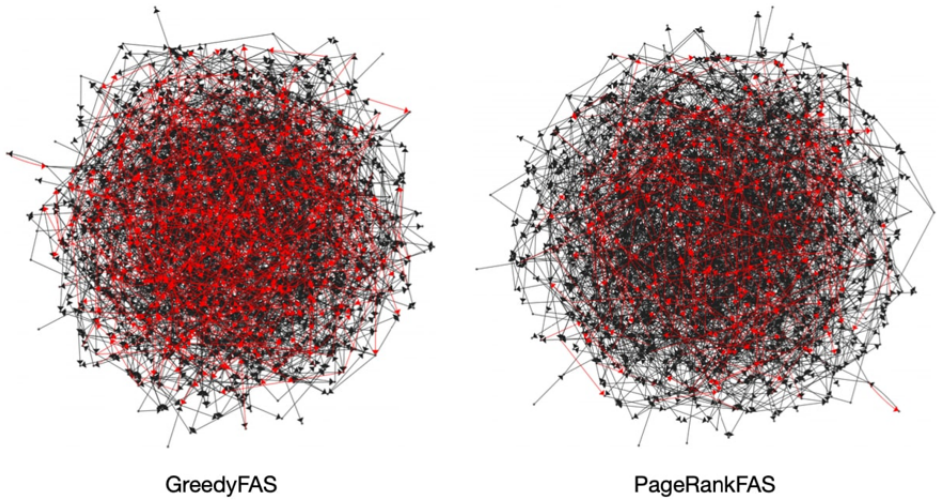


Figure 2.1: A visual comparison (a different view) of the FAS size of PageRankFAS with respect to GreedyFAS, on a graph with 1,000 nodes. The FAS is shown as red edges.

The results of this experiment are shown in Figure 2.5. It is interesting to note that the performance of SortFAS is better than the performance of GreedyFAS as the graphs become denser, and in fact, SortFAS actually out-performs GreedyFAS when the graphs have an average out-degree 5 and above, see Figure 2.5(c).

2.3.2 FAS with Respect to The Number of Back Edges

The second type of experiments make use of three graphs from graphdrawing.org. Since these graphs are directed acyclic, we randomly added back edges in different percentages of the total number of edges. We did this in a controlled manner in order to know in advance an upper bound of FAS. PageRankFAS gave by far the best FAS results and GreedyFAS also produced FAS with sizes mostly below 10%. SortFAS was not competitive in this dataset. The results are shown in Figure 2.9. The execution time taken by PageRankFAS is well below 0.15 of a second for all graphs, which is similar to the other two heuristics.

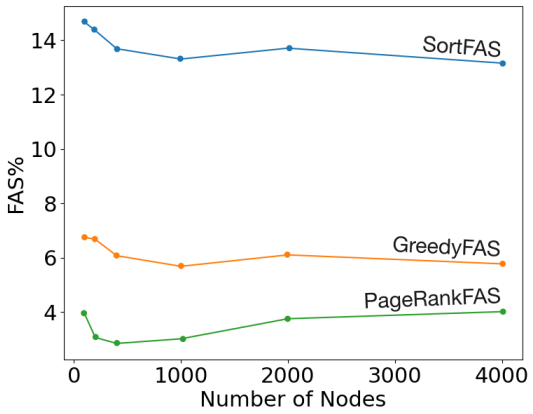


Figure 2.2: **(a)** Graphs with average out-degree 1.5

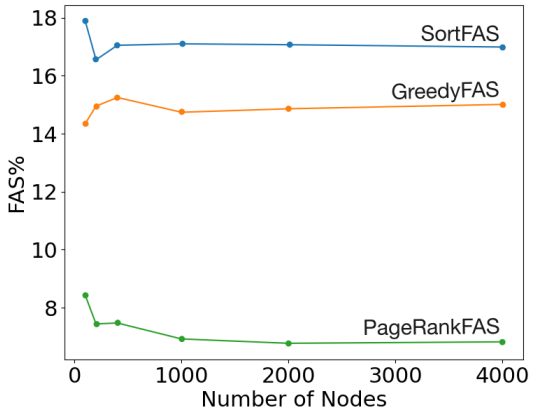


Figure 2.3: **(b)** Graphs with average out-degree 3

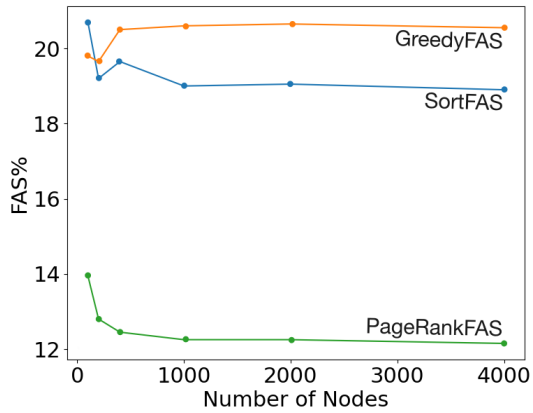


Figure 2.4: **(c)** Graphs with average out-degree 5

Figure 2.5: **(a)-(c)** FAS percentage for graphs with increasing number of nodes and three different average out-degrees.

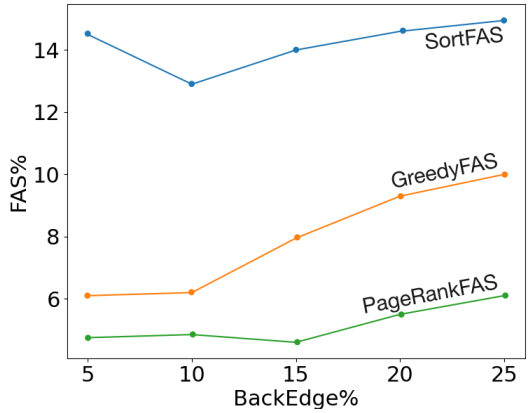


Figure 2.6: **(a)** Graph with 50 nodes and 75 edges before modification

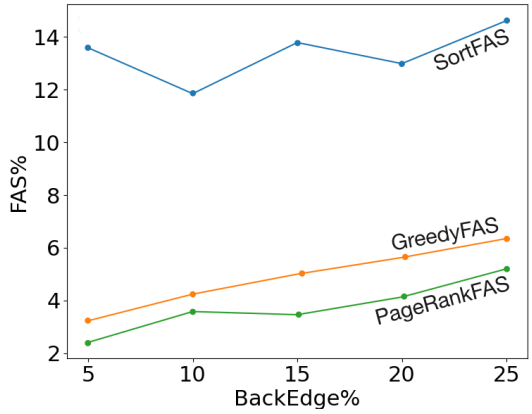


Figure 2.7: **(b)** Graph with 75 nodes and 86 edges before modification

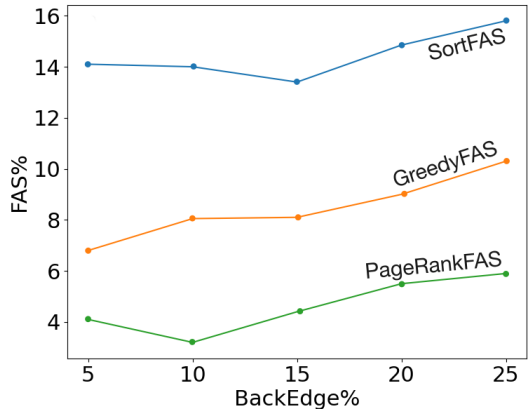


Figure 2.8: **(c)** Graph with 99 nodes and 154 edges before modification

Figure 2.9: **(a)-(c)** FAS percentage for 3 types of graphs from graphdrawing.org and for various numbers of back edges.

2.3.3 FAS with Respect to the Average Out-Degree

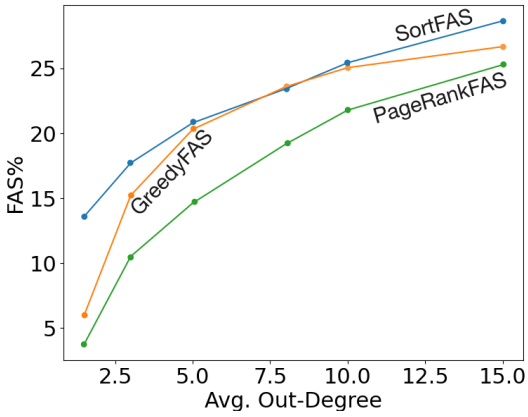
Motivated by the results shown in Figure 2.5(c) we decided to investigate the correlation between the density of a graph and its potential FAS percentage. In this experiment, we created 18 different graphs, six of them with 50 nodes, six with 100 nodes and six with 150 nodes as follows: For each node size (i.e., 50, 100, 150) six graphs with average out-degrees 1.5, 3, 5, 8, 10 and 15. Again, as with our previous experiments, the results reported here are the averages of 10 runs in order to compensate for the randomness of each graph and to get smoother curves. The results of this experiment are shown in Figure 2.10.

The results of PageRankFAS are consistently better than the results of GreedyFAS and SortFAS for all graphs. The results of GreedyFAS and SortFAS are very close to each other, for the graphs with 50 nodes. Notice however that, SortFAS outperforms GreedyFAS when the number of nodes exceeds 100 and the average out-degree exceeds five. This is aligned with the results shown in Figure 2.5(c). Furthermore, as expected, when the average out-degree increases the FAS size clearly increases. Consequently, all techniques seem to converge at higher percentages of FAS size. Again, PageRankFAS runs in a small fraction of a second for all graphs, which is similar to the running times of the other two heuristics.

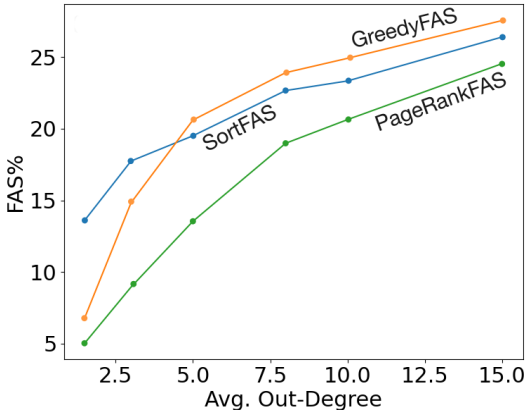
2.3.4 PageRankFAS on Webgraphs

The experiments reported in [23] use large and extra large benchmark webgraphs. Their smaller benchmarks are *wordassociation-2011* (with 10,617 nodes, 72,172 edges, which implies an average degree 6.80) and *enron* (with 69,244 nodes, 276,143 edges, which implies an average degree 3.86).

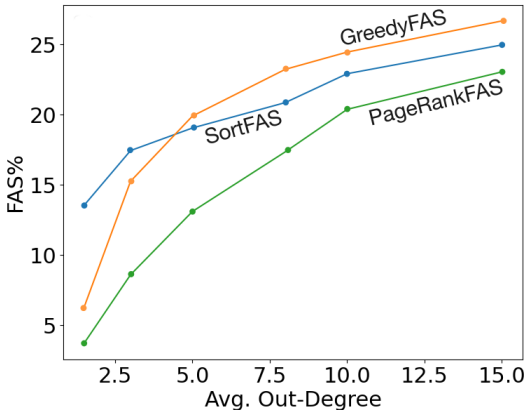
The authors report that the sizes of a FAS found by GreedyFAS and SortFAS for wordassociation-2011 are 18.89% and 20.17%, respectively [23]. We ran PageRankFAS for wordassociation-2011 and obtained a FAS of size 14.85%. Similarly, for webgraph enron they report a FAS of 12.54% and 14.16% respectively. We ran PageR-



(a) Graphs with 50 nodes

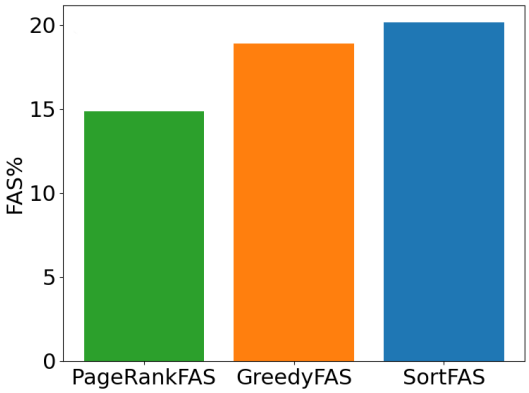


(b) Graphs with 100 nodes

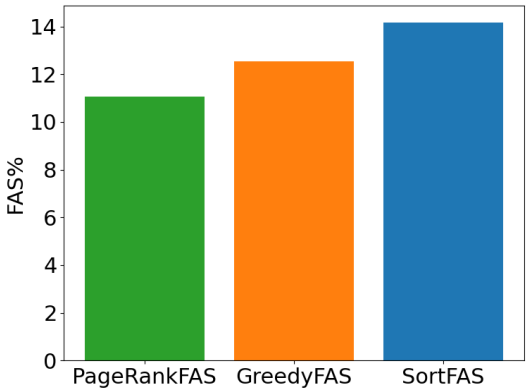


(c) Graphs with 150 nodes

Figure 2.10: FAS percentage depending on the average out-degree of three different types of graphs.



(a) wordassociation-2011



(b) enron

Figure 2.11: FAS percentage on two webgraphs.

ankFAS on webgraph enron and obtained a FAS of size 11.05%. The results are shown in Figure 2.11. As expected, and consistent with our experimental observations of the previous subsections, the FAS size of the denser webgraph (wordassociation-2011) is larger than the FAS size of the sparser graph (enron), as computed by all heuristics.

Unfortunately, the required execution time of *PageRankFAS* does not allow us to test it on the larger webgraphs used in [23]. However, it is interesting that there exists a FAS of smaller size for these large graphs, which, to the best of our knowledge, was not known before.

Discussion: In this chapter we presented a heuristic algorithm for computing a FAS of minimum size based on PageRank. Our experimental results show that the size of a FAS computed by *PageRankFAS* algorithm is typically about 50% smaller than the sizes obtained by the best previous heuristics. Our algorithm is more time consuming, but it's running time is reasonable for graphs up to 4,000 nodes. For smaller graphs, up to 1,000 nodes, the execution time is well below one second, which is similar to the running times of the other two heuristics. Therefore, this is acceptable for graph drawing applications. An interesting side result is that we found out that the FAS-size of two large graphs is significantly less than it was known before. Since it is NP-hard to compute the minimum FAS, the optimum solution for these webgraphs is unknown. Hence, we do not know how close our solutions are to the optimum. It would be interesting to investigate techniques to speedup *PageRankFAS* in order to make it more applicable to larger webgraphs.

Chapter 3

Improvements on the visualization aesthetics of the proposed hierarchical drawing framework

Hierarchical graphs are very important for many applications in several areas of research and business because they often represent hierarchical relationships between objects in a structure. They are directed (often acyclic) graphs and their visualization has received significant attention recently [1–3]. An experimental study of four algorithms specifically designed for (Directed Acyclic Graphs) DAGs was presented in [31]. DAGs are often used to describe processes containing some long paths, such as in PERT applications see for example [7, 8]. The paths can be either application based, e.g. critical paths, user defined, or automatically generated paths. A new framework to visualize directed graphs and their hierarchies was introduced in [5, 9]. It is based on a path/channel decomposition of the graph and is called *(Path-Based Framework or PBF)*. It computes readable hierarchical visualizations in two phases by "hiding" (*abstracting*) some selected edges, while maintaining the complete reachability information of a graph. However, these drawings are not satisfactory to users that need to visualize the whole graph.

In this chapter we extend the hierarchical graph drawing framework of [5, 9] in two directions: a) we draw all edges of the graph and use extensive edge bundling, and b) we minimize the height of the drawing using a compaction technique. To reduce

the width we apply algorithms similar to task scheduling. The total time required for these extensions is $O(m + n \log n)$, where m is the number of edges and n the number of nodes of a graph G. In this framework, the edges of G are naturally split into three categories: *path edges, path transitive edges, and cross edges*. Path edges connect consecutive vertices in the same path. Path transitive edges connect non-consecutive vertices in the same path. Cross edges connect vertices that belong to different paths.

The path-based framework departs from the typical Sugiyama Framework [32] and it consists of two phases: (a) Cycle Removal, (b) the path/channel decomposition and hierarchical drawing step. It is based on the idea of partitioning the vertices of a graph into node disjoint paths/channels, where in a channel consecutive nodes are connected by a path but not necessarily connected by an edge. In the rest, we only use the term "path" but of course our algorithms work also for "channels" The vertices in each path are drawn vertically aligned on some x-coordinate; next the edges between vertices that belong to different paths are drawn. Note that there are several algorithms that compute a path decomposition of minimum cardinality in polynomial time [33–36].

The Sugiyama Framework has been extensively used in practice, as manifested by the fact that various systems are using it to implement hierarchical drawing techniques. The comparative study of [31] concluded that the Sugiyama-style algorithms performed better in most of the metrics. For more recent information regarding this framework see [3]. Commercial software such as the Tom Sawyer Software TS Perspectives [37] and yWorks [38], use this framework in order to offer automatic visualizations of directed graphs. Even though it is very popular, the Sugiyama Framework has several limitations: as discussed bellow, most problems and subproblems that are used to optimize the results in various steps of each phase have turned out to be NP-hard. The overall time complexity of this framework (depending upon implementation) can be as high as $O((nm)^2)$, or even higher if one chooses algorithms that require exponential time. Another important limitation of this framework is

the fact that heuristic solutions and decisions that are made during previous phases (e.g., crossing reduction) will influence severely the results obtained in later phases. Nevertheless, previous decisions cannot be changed in order to obtain better results.

By contrast, in the framework of [5] most problems of the second phase can be solved in polynomial time. If a path decomposition contains k paths, the number of bends introduced is at most $O(kn)$ and the required area is at most $O(kn)$. Edges between non consecutive vertices in a path (the *path transitive edges*), are not drawn in the framework of [5]. Hence, users that need to visualize all the edges of a given graph are not satisfied by these drawings.

We present experimental results comparing drawings obtained by the extended-PBF to the drawings obtained by running the hierarchical drawing module of OGDF [39], which is based on the Sugiyama Framework, and is a quite active research software that implements this framework. The results show that PBF runs much faster, has better area and less bends, but OGDF has less crossings, especially for sparse graphs.

Since the usual metrics like bends, area, and crossings did not lead to concrete conclusions and the two frameworks produce vastly different drawings, we decided to perform a user study between these two drawing frameworks, in order to obtain feedback from users using these drawings. The users had to perform a set of tasks on the drawings of some DAGs. The tasks include determining if two given vertices are connected, finding the length of a shortest path, and determining if some vertices are successors of a given vertex. The users' answers were correct above 90% for PBF and above 84% for the Sugiyama Framework (OGDF). The users were also asked to express their preference, in terms of clarity and readability, between the two frameworks. 58.3% of the users showed a clear preference to using drawings produced by PBF. Hence, this technique offers an interesting alternative to drawing hierarchical graphs, especially if there are user defined paths that need to be clearly visualized. A detailed analysis of the user study and the experimental results is presented in next sections.

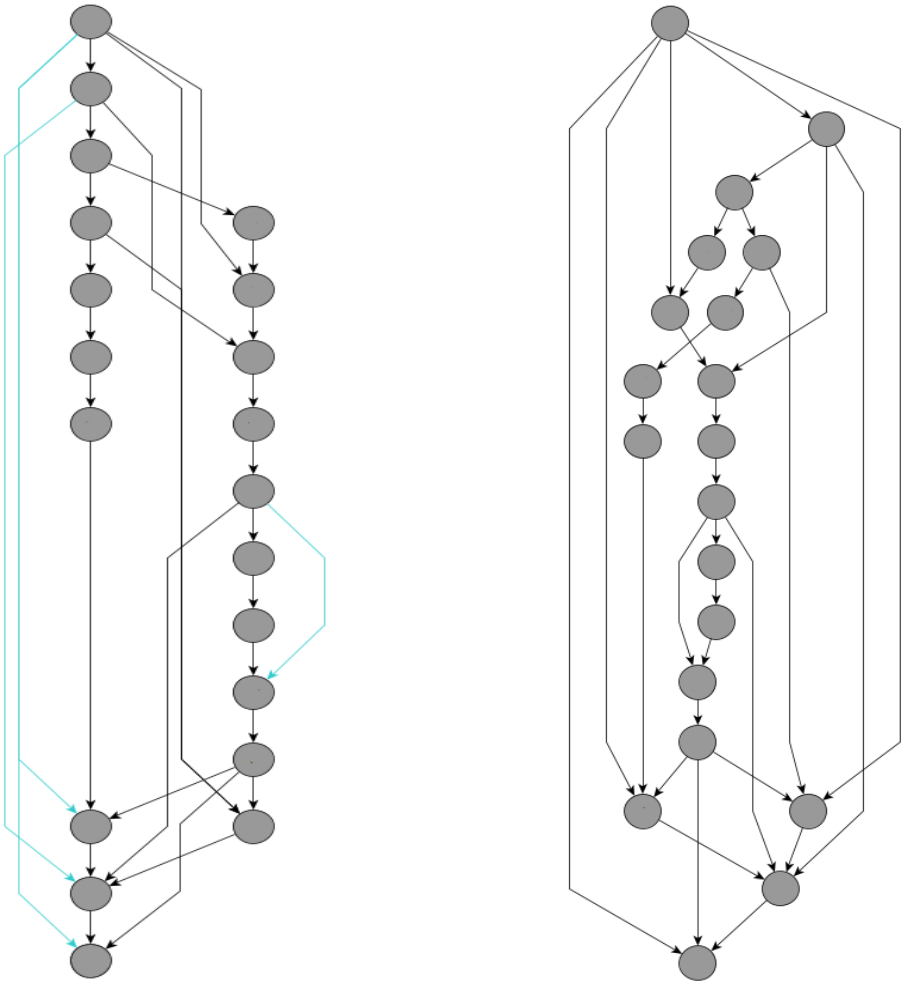


Figure 3.1: Example of a DAG G drawn by our proposed framework (left). Same DAG drawn by the Sugiyama framework as implemented in OGDF (right).

3.1 Overview of the Two Frameworks

The two hierarchical drawings shown in Figure 4.1 demonstrate the significant differences between the two frameworks: Part (a) shows a drawing of G computed by our algorithms that customize PBF. Part (b) shows the drawing of G computed by OGDF. The graph consists of 20 nodes and 31 edges. The drawing computed by our algorithms has 12 crossings, 18 bends, width 10, height 15, and area 150. On the other hand, OGDF computes a drawing that has 5 crossings, 22 bends, width 18, height 15 and area 270. Clearly, the two frameworks produce vastly different drawings with their own advantages and disadvantages.

The original Path Based Hierarchical Drawing Framework follows an approach to visualize directed acyclic graphs that hides some edges and focuses on maintaining their reachability information [5]. This framework is based on the idea of partitioning the vertices of the graph G into (a minimum number of) *channels/paths*, that we call *channel/path decomposition* of G, which can be computed in polynomial time. Therefore, it is orthogonal to the Sugiyama framework in the sense that it is a vertical decomposition of G into (vertical) paths/channels. Thus, most resulting problems are *vertically contained*, which makes them simpler, and reduces their time complexity. This framework does not need to introduce any dummy vertices and keeps the vertices of a path *vertically aligned*, which is important for specific applications (such as visualizing critical paths in PERT diagrams [7]). By contrast, the Sugiyama framework performs a horizontal decomposition of a graph, even though the final result is a vertical (hierarchical) visualization. Let $S_p = \{P_1, ..., P_k\}$ be a path decomposition of G such that every vertex $v \in V$ belongs to exactly one of the paths of S_p. Any path decomposition naturally splits the edges of G into: (a) *path edges* (b) *cross edges* and (c) *path transitive edges*. Given any S_p the main algorithm of [5], draws the vertices of each path P_i *vertically aligned* on some x-coordinate depending on the order of path P_i. The y-coordinate of each vertex is equal to its order in any topological sorting of G. Hence the height of the resulting drawing is $n - 1$.

It also important to highlight that the Path-Based Framework works for any given path decomposition. Therefore, it can be used in order to draw graphs with user-defined or application-defined paths, as is the case in many applications, see for example [7, 8]. If one desires automatically generated paths, there are several algorithms that compute a path decomposition of minimum cardinality in polynomial time [33–36]. Since certain critical paths are important for many applications, it is extremely important to produce clear drawings where all such paths are vertically aligned, see [5]. For the rest of this chapter, we assume that a path decomposition of G is given as part of the input to the algorithm.

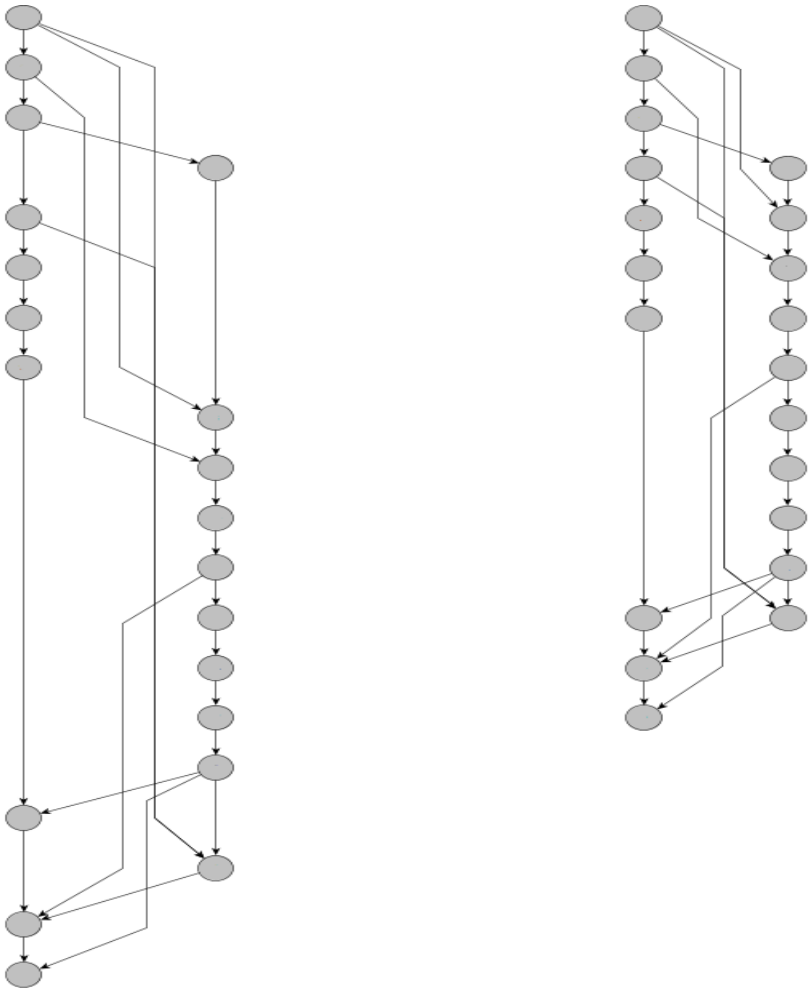
OGDF is a self-contained C++ library of graph algorithms, in particular for (but not restricted to) automatic graph drawing. The hierarchical drawing implementation of the Sugiyama Framework in OGDF is implemented following [40, 41] and it uses the following default choices: For the first phase of Sugiyama, it uses the *LongestPathRanking* (to assign vertices into layers) which implements the well-known longest-path ranking algorithm. Next, it performs crossing minimization by using the *BarycenterHeuristic*. This module performs two-layer crossing minimization and is applied during the top-down and bottom-up traversals [39]. The crossing minimization is repeated 15 times, and it keeps the best. Finally, the final coordinates (drawing) are computed with *FastHierarchyLayout* layout of OGDF.

3.2 Computing Compact and Bundled Drawings

We present an extension of the framework of [5] by (a) compacting the drawing in the vertical direction, and (b) drawing the path transitive edges that were not drawn in [5]. This approach naturally splits the edges of G into three categories, *path edges*, *cross edges*, and *path transitive edges*. This clearly adds new possibilities to the understanding of the user and allows a system to show the different edge categories separately, without altering the user's mental map.

3.2.1 Compaction

Let $G = (V, E)$ be a DAG with n vertices and m edges. Following the framework of [5, 9] the vertices of V are placed in a unique y-coordinate, which is specified by a topological sorting. Let T be the list of vertices of V in ascending order based on their y-coordinates. We start from the bottom and visit each vertex in T in ascending order. For every vertex v in this order we assign a new y-coordinate, $y(v)$, following a simple rule that compacts the height of the drawing: "If v has no incoming edges then we set its $y(v)$ equal to 0, else we set $y(v)$ equal to $a + 1$, where a is the *highest* y-coordinate of the vertices that have edges incoming into v."



(a) (b)

Figure 3.2: A DAG G drawn without its path transitive edges: (a) drawing Γ_1 is computed by Algorithm PBH, (b) drawing Γ_2 is the output after compaction and edge bundling.

Algorithm 6 takes as input a DAG G, and a path based hierarchical drawing Γ_1 of G computed by Algorithm PBH and it produces as output a new, compacted, path based hierarchical drawing Γ_2 with height L, where L is the length of a longest path in G. Clearly this simple algorithm can be implemented in $O(n+m)$ time.

Algorithm 6 Compaction(G, Γ_1)
Input: A DAG $G = (V, E)$, and a path based hierarchical drawing Γ_1 of G computed by Algorithm PBH
Output: A compacted path based hierarchical drawing Γ_2 with height L, where L is the length of a longest path in G.

1: **For** each $v \in G$:

- Let E_v be the set of incoming edges, $e = (w, v)$, into v:
  - *a.* **if** $E_v = \emptyset$ **then**:
    - $y(v)$=0
  - *b.* **else**:
    - $y(v)$=max{y-coordinates of vertices w with $(w, v) \in E_v$} + 1

Figure 3.2 shows an example of two hierarchical drawings of the same graph: Γ_1 is before compaction and Γ_2 is after compaction. The produced drawings, have the following simple properties:

Property 1. Two vertices of the same path are assigned distinct y-coordinates.

Property 2. For every vertex v with $y(v) \neq 0$, there is an incoming edge into v that starts from a vertex w such that $y(v) = y(w) + 1$.

Based on the properties above the height of the compacted drawing of the graph G is at most L and it can be computed in $O(n+m)$ time.

3.2.2 Drawing and Bundling the Path Transitive Edges

An important aspect of our work is the preservation of the mental map of the user that can be expressed, in part, by the reachability information of a DAG. Recall that path transitive edges are not drawn by the framework of [5, 9]. In this subsection we

Figure 3.3: Bundling of path transitive edges from left to right: (i) incoming edges into the last vertex of the path, (ii) bundling the incoming edges, (iii) outgoing edges from the first vertex of the path, (iv) bundling the outgoing edges.

show how to bundle and draw these edges while preserving the user's mental map of the previous drawing. Additionally, one may interact with the drawings by hiding the path transitive edges at the click of a button without changing the user's mental map of the complete drawing. We describe an algorithm that bundles and draws the path transitive edges using the minimum extra width (minimum extra number of columns) for each (decomposition) path as shown in Figure 3.3. The steps of the algorithm are briefly described as follows:

1. For every vertex of each decomposition path we compute the indegree and outdegree based only on path transitive edges.

2. If all indegrees and outdegrees are zero the algorithm is over, if not, we select a vertex v with the highest indegree or outdegree and we bundle all the incoming or outgoing edges of v, respectively. These bundled edges are represented by an *interval* with starting and finishing points, the lowest and highest y-coordinates of the vertices, respectively.

3. Next, we insert each interval on the left side of the path on the first available column such that the interval does not overlap with another interval.

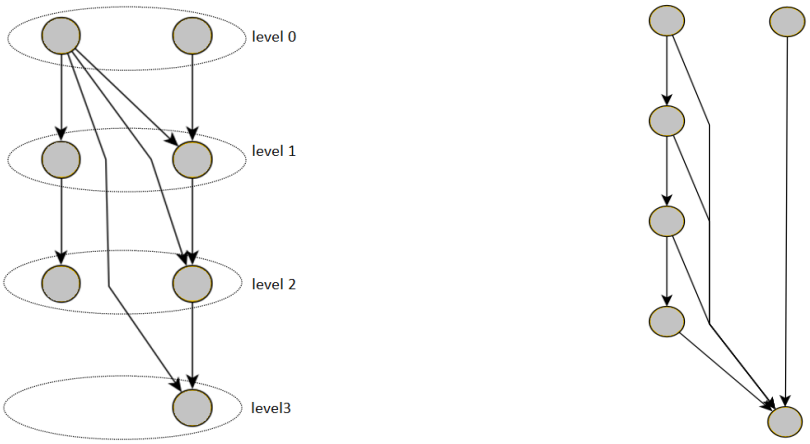
4. We remove these edges from the set of path transitive edges, update the indegree and outdegree of the vertices and repeat the selection process.

5. The intervals of the rightmost path, are inserted on the right side of the path in order to avoid potential crossings with cross edges.

6. A final, post-processing step can be applied because some crossings between intervals/bundled edges can be removed by changing the order of the columns containing them.

The above algorithm can be implemented to run in $O(m+n \log n)$ time by handling the updates of the indegrees and outdegrees carefully, and placing the appropriate intervals in a (Max Heap) Priority Queue. As expected, the fact that we draw the path transitive edges increases the number of bends, crossings, and area, with respect to not drawing them.

For each decomposition path, suppose we have a set of b intervals such that each interval I has a start point, s_I, and a finish point f_I. The starting point is the position of the vertex of the interval with the lowest y-coordinate. Similarly, the finish point f_I is the position of the node of the interval with the highest y-coordinate. We follow



(a) Placing bends on cross edges (b) Bundling of cross edges

Figure 3.4: Examples of bends and bundling of cross edges with a common end node.

a greedy approach in order to minimize the width (number of columns) for placing the bundled edges. The approach is similar to Task Scheduling [42], for placing the intervals. It uses the optimum number of columns and runs in $O(b \log b)$ time, for each path with b intervals. Since the sum of all b's for all paths in a path decomposition is at most n we conclude that the algorithm runs in $O(n \log n)$ time. For details and proof of correctness see [42].

3.2.3 Drawing and Bundling the Cross Edges

Cross edges connect vertices that belong to different paths. The number of bends of every cross edge depends on the vertical distance of its incident nodes. An example is shown in Figure 3.4a. Each cross edge $(u1, u2)$ has:

1. Two bends if the vertical distance between $u1$ and $u2$ is more than two.
2. One bend if the vertical distance between $u1$ and $u2$ is two.
3. The edge is a straight line segment (no bend) if the vertical distance between $u1$ and $u2$ is one.

We bundle all incoming cross edges for each vertex (except those with one unit of vertical distance from the target). We can place the bundled cross (Figure 3.4a) edges

between the paths/channels using the same technique we used for path transitive edges, using a technique that relies on task scheduling as we described above. Figure 3.4b shows an example of bundled cross edges.

3.3 Experimental Evaluation and User Study

In this section we present experimental results obtained by the extended path-based framework and we compare them with the respective experimental results obtained by running the hierarchical drawing module of OGDF, which is based on the Sugiyama Framework. In order to evaluate the performance, we used the following standard metrics:

- **Number of crossings.**
- **Number of bends.**
- **Width of the drawing:** The total number of distinct x coordinates that are used by the framework.
- **Height of the drawing:** The total number of distinct y coordinates that are used by the framework.
- **Area of the drawing:** The area of the enclosing rectangle.

Based on these metrics, we conducted a number of experiments to evaluate the performance of the two different hierarchical frameworks using a dataset of 20 randomly generated DAGs. Additionally, the metrics of PBF could vary depending on the path/channel decomposition algorithm we use and the ordering of the columns.

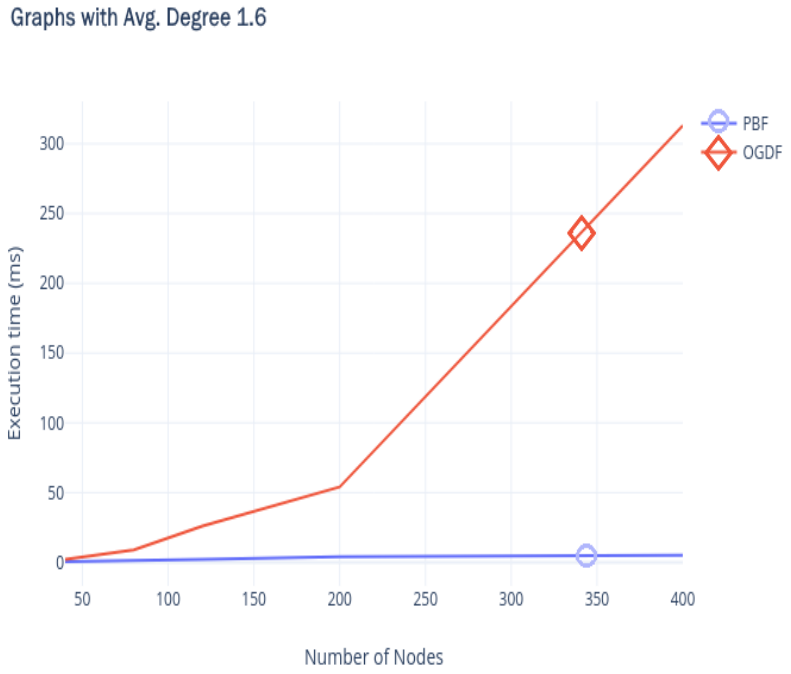
In general, our experiments showed that PBF produces readable drawings. Additionally, it clearly partitions the edges into three distinct categories, and vertically aligns certain paths, which may be user/application defined. This may be important in certain applications. The results showed that our approach differs from the

Sugiyama Framework completely, since it examines the graph vertically. The extended PBF performs bundling very efficiently and computes the optimal height of the graph. In most cases, the drawings based on PBF need less area than OGDF and contain fewer bends. On the other hand, OGDF generally has fewer crossings than PBF. This is expected since OGDF places a major computational effort into the crossing minimization step, whereas PBF does not perform any crossing minimization. Figure 3.5 shows that the time for PBF grows linearly in contrast to ODFG where it's time complexity seems to be cubic. For all the reasons described above this approach seems to be an interesting alternative to the Sugiyama style hierarchical drawings.

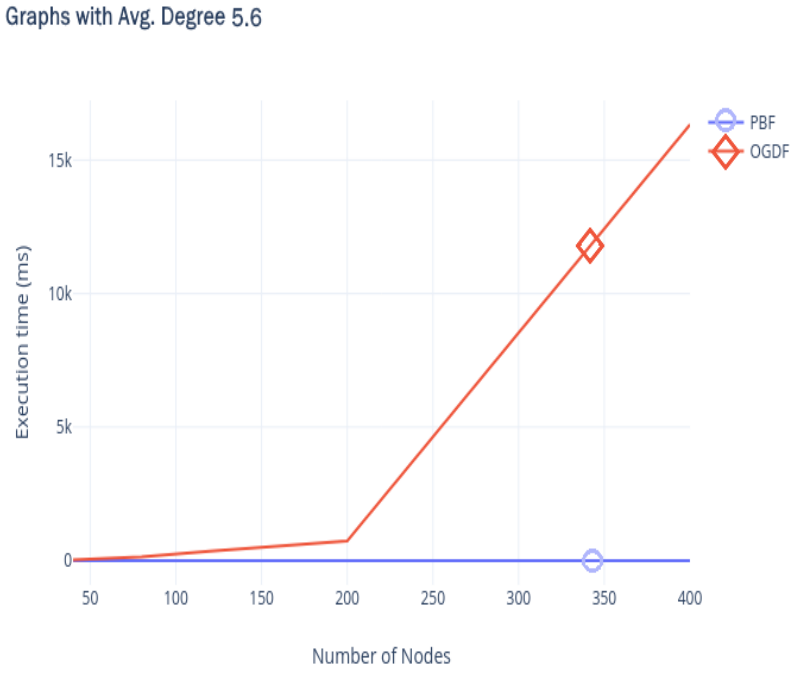
Additionally, we conducted another series of experiments in order to validate this statement further. To this respect, we used the benchmarks found at www.graphdrawing.org. The archive consists of graphs with 10 to 100 nodes with average degree about 1.6. The results of these additional experiments are similar to the results reported above and highlight the fact that the two frameworks focus on different aspects of the graphs and produce vastly different drawings. The new results reinforce our initial consideration that comparing quantitative metrics alone does not lead to a concrete conclusion. Hence, we decided to perform a user study in order to evaluate the readability and clarity of PBF comparing it with the Sugiyama framework from the perspective of a user.

3.3.1 User Study

Evaluation of visualizations [43] and analysing data is a difficult research challenge [44], [45]. Motivated by the work of [46], we design an evaluation with users, with a set of tasks that focus on revealing any usability issues. To this respect, we choose a set of "cognitive task" [47] where we evaluate the two systems based on attributes of clarity and readability as expressed by the reachability information within a drawing. Moreover, in order to design our evaluation we took into account the various restrictions



(a) Graphs with average degree 1.6



(b) Graphs with average degree 5.6

Figure 3.5: Execution time of PBF and $OGDF$ on various graphs.

of [47] where authors highlighted the basic guidelines toward to that direction. In our case, the participants had to choose an answer among "Yes", "No", or "Do Not Know" while also there was no time limit, although participants were expected to answer "Do Not Know" if a question was too difficult or too time-consuming to answer. The purpose of this assumption was to focus on tasks that can be trivial also to non expert users in order to detect and usability issue.

Users. We recruited 72 participants. In order to have more accurate and sophisticated results, we selected an audience that was familiar with graph theory and graph drawing styles. More specifically, 35% were software developers and researchers and 65% were postgraduate and advanced undergraduate students.

Training. We created a Google form and we invited all participants to fill it in. Initially, the users were asked to watch a short video used for training: the tutorial gave a short description of the two hierarchical drawing frameworks (the video is available at https://youtu.be/BWHc2xO4jmI).

Datasets. We experimented with a dataset of 3 graph categories with different number of nodes (20 nodes, 50 nodes and 100 nodes, i.e., small, medium and large graphs) with average degree around 1.6 (Table 3.1).

Tasks. We asked the users to answer a set of questions for the two different drawings and carry out a sequence of basic tasks. Similar to previous user studies (see, e.g., [48],[49],[50], [51]), we decided to choose tasks involving graph reading which are easily understandable also to non-expert users. Moreover, we also took into account that the purpose is to evaluate hierarchical drawings and as expected some tasks such as counting incoming or outgoing edges are rather simple and they would not produce useful insights. Thus, we considered the tasks shown in Table 3.2.

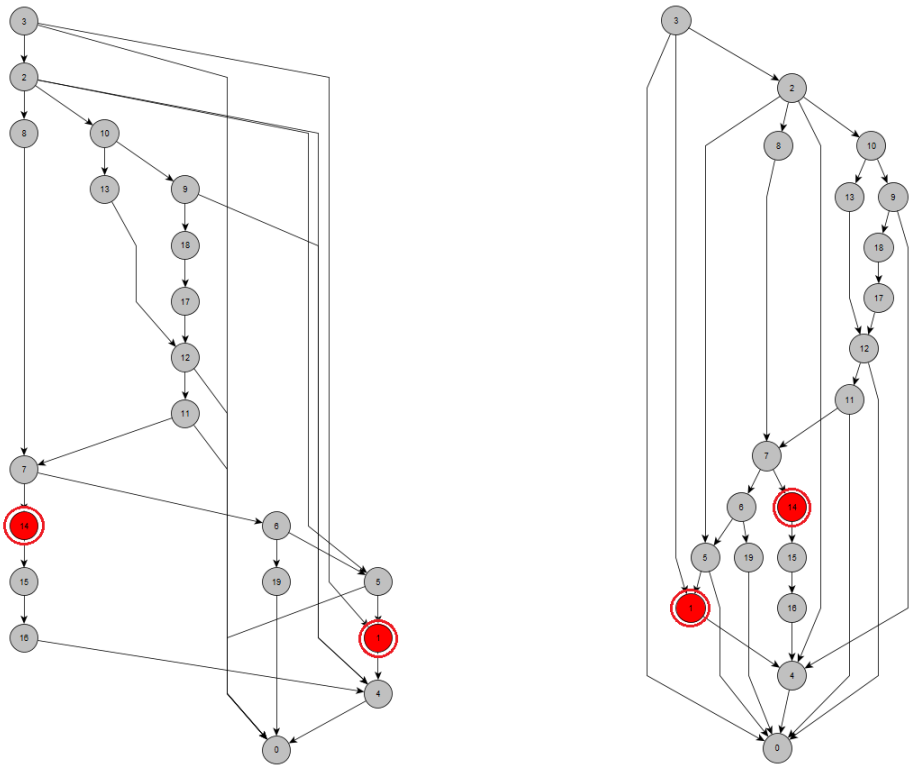
Graph	Number of nodes	Number of edges	Graph Sizes
Graph1	20	31	Small Graphs
Graph2	20	31	
Graph3	50	82	Medium Graphs
Graph4	50	79	
Graph5	100	163	Large Graphs
Graph6	100	169	

Table 3.1: Graphs dataset.

ID	*Task Description*
1	Is there a path between the two highlighted vertices?
2	How long is the shortest path between the two highlighted vertices?
3	Is there a path of length at most 3 that connects the two highlighted nodes?
4	Are all of the green vertices successors of the red vertex?

Table 3.2: The set of tasks participants had to answer for each of the 2 different graph drawing frameworks over various graphs.

For questions on Task 2, the participants had to choose a number as an answer. We do not require numeric answers for all the tasks. More specifically, for Tasks 1, 3 and 4 the participants had to choose an answer among "Yes", "No", or "Do Not Know". There was no time limit, although participants were expected to answer "Do Not Know" if a question was too difficult or too time-consuming to answer. Each of the previous tasks was repeated for each drawing framework 4 times: i.e., 2 for small and 2 for medium size graphs. Note that for each question of the same task we used different highlighted nodes. In total, the number of tasks was 32 i.e., 16 for PBF and 16 for OGDF drawings. The questions on small graphs (20 nodes) preceded those on medium graphs (50 nodes). Finally, to counteract the learning effect, the questions appeared in a randomized order. Figure 3.6 shows a snapshot for the question "Is there a path between the two highlighted vertices" for both drawings.



(a) Drawing produced by PBF (b) Drawing produced by OGDF

Figure 3.6: Snapshots of drawings of the same graph used in the user study for the question *"Is there a path between the two highlighted vertices?"*

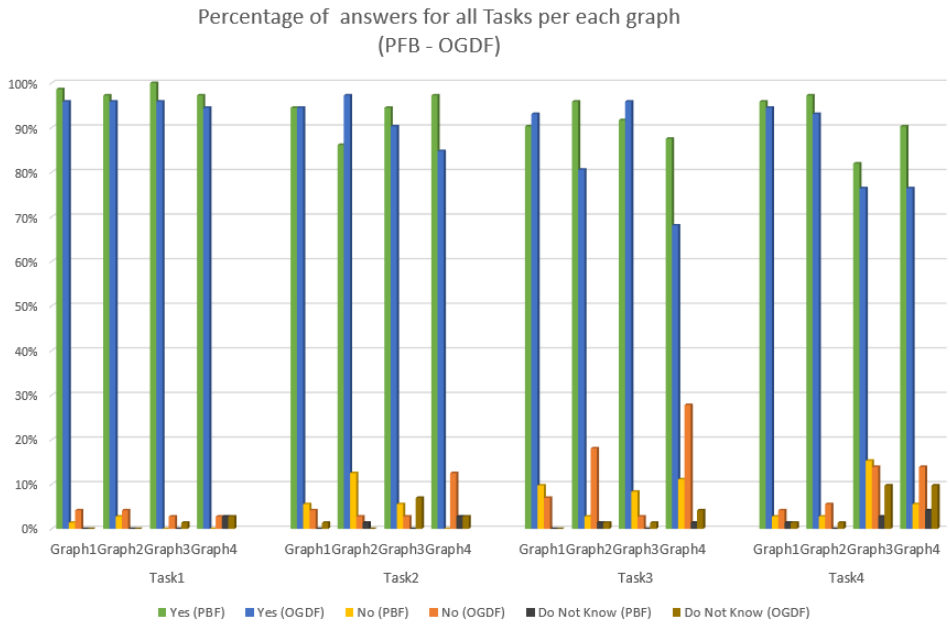
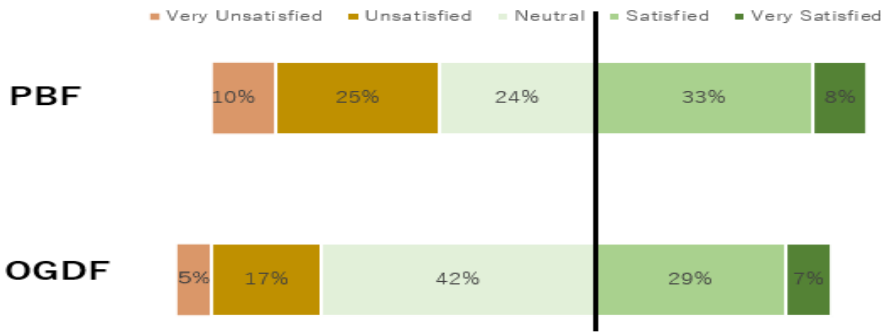
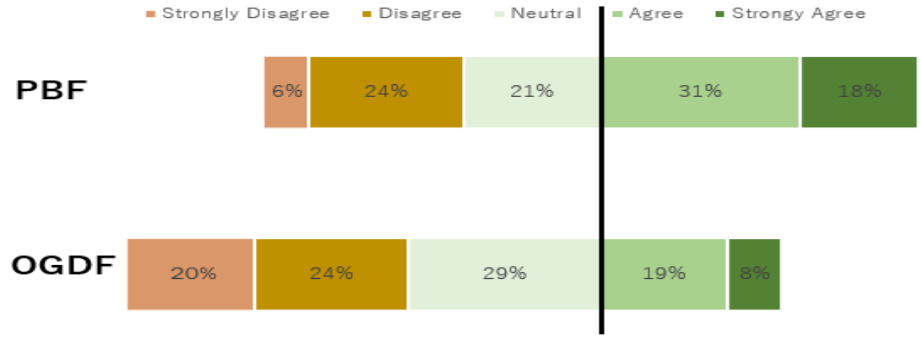


Figure 3.7: Results *(the ratio of participants that answered "Yes", "No" and "Do Not Know" over the total number of answers)* on the various tasks for each of the drawing framework *(PBF)*, *(OGDF)* over different graphs.

Task Based Comparison. For the results, we recorded the total *number of correct answers* for each question of the 2 different drawing frameworks, for all participants. Also note that the "Do Not Know" answer was considered incorrect. More specifically, regarding the first graph (i.e., *graph*1) for all tasks, we have that the users had the same performance in terms of correct answers for both PBF and $OGDF$ drawings. By examining the rest of the results the average percentage revealed that for all the graphs the performance of both drawing frameworks is not significantly different, although it is slightly better for PBF drawings for almost all tasks and drawings. We observe the same when comparing the average percentage for each task: the numbers for PBF drawings are consistently better than the numbers for OGDF drawings, but the differences are small. We show a comprehensive visualization of these results in Figure 3.7. It shows the ratio of participants that answered *"Yes"*, *"No"* and *"Do Not Know"* on the various tasks for each of the two drawing frameworks, over the different graphs where we observe again that PBF is slightly better than $OGDF$ for all cases



(a) *Percentage results for PBF and OGDF for the question I, over graph*5



(b) Percentage results for PBF and OGDF for the question *II*, over *graph*6

Figure 3.8: *Results for questions I and II.*

(Tables of Figures 3.10, 3.9). We observe that although the numbers are very low for both, the number of users that answered *"Do Not Know"* for OGDF is often double the corresponding number for PBF, which may imply that some drawings may be more confusing to some users.

In general, the performance of the participants is slightly better when they are working with PBF drawings than with OGDF drawings. Since the differences are rather small we cannot extract a concrete conclusion as to which is better for the users. However, it has become clear that the path-based framework is an interesting alternative to the Sugiyama Framework for visualizing hierarchical graphs. Furthermore, for specific applications, that require to visualize specific paths (such as critical paths) it would be the preferred choice since the nodes of each path are placed on the same x-coordinate.

Graph	Avg per graph	Task1	Task2	Task3	Task4	Avg per graph	Task1	Task2	Task3	Task4
1	95%	71/72	68/72	65/72	69/72	94%	69/72	68/72	67/72	68/72
2	94%	70/72	62/72	69/72	70/72	92%	69/72	70/72	58/72	67/72
3	92%	72/72	68/72	66/72	59/72	91%	69/72	65/72	69/72	55/72
4	93%	70/72	70/72	63/72	65/72	80%	68/72	61/72	49/72	55/72
Avg per task		98%	93%	91%	91%	Avg per task	95%	91%	84%	85%

Figure 3.9: Results *(number of correct answers)* on the various tasks for each of the drawing framework *(PBF)*, *(OGDF)* over different graphs.

Graph	Avg per graph	Task1	Task2	Task3	Task4	Avg per graph	Task1	Task2	Task3	Task4
1	0.3%	0/72	0/72	0/72	1/72	0.6%	0/72	1/72	0/72	1/72
2	0.6%	0/72	1/72	1/72	0/72	0.6%	0/72	0/72	1/72	1/72
3	0.6%	0/72	0/72	0/72	2/72	4.8%	1/72	5/72	1/72	7/72
4	2.7%	2/72	2/72	1/72	3/72	4.8%	2/72	2/72	3/72	7/72
Avg per task		0.6%	1%	0.6%	2%	Avg per task	1%	2.7%	1.7%	5.5%

Figure 3.10: Results *(number of "Do Not Know" answers over the total number of all answers)* on the various tasks for each of the drawing framework *(PBF)*, *(OGDF)* over different graphs.

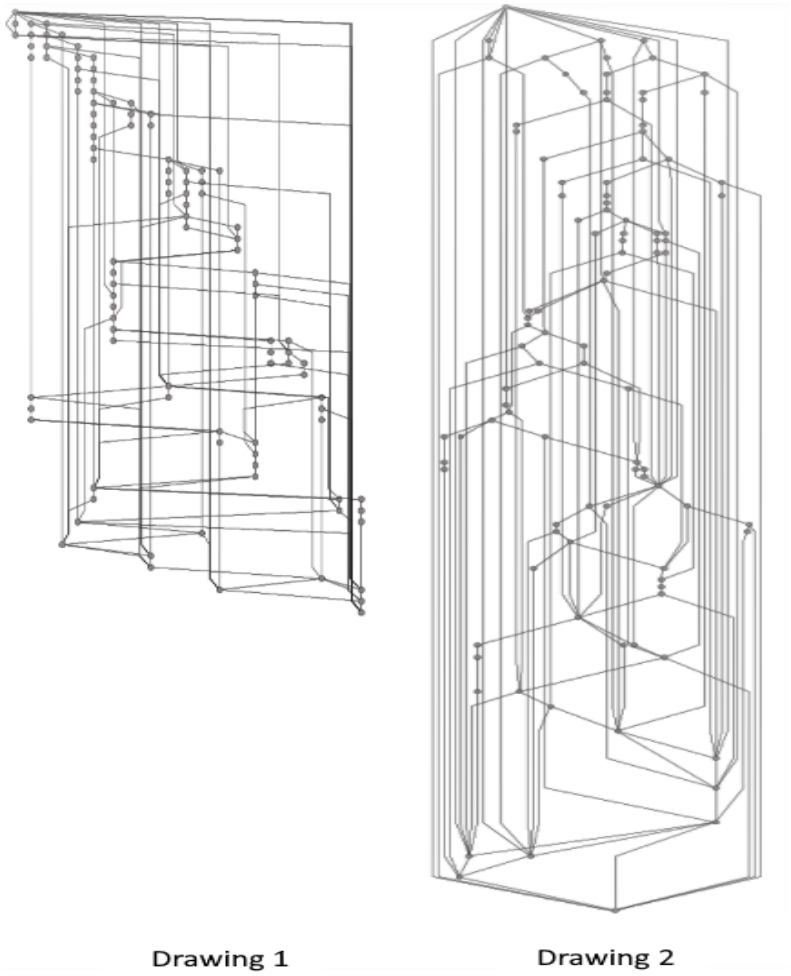
Direct Comparison of the two frameworks. As a second-level of analysis, we perform a direct comparison of the two drawing frameworks. We used the two (large) graphs of 100 nodes. To this respect we asked the participants to rate each of the 2 models by answering the following questions:

I. On a scale of 1 to 5, how satisfied are you with the following graph drawings?
II. Do you believe it would be easy to answer the previous tasks for the following graph?
III. Which of the following drawings of the same graph do you prefer to use in order to answer the previous tasks?

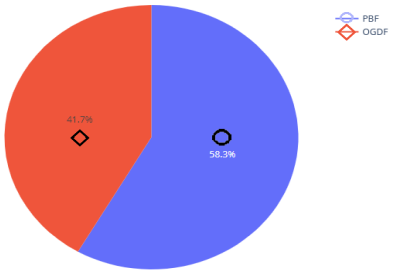
Similar to the previous experiments, we had two PBF drawings and two OGDF drawings. Since the objective of this section was to evaluate the usability of both drawing frameworks, using the System Usability Score (SUS) [52], we asked the users to answer questions I and II, by giving a rate using the following scale $<$*Very Unsatisfied, Unsatisfied, Neutral, Satisfied, Very Satisfied*$>$ and $<$*Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree*$>$ respectively. The results show that for Question I, 41% of the participants rated PBF from scale 4 and 5, in contrast to 36% for $OGDF$. Notice that the answers scale 1 and 2 are worse for PBF. This probably signifies that the users are not familiar with this new hierarchical drawing style. Question II, almost 50% of the participants rated PBF from scale 4 to 5, in contrast to less than 30% for $OGDF$, see Figure 3.8.

At the end of this user study, we asked the participants to perform a direct comparison of the two drawing frameworks for the same graph, by answering this question: "Which of the following drawings of the same graph do you prefer to use in order to answer the previous tasks" (Figure 3.11a). The results as shown in Figure 3.11b highlight that 58.3% of the participants stated that they prefer the drawing produced by PBF over the $OGDF$. In terms of statistical significance, the exact (Clopper-

Pearson) 95% Confidence Interval (CI) is (48%, 72%) indicating that PBF drawings are preferred by the users over OGDF drawings. However, given the small differences, we conclude that PBF is a significant alternative to the Sugiyama Framework for visualizing hierarchical graphs.



(a)



(b)

Figure 3.11: In (a) we show snapshots of the same graph as used in our survey. Drawing 1 is the one computed by *PBF* and Drawing 2 is the one as produced by *OGDF*. In (b) we see the percentage results for the task *"Which of the following drawings of the same graph do you prefer to use in order to answer the previous tasks"*.

Discussion:

We present a detailed general-purpose hierarchical graph drawing framework that is based on the Path Based Framework (PBF) [5]. We apply extensive edge bundling to draw all the path transitive edges, and cross edges of the graph and we minimize its height by using compaction. The experiments revealed that our implementation runs very fast and produces drawings that are readable and efficient. We also evaluated the usability of this new framework compared to *OGDF* which follows the Sugiyama Framework. The experimental results show that the two frameworks differ considerably. Generally, the drawings produced by our algorithms have lower number of bends and are significantly smaller in area than the ones produced by OGDF, but they have more crossings for sparse graphs. Thus, the new approach offers an interesting alternative for visualizing hierarchical graphs, since it focuses on showing important aspects of a graph such as critical paths, path transitive edges, and cross edges. For this reason, this framework may be particularly useful in graph visualization systems that encourage user interaction. Moreover, the user evaluation shows that the performance of the participants is slightly better in PBF drawings than in OGDF drawings and the participants prefer PBF in overall rating compared to OGDF.

Chapter 4

Variants of a new hierarchical drawing framework graphs

The visualization of directed (sometimes acyclic) graphs has many applications in several areas of science and business. Such graphs often represent hierarchical relationships between objects in a structure (the graph). In several applications, such as graph databases and big data, the graphs are very large and the usual visualization techniques are not applicable. In their seminal paper of 1981, Sugiyama, Tagawa, and Toda [6] proposed a four-phase framework for producing hierarchical drawings of directed graphs. This framework is known in the literature as the "Sugiyama" framework, or algorithm. Most problems involved in the optimization of various phases of the Sugiyama framework are NP-hard. In [9] a new framework is introduced to visualize directed graphs and their hierarchies which departs from the classical four-phase framework of Sugiyama and computes readable hierarchical visualizations by "hiding" (*abstracting*) some selected edges while maintaining the complete reachability information of a graph.

In this chapter we present several algorithms that follow that framework. Our algorithms reduce the visual complexity of the resulting drawings by (a) drawing the vertices of the graph in some vertical lines, and (b) by progressively *abstracting* some transitive edges thus showing only a subset of the edge set in the output drawing. The process of progressively abstracting the edges gives different visualization results, but

they all have the same transitive closure as the input graph. Notice that this type of abstraction has additional applications in storing the transitive closure of huge graphs, which is a significant problem in the area of graph databases and big data [53–57]. We also present experimental results that show a very interesting interplay between bends, crossings, clarity of the drawings, and the abstraction of edges.

A *path* and a *channel* are both ordered sets of vertices. In a path every vertex is connected by a direct edge to its successor, while in a channel any vertex is connected to it by a directed path (which may be a single edge). The concept of channel can be seen as a generalization of the concept of path. In the literature the channels are also called *chains* [53].

Figure 4.1 shows an example of three different hierarchical drawings: part (a) shows the drawing of a directed graph G computed by Tom Sawyer Perspectives [37] (a tool of Tom Sawyer Software) that follows the Sugiyama framework; part (b) shows a hierarchical drawing computed by our first variant algorithm taking G as input; part (c) shows an abstracted hierarchical drawing computed by our final variant that removes all path edges and selected transitive cross edges. Notice that in part (b) the transitive edges within each vertical path are not shown. Part (c) shows a hierarchical drawing where all path edges and transitive cross edges are abstracted. The advantages of the last drawing are (i) clarity of the drawing due to the sparse representation, (ii) all path edges and transitive edges (within a path) are implied by the x and y coordinates, (iii) the drawn graph has the same transitive closure as G, (iv) it gives us a technique to store the transitive closure of G in an extremely compact data structure, and (v) a path between vertices that are on different paths (of the decomposition) can be obtained by traversing one cross edge.

Even though the Sugiyama framework is very popular, and many of the (sub)problems for each phase have turned out to be NP-hard, its main limitation is the fact that the heuristic solutions and decisions that are made during previous phases (e.g., crossing reduction) will influence severely the results obtained in later phases. Nevertheless,

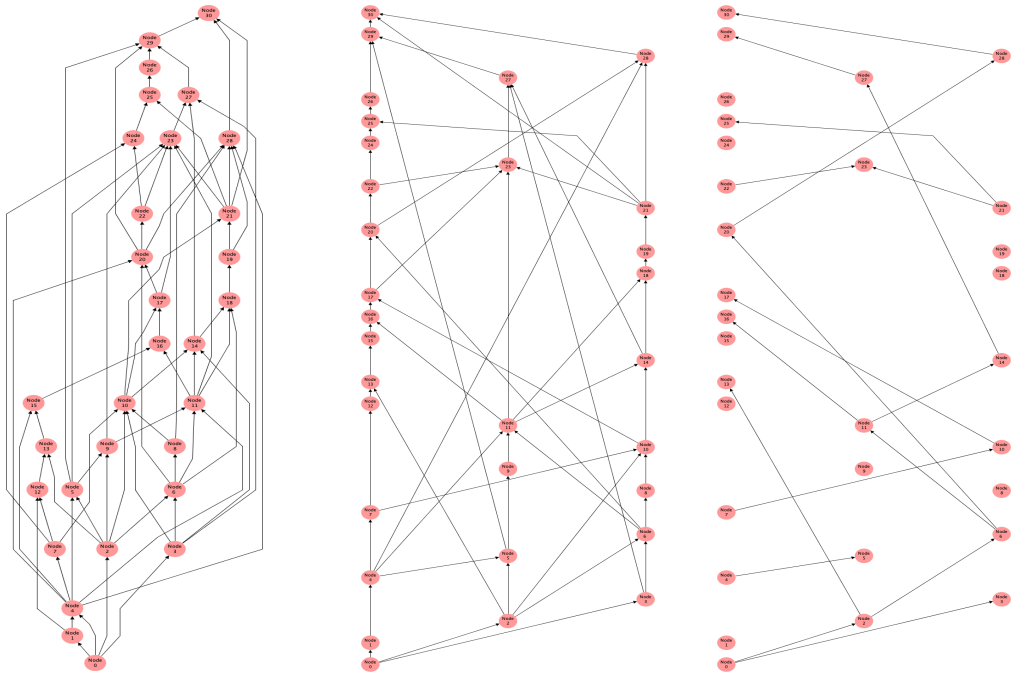


Figure 4.1: (a) A drawing of a Graph G as computed by Tom Sawyer Perspectives following the Sugiyama framework; (b) a drawing based on G computed by our first variant; (c) an abstracted hierarchical drawing computed by our final variant.

previous decisions cannot be changed in order to obtain better results. This framework can be viewed as a horizontal decomposition of G into (horizontal) layers. By contrast, the framework of [9] and all variants presented here can be viewed as a vertical decomposition of G into (vertical) paths/channels. Most problems here are vertically contained thus reducing their time complexity. It draws either (a) graph G without the transitive "path/channel edges" or (b) a condensed form of the transitive closure of G. Of course, the "missing" incident (transitive) edges of a vertex can be drawn interactively on demand. An added advantage of this framework is that it allows (or it even encourages) the user to use his/her own paths as input to the algorithms. This means that paths/channels that are important for specific applications can be easily visualized by vertically aligning their nodes.

The algorithms presented in this chapter are variants of the path based algorithm presented in [9]. Namely we present seven variants (including the original one) that progressively remove edges, crossings and bends. Each variant has its own advantages

and disadvantages that can exploited in various applications. Furthermore, due to its flexibility, new variants can be created based on the needs of specific applications. We also present experimental results that further demonstrate the power of edge abstraction and their impact on the number of bends, crossings, edge bundling, etc. Notice that the above variants can be easily modified to work using the concepts of channel decomposition of a DAG and of channel graph as described in [9].

This chapter is organized as follows: the next section presents necessary knowledge, including a brief description of the basic concepts of the path based algorithm of [9]. In Section 2 we present the variants that are based on the path based algorithm and the metrics of our experiments. Section 3 presents the experimental results and offers a comparison of the pros and cons of each variant with respect to bends, crossings, and clarity. In Section 5 we present our findings and interesting open problems.

4.1 Overview of the Path Based Framework

The Path Based Hierarchical Drawing Framework exploits a new approach to visualize directed acyclic graphs that focus on their reachability information [9]. This framework is orthogonal to the Sugiyama framework in the sense that it is a vertical decomposition of G into (vertical) paths/channels. Most problems are vertically contained thus reducing their time complexity. The vertices of a graph G are partitioned into paths, called a *path decomposition* and the vertices of each path are drawn *vertically aligned*. It consists of only two steps: (a) the cycle removal step (if the graph contains directed cycles) and (b) the hierarchical drawing step.

For the purposes of reachability we propose that Step (a) follows a simple approach: compute the *Strongly Connected Components (SCC)* of G in linear time and cluster and collapse each SCC into a supernode. Clearly, the resulting graph will be acyclic. This approach has been used in previous papers for various applications, see for example [9, 58, 59].

Regarding Step (b), the path decomposition may be application defined, user de-

fined or automatically computed by an algorithm. There are several algorithms that compute a path decomposition of minimum cardinality [33–36]. For the rest of this chapter, we will assume that the path decomposition is an input to the algorithm along with G. We use an algorithm that computes a path based hierarchical drawing given a DAG $G = (V, E)$ and a path decomposition S_p of G, see [9].

A *path decomposition* of G is a set of vertex-disjoint paths $S_p = \{P_1, ..., P_k\}$ such that every vertex $v \in V$ belongs to exactly one of the paths of S_p. A path $P_i \in S_p$ is called a *decomposition path*. The *path decomposition graph* of G associated with a path decomposition S_p is a graph $H = (V, A)$ obtained from G by removing every edge $e = (u, v)$ that connects two vertices on the same decomposition path $P_i \in S_p$ that are not consecutive in the order of P_i. An edge of H is a *cross edge* if it is incident to two vertices belonging to two different decomposition paths, else it is a *path edge*. Graph H is obtained from G by removing some transitive edges between vertices in a same path. A *path based hierarchical drawing* of G given S_p is a hierarchical drawing of H where two vertices of V are placed in a same x-coordinate if and only if they belong to a same decomposition path $P_i \in S_p$. Algorithm PB-Draw computes a path based hierarchical drawing of G. Thus we can read and understand correctly any reachability relation between the vertices of G by visualizing H, as shown in Subsection 4.1.1.

Using a path decomposition with a small cardinality may improve the performance of our algorithm in terms of area, bends, number of crossings and computational time. As already discussed, computing such a minimum size path decomposition is a well known problem and it provides a great advantage to this framework. Also, the use of the path decomposition concept adds flexibility to the framework, since the paths can be user defined or application specific. The visibility of such important/critical paths is extremely clear in our drawings, since they are all vertically aligned.

4.1.1 Algorithm PB-Draw

The following algorithm and theorems and lemmas are from [9]

Theorem 1 *Let G be a DAG and let S_p be a path decomposition of G. The path based graph H of G associated with S_p have the same reachability properties of the G.*

Algorithm 7 PB-Draw(S_p, H)
Input: a path decomposition S_p of a DAG G; a path based graph H of G associated with S_p; a topological sorting T of the vertices of H, where T_v is the position of v in this sorting.
Output: The path based hierarchical drawing Γ of G associated with S_p.

1: Compute a drawing Γ of H by:

   1. assign to every vertex $v \in V$ belonging to the decomposition path P_i an x-coordinate $X(v) = 2i$ and an y-coordinate $Y(v) = T_v$
   2. draw every edge $e = (u, v) \in E$ straight line.

2: If the straight line drawing of e intersects some vertex $w \in V$ different from u or v in Γ, introduce a bend on e in the position (X_b, Y_b), where $Y_b = Y(v) - 1$ and:

   1. If $X(u) < X(v)$: $X_b = X(u) + 1$
   2. Else, if $X(u) \geq X(v)$: $X(b_e) = X(u) - 1$

Let Γ be a drawing computed by PB-Draw. The following Lemmas and Theorem are proved in [9].

Lemma 2 *Any edge $e = (u, v)$ does not intersect a vertex different from u and v in Γ.*

The following lemma shows how Algorithm PB-Draw bundles the edges. We apply this technique of bundling in every variant of PB-Draw that we introduce in the next sections; for this reason, we better describe it in Section 4.2.

Lemma 3 *Let $e = (u, v)$ and $e' = (u', v')$ be two edges drawn with a bend in Γ. Their bends are placed in the same point if and only if u and u' are in the same decomposition path and $v = v'$.*

Finally, the following theorem shows that Algorithm PB-Draw computes efficiently path hierarchical drawings with a very small area, number of bends and number of bends per edge.

Theorem 4 *Let G be a DAG with n vertices and m edges, let S_p be a path decomposition of G and let k be the cardinality of S_p. Algorithm PB-Draw computes a path based hierarchical drawing Γ of G given S_p in $O(mk)$ time. Furthermore, $Area(\Gamma) = O(kn)$ and every edge has at most one bend.*

4.2 Variants, Metrics, and Datasets

In this section we present the variants of Algorithm 7 that we used for our experiments and the metrics that we considered. We performed two types of experiments: (a) based on measurements over datasets with respect to the number of bends and crossings *(Variant 0 and Variant 1)* and (b) based on edge abstraction *(Variant 2, Variant 3, Variant 4, Variant 5, and Variant 6)*.

All variants use edge bundling as described by Lemma 3 of Section 4.1.1. Refer to Figure 4.2a. Namely, all edges that start from vertices of a decomposition path P and go into the same target vertex v bend at the same point. All such edges use the same straight line segment from the bend to vertex v. For example, we bundle edges $(21, 30)$ and $(28, 30)$ by bending them at the same point and by overlapping them from this point to the target vertex, which is vertex 30. Similarly we do the same for edges $(4, 28)$ and $(20, 28)$. This type of edge bundling is very useful in the sense that it reduces the total number of bends and crossings, and it reuses some portions of edges.

4.2.1 Variants

We present now a suite of drawing techniques, our variants, that are based on Algorithm 7. Our variants are motivated by the concept of Data-Ink ratio. Tufte in [10]

introduced the concept Data-Ink ratio, as the ratio of ink that is used to present essential data compared to the total amount of ink used in the entire drawing. This type of abstractions has additional applications in querying huge graphs, which is a significant problem in the area of graph databases and big data [53–57]. Our variants can be further customized depending upon the requirements of an application or a user. For example, *Variants 1-4* can be used in PERT flows and in Business process visualizations.

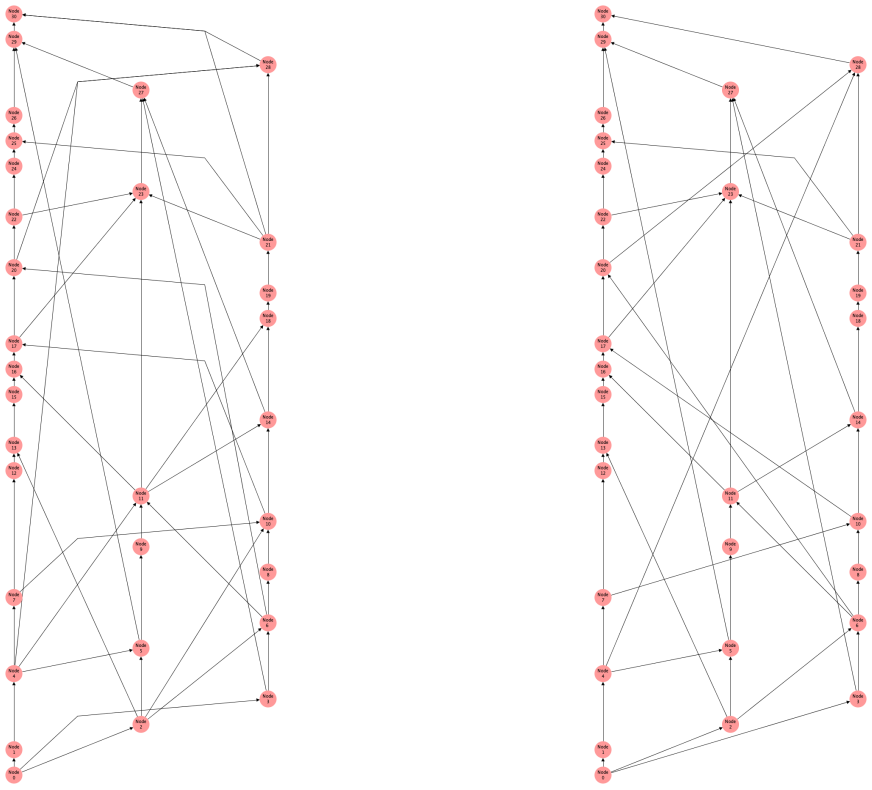
- **Variant 0:** This variant is precisely the same as our baseline, Algorithm 7. See, for example, Figure 4.1.

- **Variant 1:** We denote by *jumping cross edge* an edge $e = (u, v)$ such that $|X(v) - X(u)| > 1$. In this variant we place a bend on every jumping cross edge of Γ. Refer Figure 4.2a, where, for example, the jumping cross edge $e = (7, 10)$ has a bend.

- **Variant 2:** For every vertex u we abstract edge $e_1 = (u, v)$ if there exists an edge $e_2 = (u, v')$ such that v' and v are in the same decomposition path P and v' precedes v in the order of P (edges have common source node). Refer to Figure 4.2b, where, for example, $e_1 = (2, 10)$ and $e_2 = (2, 6)$.

- **Variant 3:** For every vertex v we abstract the edge $e_1 = (u, v)$ if there exists an edge $e_2 = (u', v)$ such that u' and u are in the same decomposition path P and u precedes u' in the order of P (edges have common target node). Refer to Figure 4.3a, where, for example, $e_1 = (21, 30)$ and $e_2 = (28, 30)$.

- **Variant 4:** Apply the removal of Variant 2 and Variant 3. Refer to Figure 4.3b, where we removed both $(2, 10)$ and $(21, 30)$.



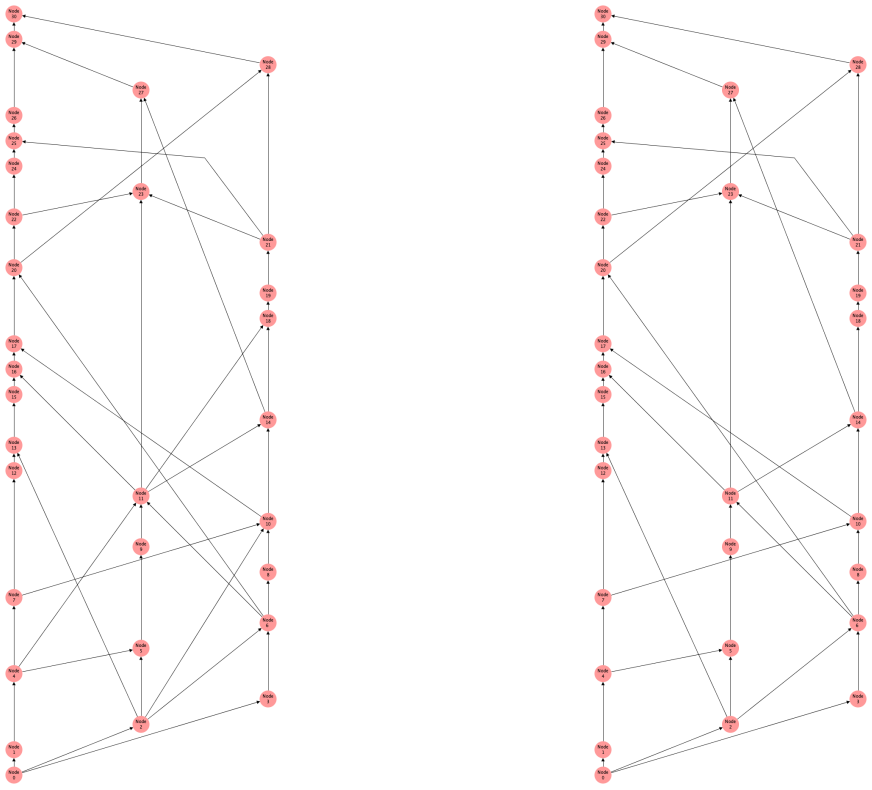
(a) (b)

Figure 4.2: Drawings of DAG 1 drawn with (a) Variant 1 and (b) Variant 2.

4.2.2 Final Abstraction

An important aspect of our variants is the preservation of the mental map that can be expressed by the reachability information of a DAG. Since the nodes in each path of the decomposition are vertically aligned, drawing the path edges does not add much information to the mental map of the user. Hence their removal from the drawing will reduce the number of crossings and the number of edges drawn. Toward to that, we propose an extended abstraction drawing model generated as a combination of the aforementioned variants as shown in Figure 4.4a and 4.4b.

The main purpose of this abstraction is that we want to retain the visual reachability while minimizing the visual complexity of the drawing. For instance, in our variants as stated in previous sections, paths can be either application based e.g., critical paths or user defined. Consider Variant 0 the path edges can be removed

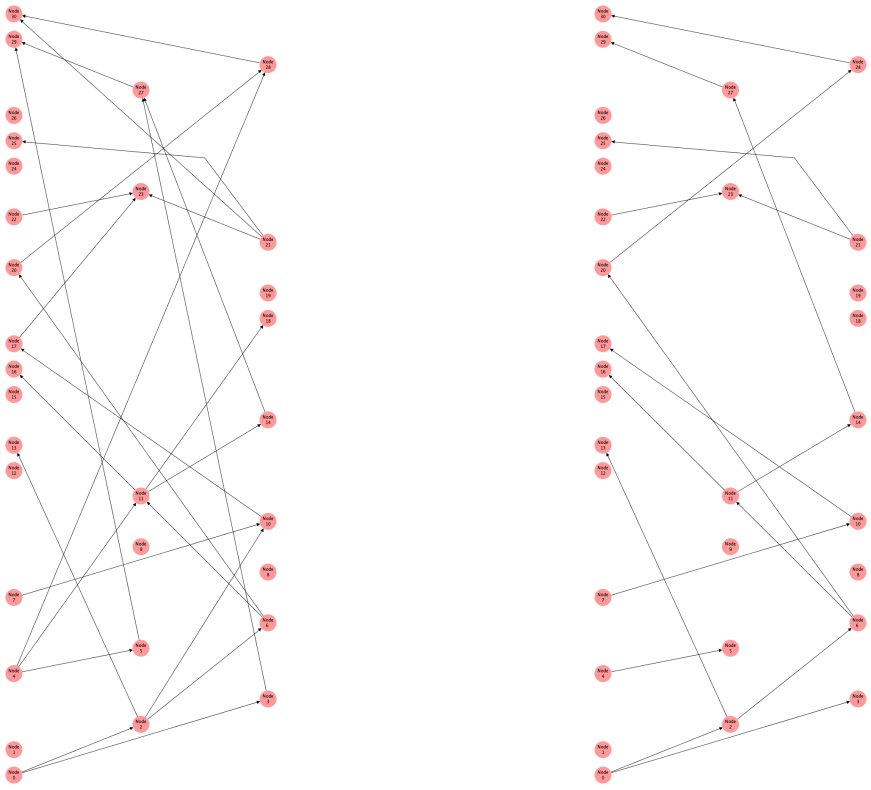


(a) (b)

Figure 4.3: Drawings of DAG 1 drawn with (a) Variant 3 and (b) Variant 4.

from the drawing since their existence is implied by the fact that they share the same x-coordinate. We refer to this variant as *Variant 5*. Please notice that we do not remove any number of "random" edges in order to create less complex drawings of the same graph but rather we use the unique characteristics of the drawing which may also be application depended. We can further reduce the total number of edges drawn, and as a result the number of crossings, by using this abstraction in combination with *Variant 4* to create a more abstracted drawing, called *Variant 6*. Therefore, we define the following two variants:

- **Variant 5**: These drawings are obtained from the drawings of Variant 0 by removing all path edges (see Figure 4.4a).
- **Variant 6**: These drawings are obtained from the drawings of Variant 4 by removing all path edges (see Figure 4.4b).



(a) (b)

Figure 4.4: Drawings of DAG 1 drawn with (a) Variant 5 and (b) Variant 6.

Theorem 5 *Let G be a DAG with n vertices and m edges, let S_p be a path decomposition of G and let k be the cardinality of S_p. It is possible to compute the drawings Γ_1 according to Variant 1 in $O(n+m)$ time and the drawings Γ_2, Γ_3, Γ_4, Γ_5, and Γ_6 according to Variant 2, Variant 3, Variant 4, Variant 5, and Variant 6, respectively, in $O(mk)$ time.*

Proof of Theorem 5

In this section we prove Theorem 5. First of all, we study the case of the drawing Γ_1 computed according to Variant 1. We can compute Γ_1 by simply ignoring Step 2 of Algorithm 7, which requires $O(mk)$ time, and placing a bend in every jumping cross edge. This operation and all the other operations of Algorithm 7 require $O(n+m)$ time. Hence, we can compute Γ_1 in $O(n+m)$ time. We now focus our attention on Variants 2-6.

Let G be a DAG and let S_p be a path decomposition of G having cardinality k. Let Γ_0 be a drawing of G computed by Algorithm 7. We present Algorithm Compute-V2, which describes how it is possible to compute a drawing Γ_2 according to Variant 2 given Γ_0. The algorithm takes as input a path decomposition S_p of G and Γ_0 and it gives as output Γ_2. For every vertex of Γ_0 the algorithm checks the adjacent vertices and, for every path P_i, it stores the one having minimum y-coordinate in an array A_k. Then it removes all edges (v, w) such that $Y(w)$ is not stored in A_k.

Algorithm 8 Compute-V2 (S_p,Γ_0)
Input: a path decomposition $S_p = (P_1, ..., P_k)$ of a DAG G; a path based hierarchical drawing Γ_0 according to Variant 0
Output: a path based hierarchical drawing Γ_2 according to Variant 2

1: $\Gamma_2 = \Gamma_0$
2: For any vertex v drawn in Γ_0:

- Create an array A of k positions
- For every edge $e = (v, w)$ in Γ_0
  - Let P_i be the path of w
  - If $A[i] = void$:
    - $A[i] = Y(w)$
  - Else:
    - $A[i] = min(A[i], Y(w))$
- For every edge $e = (v, w)$
  - Let P_i be the path of w
  - If $T_w \neq A_i$
    - remove e from Γ_2

Algorithm Compute-V2 requires linear time, since it visits every vertex once and every edge twice and, for every visit, it performs a constant number of operations.

It is easy to see that we can define two similar algorithms in order to compute drawings Γ_3 and Γ_4 according to Variant 3 and Variant 4, respectively, by taking Γ_0 and S_p as input. Therefore, given Γ_0 it is possible to compute Γ_2, Γ_3 and Γ_4 in $O(n + m)$ time. It is also true for the drawings Γ_5 and Γ_6 according to Variant 5 and Variant 6, since they can be computed from Γ_0 and Γ_4 by removing the path

Table 4.1: Average **execution** times of the variants over the *5 DAGs*.

Variants	*DAG 1*	*DAG 2*	*DAG 3*	*DAG 4*	*DAG 5*
Variant 0	36 ms	51 ms	59 ms	103 ms	137 ms
Variant 1	33 ms	40 ms	55 ms	114 ms	128 ms
Variant 2	37 ms	39 ms	62 ms	99 ms	138 ms
Variant 3	41 ms	48 ms	46 ms	93 ms	141 ms
Variant 4	49 ms	43 ms	54 ms	108 ms	104 ms

edges. Moreover, notice that Γ_0 can be computed in $O(km)$, according to Theorem 4. Hence, we can compute Γ_2, Γ_3, Γ_4, Γ_5, Γ_6 in $O(mk)$.

4.2.3 Metrics and Datasets

The set of DAGs that was used in the experiments contains five Datasets (DAGs) which were produced in a controlled fashion in order to have a number of nodes and edges, as a factor of the density of the graph. DAG 1 is one of the DAGs that was used to illustrate Algorithm 7 in [9]. Table 4.2 gives a summary for each DAG.

Metrics for the Experimental Results.

Our analysis aims to evaluate the performance of the various variants of the basic algorithm for each of the aforementioned DAGs. To this end, we use the following:

- **Number of edges drawn in the drawing.**
- **Number of cross edges drawn in the drawing.**
- **Number of bends.**
- **Number of crossings.**
- **Execution time:** is the average execution time for producing each drawing.

Table 4.2: DAGs Statistics.

Name of Dataset	Number of Nodes and Edges	Completeness (%)
DAG 1	*30 nodes and 69 edges*	~ 16
DAG 2	*50 nodes and 61 edges*	5
DAG 3	*50 nodes and 121 edges*	10
DAG 4	*100 nodes and 246 edges*	5
DAG 5	*100 nodes and 494 edges*	10

4.3 Analysis of the Performance

Here we report the experimental details describe the results. The experiments run on a single machine having an 3.1 Ghz i7 dual core, 16 GB main memory (using 5GB allocated for our implementation) and 500GB flash storage disk space. We report the average time of 5 runs per DAG.

Table 4.1 shows the performance, *Execution time* (ms), of the Java implementation of our suite of drawing solutions as produced by Tom Sawyer Software TS Perspectives [37]. The first figure reflects the *the number of edges drawn* for each of the variants over the five DAGs illustrated in Figure 4.5. Similar to that, Figures 4.6, 4.7, and 4.8 show the results regarding the *number of cross edges drawn, bends and crossings* respectively for each of the variants.

An interesting observation is that our variants produce hierarchical drawings suitable for large datasets since the reachability information can be seen with little effort while the execution time to produce these results is rather small. Toward to this, *Variant 4* gives the most promising results since it outperforms in number of crosses, bends and drawn edges for all DAGs.

Let us analyze the results of the experiments. We remark that the variants and experiments are described for the path based framework, but, of course, they can be used with the channel based framework as well.

First we discuss the number of edges drawn by our variants. By construction

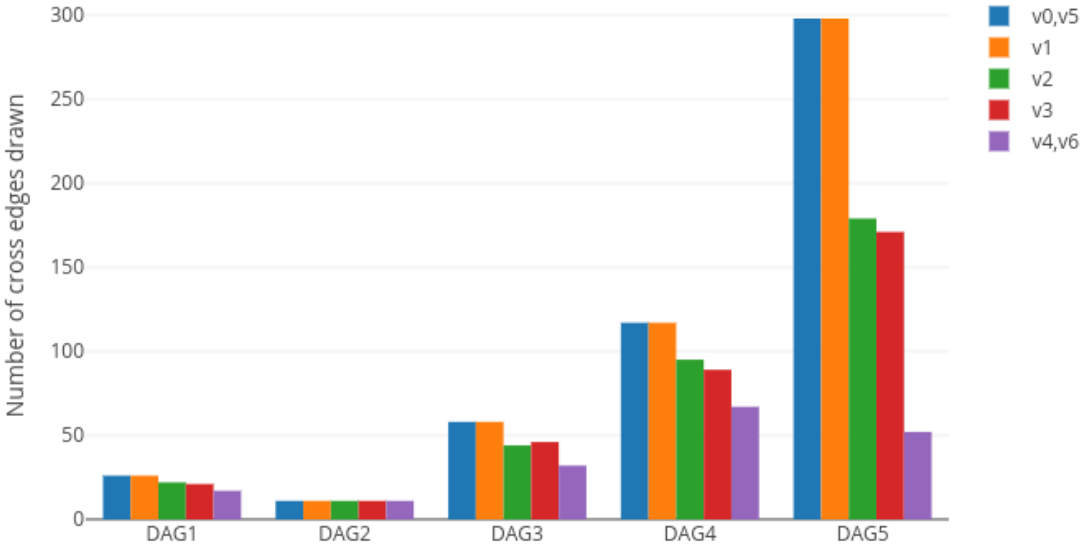


Figure 4.5: Results on *number of cross edges drawn* for each variant over all DAGs.

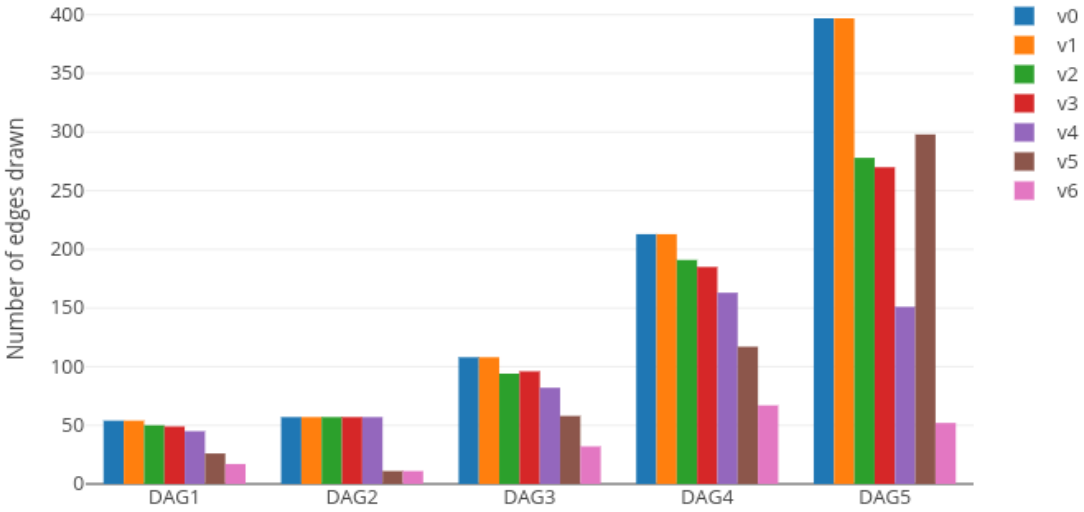


Figure 4.6: Results on *number of edges drawn* for each variant over all DAGs.

Variant 0 and Variant 1 draw exactly the same set of edges as it is evidenced by Figures 4.5 and 4.6. The same figures show that Variant 2 and Variant 3 are similar in the number of edges they draw. Clearly, the number of edges drawn by Variant 4 is significantly lower than the number of edges drawn by the other variants. This effect is emphasized in Figure 4.5, where the number of cross edges drawn by Variant 4 for DAG 5 is about one sixth of the number of cross edges drawn by Variant 0 and Variant 1. Finally, we focus on Variant 5 and Variant 6. The sets E_5 and E_6 of the

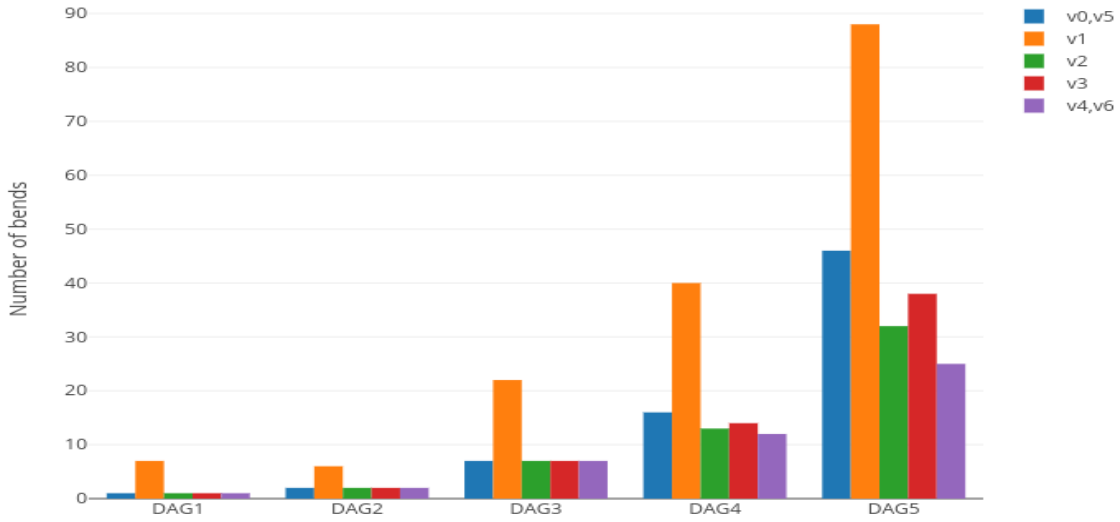


Figure 4.7: Results on *number of bends* for each variant over all DAGs.

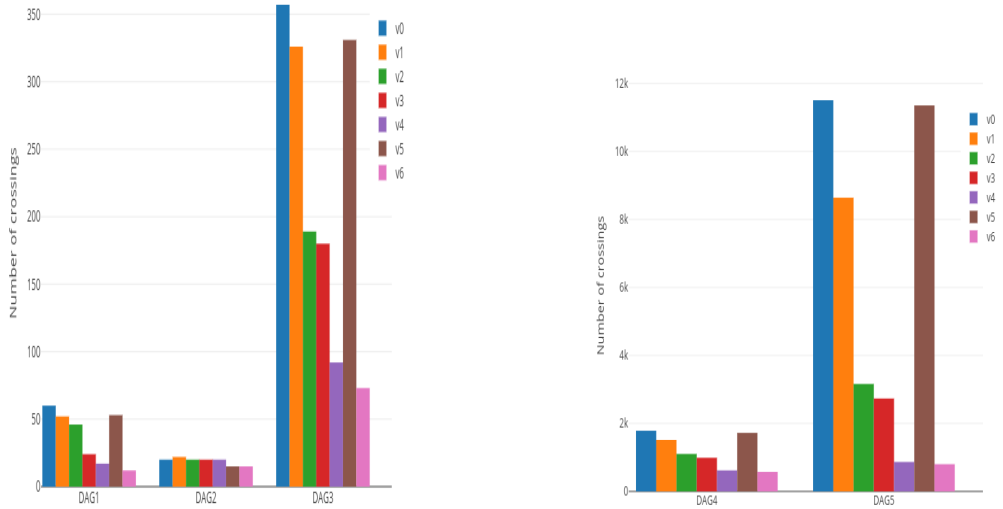


Figure 4.8: Results on *number of crossings* for each variant over the (a) DAGs 1,2,3 and (b) DAGs 4,5.

edges that Variant 5 and Variant 6 draw is a subset of the sets of edges E_0 and E_4 that Variant 0 and Variant 4 draw, respectively. The cardinality of E_5 and E_6 is much smaller than the cardinality of E_0 and E_4 if most of the edges drawn by Variant 0 and Variant 4 are path edges, as shown in Figure 4.6 for DAG 2. Variant 5 and Variant 6 by construction draw the same set of cross edges of respectively Variant 0 and Variant 4.

As can be seen in Figure 4.7 the drawings computed by Variant 2, Variant 3, and

Variant 4 have very few bends on the average. For example, DAG 5 in Variant 3 has 270 edges and the corresponding drawing has only 38 bends, i.e., we have 0.14 bends per edge. On the other hand, the drawing computed by Variant 1 is less efficient in placing bends. Refer again to DAG 5: in Variant 1 this DAG has 397 edges and the corresponding drawing has 88 bends, i.e, we have 0.22 bends per edge. The number of bends in drawings computed by Variant 5 and Variant 6 and respectively Variant 0 and Variant 4 is the same, since the path edges are drawn straight line in all our variants.

The number of crossings is influenced heavily by the number of edges drawn and the extent of edge bundling. Figure 4.8 shows that the performance of Variant 1 is slightly better than that of Variant 0. This can be explained by the fact that in Variant 1 there are more bundles of edges and this naturally decreases the number of crossings. The other variants all have much better performance than Variant 0 and Variant 1 because the corresponding drawings contain significantly fewer edges. Figure 4.8 shows that the number of crossings is almost the same in the drawings of Variant 5 and Variant 6 and Variant 0 and Variant 4, respectively. This result is very important, since it is an evidence of the fact that path edges participate in a few crossings and, therefore, the decomposition paths can be visualized very clearly in our drawings.

Finally, Table 4.1 shows that the execution time does not vary significantly depending on which variant we choose. However, Variant 4 seems to be the most scalable, since the increase of its execution time is modest as the number of edges of the input DAG increases. Notice that we do not report the execution times of Variant 5 and Variant 6 since they are similar to the execution times of Variant 0 and Variant 4, respectively.

Discussion: In this chapter we presented a set of variant algorithms that attempt to draw DAGs hierarchically with few bends and crossings, and by abstracting edges in order to improve the clarity of the drawings. Our study assumes that the path decomposition is given as part of the input, or a minimum size decomposition is computed by one of the known algorithms. However, it is interesting to study the problem of computing a path decomposition and placement of the paths of G which implies the minimum number of jumping cross edges in our drawings. The use of such a decomposition and placement would considerably reduce the number of edges drawn, bends, and crossings in our drawings. Another interesting open problem is the development and implementation of some compaction strategies, which would improve the readability of our drawings and reduce their height. Finally, it would be important to comprehend human understanding issues related to the removal of some transitive edges and increasing reachability comprehension.

Chapter 5

Reachability queries in directed acyclic graphs (DAGs)

We consider the problem of answering reachability queries in directed acyclic graphs (DAGs), which is an operation required by many applications. Hence, this problem is well studied from various points of view. For example, a recent algorithm that answers reachability queries very fast with high probability is presented in [60]. Our approach is based on dominance drawings of directed acyclic graphs (DAGs), which are important in many research areas, including graph drawing [61], computational geometry [62], information visualization [63] and very large databases [56, 64]. We present efficient algorithms to construct and search a space-efficient data structure in the k-dimensional space. Let G be a DAG with n nodes and m edges. Our algorithms construct this data structure in $O(km)$ time while it can be stored in $O(kn)$ space. Any reachability query is answered in constant time. We also present experimental results that show that the number of dimensions, k, in the solutions produced by our techniques is low, which implies that our techniques perform better than the state of the art.

Let u and v be two nodes in a DAG G. If there is a path from u to v then we say that v is reachable from u (or u reaches v). Two nodes $u, v \in V$ are *incomparable* if u does not reach v and vice versa. Clearly, any reachability query can be easily answered by employing simple reachability algorithms, such as BFS or DFS. However,

these techniques require $O(n+m)$ time for each query, which can be expensive if the size of the input DAG and the number of queries are large. Our technique computes an $O(kn)$-space data structure in order to answer any reachability query in constant time. The algorithms to construct it are based on the concept of k-dimensional dominance drawing, defined as follows:

In a k-dimensional dominance drawing Γ of a DAG G a node v is reachable from a node u if and only if all k coordinates of v are greater than the corresponding coordinates of u in Γ. Clearly, it is important to minimize the number of dimensions, k, required to draw a DAG. Although testing if a DAG G has a dominance drawing in 2 dimensions requires only linear time [65], testing if G can be drawn in three or more dimensions is NP-complete [66].

Our work is motivated by the approach and the results reported in [59]. Specifically, in pages 680-681 of [59] the authors make the following statement:

"However, there is no theory or algorithm that can calculate the exact dominance drawing dimension k of a given graph. k can be extremely large in real graphs. Since no theory can give an upper bound of k, the original dominance drawing method cannot be applied directly."

In this chapter we provide answers to the above statement by presenting solutions to these important open problems. We present (a) an upper bound of k, and (b) experimental results that indicate that k is not "very large". In fact our experiments in a large spectrum of graphs show that k is usually between a small fraction of $\sqrt{n}$ and $O(\log n)$, and sometimes even a constant (for rather dense graphs). Namely, our algorithms construct a data structure in the k-dimensional space showing there is a theory that gives such upper bounds, even though finding the minimum number of dimensions is known to be NP-hard [66].

The algorithm constructs this data structure (Index) in $O(km)$ time, and it can be stored in $O(kn)$ space. Any reachability query is answered in $O(1)$ (constant) time, since we can answer it by checking only one specific dimension. Our algorithm

requires a *channel decomposition* of a DAG G as input. A minimum size channel decomposition can be computed in $O(kn^2)$ time, see [67]. It is well known that scalability is a challenge in the study of reachability, e.g., an algorithm that answers label-constrained reachability queries very fast even for web-scale graphs is presented in [68]. Since the $O(kn^2)$ computational time may not be suitable for some web-scale graphs, there exist some heuristics that compute a channel decomposition of G in linear time, see [69]. More details on these concepts are given in the next sections.

We also present experimental results that show that our techniques perform better than the state of the art. Namely, we provide experimental evidence that shows that the number of dimensions k is usually equal to a small fraction of $\sqrt{n}$ (for very sparse graphs) and sometimes it grows as slow as $O(\log n)$. For rather dense graphs sometimes k seems to be a constant as the number of nodes grows. Furthermore, the experimental results validate the choice on the maximum number of dimensions used in the experimental results of [59]. But in several cases the results show that the number of dimensions computed by our algorithms is less than the maximum number of dimensions considered in the experimental results of [59]. Another important point here is that their solutions contain "falsely implied paths (fips)" (or "false positives", in their notation). In order to resolve fips (to be defined formally in the next section) they require extra computation time, which is linear in the worst case as discussed in [59].

Additionally, we show experimentally that the number of dimensions is influenced by two properties of a given DAG: Its *density* and its *structure*. Both are important and should be taken into account when analyzing experimental results, as discussed at the end of Section 4. Motivated by this fact, we introduce a new model that is specifically targeted to randomly create DAGs with predefined density and structure. Indeed, in Section 5 we show that there is an interplay between the density and structure of DAGs which influences the number of required dimensions. To the best of our knowledge, this is the first time that this interplay is presented for this problem.

5.1 Preliminaries

Reachability queries in a directed graph G can be answered in two levels: (a) If G contains cycles then we compute the strongly connected components of G, and answer such queries in positive if the two nodes in the query belong to the same strongly connected component. Next, the strongly connected components can be reduced to supernodes in order to construct a new graph, which is a Directed Acyclic Graph (DAG). (b) Answer a reachability query in the DAG. This approach is well known and has been described in several papers for various applications, most recently in [5, 59].

Hence, in the rest of this chapter we only consider reachability queries in DAGs. An *st-graph* is a DAG, $G = (V, E)$, with one source s and one sink t; G has $n = |V|$ nodes and $m = |E|$ edges. Since any given DAG can be converted into an st-graph by simply creating a new node (new source) and connecting it to all sources (same for sinks) in the rest of the chapter, in order to simplify our presentation, we assume without loss of generality that every DAG is an st-graph.

A k-dimensional dominance drawing Γ of a DAG G is defined as follows: Each node in Γ has k coordinates such that a node v is reachable from a node u if and only if all k coordinates of v are greater than the corresponding coordinates of u. Thus, in a k-dimensional dominance drawing Γ it is possible to check if u reaches v by simply comparing the k coordinates of u and v. On the other hand, if there exists a dimension D of Γ such that $D(u) > D(v)$, then u does not reach v. A dominance drawing Γ of a DAG G combines the aspect of drawing G in the grid with the fact that the transitive closure of G is implicit by the dominance relation between grid points associated with the nodes of G. Recall that two nodes u, v of DAG G are incomparable if u does not reach v and vice versa. The *width* of a DAG G, w_G, is the maximum size of a set of incomparable nodes of G. The smallest number d for which a given DAG G has a d-dimensional dominance drawing is called *dominance drawing*

dimension of G and it is a known NP-hard problem [66].

In 2-dimensions dominance drawings of planar DAGs have many important aesthetic properties [62, 70]. A 2-dimensional dominance drawing Γ of a planar st-graph G can be computed in linear time, such that for any two nodes 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 [62, 70]. Since most DAGs have dominance dimension higher than two, it is not possible to find dominance drawings in 2-dimensions for most DAGs. Therefore, the concept of weak dominance drawings was introduced by Kornaropoulos and Tollis in [71, 72]. This concept is an extension of the concept of dominance drawing by relaxing the necessity of the existence of a path. This concept has many applications including very large databases [56, 64] and the drawing of DAGs in the overloaded orthogonal model [73]. In weak dominance, for any two nodes u and v if there is a directed path from u to v in G then $x(u) \leq y(v)$ and $y(u) \leq y(v)$ in Γ. However, the reverse may not necessarily hold. Hence, we have a *falsely implied path (fip)* when $x(u) \leq x(v)$ and $y(u) \leq y(v)$, but there is no path from u to v. Furthermore, the problem of computing a weak dominance drawing that minimizes the number of fips is shown to be NP-hard in [71, 72].

In [59] Li, Hua, and Zhou extended the concept of weak dominance drawings in higher dimensions and presented interesting experimental results. Motivated by their approach, we investigate the possibility of constructing a technique based on k-dimensional dominance drawing, where $k \geq w_G$, for any given DAG G. In the next section we present a technique for constructing an index that can be used to determine reachability between any pair of nodes in $O(k)$ time, since it is guaranteed to contain no fips. We will also present experimental results that bring to the foreground the following interplay:

Structure vs Density: There are two parameters of G that play an important role in determining the number of dimensions $k = w_G$ for G: (a) its structure and (b) its

density. Next we discuss two extreme cases in order to exhibit the importance of both parameters: (i) If G is a single Hamiltonian path from s to t, then it is clear that $k = 1$ although its density is very low since it has only $m = n-1$ edges (in fact, this density is minimum for a connected DAG). (ii) If G is a complete bipartite graph with $n/2$ nodes in each side and all edges are directed from the nodes of one side to the nodes of the other side, then the number of dimensions is clearly $k = n/2$ although it is very dense since it contains $m = n^2/4$ edges. As we will show in the following sections, each of the above parameters plays an important role in determining k, but for DAGs of "similar structure" density plays a decisive role in determining k. Conversely, we will see that for DAGs of "similar density" their structure plays a decisive role in determining k.

5.2 k-Dimensional Dominance Drawings

Let $G = (V, E)$ be an st-graph with n nodes and m edges and let s and t be the source and the sink of G, respectively. In this section we will present an algorithm to compute an upper bound k on the number of dimensions. The algorithm will also construct a k-dimensional dominance drawing for G by computing the indices of each node. Such k-dimensional dominance drawings contain no fips.

A *channel* C is an ordered set of nodes such that, given any two nodes $v, w \in C$, v precedes w in the order of channel C if and only if w is reachable from v in G. We denote by *channel decomposition* of G a set of channels $S_c = \{C_1, ..., C_k\}$ so that the source s and the sink t of G are contained in every channel and every other node of G is contained in exactly one channel.

We denote by *width* of a DAG G, w_G, the maximum size of a set of incomparable nodes of G. It was proved in [74] that the minimum size of a channel decomposition of G is w_G. Furthermore, a channel decomposition of G with w_G channels can be computed in $O(n^3)$, see [69]. A faster algorithm for computing such a channel decomposition runs in $O(w_G n^2)$ time, see [67].

Let $S_c = \{C_1, ..., C_k\}$ be a channel decomposition of G. We denote by $u = (i, j)$ the fact that u is the jth node of channel C_i. Notice that, by definition of S_c, we have $t = (i, |C_i|)$ and $s = (i, 0)$ for any $i \in [1, k]$. The *projection* of a node v in a channel $C_i \in S_c$, denoted by $prj(v, C_i)$, is the node of C_i reachable from v having the lowest position in C_i. The projection of v in the channel of S_c containing v is defined to be v itself.

Next we describe how to compute and store the projections in a $k \times n$ matrix prj, called the *projection matrix*. This matrix is similar to the compressed transitive closure of G [69]. The element stored in $prj(v, C_i)$, i.e., the element in the row associated to node v and in the column associated to channel C_i, is the lowest node in C_i) that can be reached by node v. We are ready now to describe *Algorithm Projections*, which computes the projection of each node v of G in every channel of a channel decomposition S_c.

Algorithm 9 Projections(G, S_c)
Input: A DAG G and a channel decomposition $S_c = \{C_1, ..., C_k\}$ of G.
Output: The projection matrix of G given S_c.

1: prj = new $n \times k$ matrix
2: **For** $v = (i, j) \in G$:

- **For** $h \in [1, k]$:
  a. **If** $i \neq h$:
     - prj(v, C_h)=t
  b. **Else**:
     - prj(v, C_h)=v

3: Compute a topological order T of the nodes of G.
4: **For** $l = n, ..., 1$:

- $v = T(l)$
- Let $v_1, ..., v_c$ be nodes incident to an outgoing edge of $v = (i, j)$.
- **For** $u = v_1, ..., v_c$:
  - **For** $h \in [1, k]$ and $h \neq i$:
    - prj(v, C_h)=(h, a)
    - prj(u, C_c)=(h, b)
    - prj(v, C_h)=$(h, min\{a, b\})$

Algorithm Projections takes as input a DAG G and a channel decomposition $S_c = \{C_1, ..., C_k\}$ of G and it produces as output the projection matrix of G, according to S_c. In Step 1 we initialize the matrix. In Step 2 we initialize the projections of every node v by setting its projection to be the sink t (Step 2a), except for the channel that contains v, for which the projection of v into this channel is v itself (Step 2b). In Step 3 we compute a topological order T of the nodes of G. Next (Step 4) we visit the nodes of G in descending order of T, considering its outgoing edges, ending at nodes $v_1, ..., v_c$. Since nodes $v_1, ..., v_c$ were already visited previously (due to our descending order visit), for every node $u \in \{v_1, ..., v_c\}$, we consider all the channels. For every channel C_h, we consider the projections of u and v, (h, b) and (h, a) respectively. If the projection of u has a lower position in the channel than the projection of v (i.e., $b > a$) we set the projection of v equivalent to the projection of u. Otherwise, we do not change the projection of v. This is accomplished by taking the minimum between a and b, denoted by c, and assigning to $prj(v, C_h)$ the value $(h, min\{a, b\})$. Algorithm Projections runs in $O(km)$ time, since the two outer **For** loops of Step 4 are repeated $O(m)$ times. Hence we the following lemma:

Lemma 6 *Let S_c be a channel decomposition of G. Algorithm Projections computes the projections of each node v of G in each channel C_i in $O(mk)$ time. The projections can be stored in $O(nk)$ space.*

Next we present Algorithm Indexer, which computes k topological orders of the nodes of G.

Algorithm Indexer: The algorithm takes as input a DAG G and a channel decomposition $S_c = \{C_1, ..., C_k\}$ of G and produces as output k topological orders $T_1, ..., T_k$, that will imply the coordinates of the nodes in a k-dimensional dominance drawing of G. First the algorithm computes the projection matrix by using Algorithm Projection (Step 1). Next it places every node $v = (i, j)$ in the position j of the topological order T_i (Step 2). Then, in Step 3, we take any channel C_i of S_c and we compute

the position of all the nodes of G in T_i, that do not belong to C_i: This is done by taking such a node, v, and looking at the position of its projection in T_i, which was computed in Step 2. It places v in this position. Notice that in this case some nodes are placed in the same position in T_i, and hence T_i is not a "strict" topological order at this point.

Next we perform operations on $T_1, ..., T_k$ so that, at the end of the algorithm, they will become "strict" topological orders (Step 5). In order to do that, we will use the topological order T of G, computed in Step 4. For every taken position j in T_i (i.e., a position such that there exists a node v in G where $T_i(v) = j$) we need to have a list of nodes having the same position in T_i. In Step 5a, for every $T_i \in T_1, ..., T_k$, we initialize a void list $L_{i,j}$ for every used position of T_i. In Step 5b we visit the nodes of G in increasing order given T and we add v in the last position of list $L_{i,T(v)}$. At the end of Step 5b every list $L_{i,j}$ is an ordered list on the order defined by T containing every node v of G so that $T_i(v) = j$. Finally, in Step 5c we take all the used positions of T_i in a descending order. If the list $L_{i,j}$ of the position j contains more than one nodes, then we shift the nodes in T_i according to Step 5cα and Step 5cβ: In Step 5cα we shift of $|L_{i,j}|$ positions in $T(i)$ the nodes having position in $T(i)$ higher than j, in order to create space for the nodes in $L_{i,j}$. Finally, in Step 5cβ we shift by x positions every node in position x in $L_{i,j}$.

Algorithm 10 Indexer(G, S_c)
Input: A DAG G and a channel decomposition $S_c = \{C_1, ..., C_k\}$ of G.
Output: k topological orders $T_1, ..., T_k$.

1: prj = Projections(G, S_c)
2: **For** $i \in [1, k]$:
- **For** any $v = (i, j) \in G$:
  - $T_i(v) = j$

3: **For** $i \in [1, k]$:
- **For** any $v = (h, j)$ such that $h \neq i$:
  - $prj(v, C_i) = (i, l)$
  - $T_i(v) = l$

4: Compute a topological order T of the nodes of G.
5: **For** $i \in [1, k]$:

a: **For** $j \in [0, T_i(t)]$:
- $L_{i,j}$ = new list or nodes

b: **For** $l = 1, ..., n$:
- $v = T(l)$
- $L_{i,T_i(v)}.addLast(v)$

c: **For** $j \in T_i(t), T_i(t) - 1, ..., 0$:
- **If** $|L_{i,j}| > 1$:
  - α. **For** any $v \in G$:
    - **If** $T_i(v) > j$: $T_i(v) = T_i(v) + |L_{i,j}|$
  - int $x = 0$.
  - β **For** $v = L_{i,j}[0], ..., L_{i,j}[|L_{i,j}| - 1]$:
    - $T_i(v) = j + x$
    - $x = x + 1$

Algorithm Indexer is inspired by the main concepts of Algorithm kD-Draw, which computes a k dimensional dominance drawing of a DAG G, presented in [75]. Algorithm Indexer performs this operation in Step 5. This operation does not change the property of the drawing. Indeed, if Step 5 of the algorithm changes the relative position of two nodes u and v in a given order T_i we have that: If u and v are incomparable, then there must be another two orders T_j and T_h such that $T_j(v) > T_j(u)$ and $T_h(v) < T_h(u)$; else, suppose $T(v) > T(u)$, where T is the topological order computed in Step 4. In this case we have that u reaches v. After Step 5 we have $T_i(u) < T_i(v)$, thanks to the fact that the nodes in $L_{i,j}$ are ordered in ascending order in T.

Following the main concepts of Algorithm kD-Draw and the corresponding theorem proved by Ortali and Tollis in [75], and according to the brief proof regarding Step 5 given in the previous paragraph, we present a theorem that describes the main properties of Algorithm Indexer.

Theorem 7 *Let G be a DAG and S_c be a channel decomposition of G. Assume that $T_1, ..., T_k$ are the k topological orders computed by Algorithm Indexer(G, S_c). Then, for any pair of nodes of G, v and w, v reaches w if and only if $T_i(v) \leq T_i(w)$ for all $i \in [1, k]$.*

The above theorem gives an immediate algorithm to answer any reachability query between two nodes in $O(k)$ time using the constructed data structure that requires $O(nk)$ space. However, due to the structure that we have created for any DAG G, it turns out we can do much better than that. Please recall that, given any channel decomposition $S_c = \{C_1, ..., C_k\}$ of G, each node u is assigned two numbers, denoted by $u = (i, j)$, meaning that u is the jth node of channel C_i. More generally, for any two nodes v and u of G we say that the index of the channel containing v (resp. u) is i_v (resp. i_u). It is easy to construct a simple data structure that contains these values for each node v in G in linear time. Now, we will use this index in order to answer any reachability query in constant time.

Let $v = (i_v, j_v)$ and u be any two nodes of G. According to Algorithm Indexer we have $T_{i_v}(u) = prj(u, C_{i_v}) = h$. By the definition of projection, we have that u reaches v if and only if $h \leq j_v$. It follows that, since $h = T_{i_v}(u)$ and $j_v = T_{i_v}(v)$, u reaches v if and only if $T_{i_v}(u) \leq T_{i_v}(v)$. Hence, in order to check if u reaches v, we only need to check the relative position of u and v in one specific topological order: T_{i_v}. Hence, we have the following theorem.

Theorem 8 *Let $T_1, ..., T_k$ be the topological orders of a DAG G computed by Algorithm Indexer. Given two nodes u and v of G, denoted by $u = (i_u, j_u)$ and $v = (i_v, j_v)$, u reaches v if and only if $T_{i_u}(u) \leq T_{i_u}(v)$. Similarly, v reaches u if and only if $T_{i_v}(v) \leq T_{i_v}(u)$.*

Theorem 8 implies that we can answer any reachability query by simply checking the position of two given nodes in (only) two particular topological orders from the set of all topological orders $T_1, ..., T_k$ computed by Algorithm Indexer. Hence, we have a faster algorithm to answer any reachability query as follows:

Corollary 9 *Given the set $T_1, ..., T_k$ of topological orders of a DAG G computed by Algorithm Indexer, it is possible to answer any reachability query in $O(1)$ time.*

Algorithm Reachability, directly applies Theorem 8 and the corresponding corollary in order to answer any reachability query in constant time.

For the rest of this section we assume that $k = w_G$. As discussed above, the time required to compute a minimum channel decomposition $S_c = \{C_1, ..., C_k\}$ is $O(w_G n^2)$. If k is significantly smaller than n, say, if k is considered as a constant with respect to n, then this drawing can be computed very fast, in $O(m)$ time, and can be stored in a very small space $O(n)$. In the next sections we will see several cases where k is a constant. On the other hand, if k has a value that is a fraction of n then it is computed in a time and stored in a space that is comparable with the ordinary transitive closure of the graph (i.e., $O(nm)$ time and $O(n^2)$ space in the worst case).

Algorithm 11 Reachability(G, $u = (i_u, j_u)$, $v = (i_v, j_v)$, T_{i_u}, T_{i_v})
Input: A DAG G, two nodes $u, v \in G$, and two topological orders $T_{i_u}, T_{i_v} \in \{T_1, ..., T_k\}$ computed by Algorithm Indexer
Output: An answer to the reachability query, that is: "u reaches v", or "v reaches u", or "u and v are incomparable"

1: **If** $T_{i_u}(u) < T_{i_u}(v)$:
   - **return** "u reaches v"

2: **Else, if** $T_{i_v}(v) < T_{i_v}(u)$:
   - **return** "v reaches u"

3: **Else**:
   - **return** "u and v are incomparable"

However, as we will see in the next sections, k is usually close to a "constant" if the density of the input DAGs is relatively high. If on the other hand, the density of the input DAGs is low then k typically grows as a function of n, usually between $\log n$ and $\sqrt{n}$.

5.3 Experimental Results of Algorithm Indexer

In [59] Li, Hua, and Zhou presented an algorithm that computes a multidimensional weak dominance drawing that can be stored in $O(dn)$ space, where d is the number of dimensions. They also presented experimental results on the number (or ratio) of fips ("false positives", in their terminology). More precisely, this variable is called "Fip Ratio", which is the ratio between the number of fips (false positive queries) over the total number of queries that give a positive answer. Clearly, this number is a value between 0 and 1. They also presented an algorithm to test reachability between two nodes that in the case of unreachability takes constant time, but in the case of reachability it takes $O(n + m)$ time. In our work the fip ratio is always zero, since our Algorithm Indexer computes k topological orders of the nodes of G that constitute a complete dominance drawing. The time complexity of our reachability algorithm is $O(1)$ for both reachability and unreachability. For this reason, in the

rest of this section we compare the number of dimensions that we use in order to have no fips (in other words, w_G) to the number of dimensions computed in [59], even though their results contain fips, which implies that they need $O(n+m)$ time to establish reachability for each query. The models that we use for the comparison are the same models used in [59], which are the *Erdős-Rényi*, the *Barabasi-Albert*, and the *Watts-Strogatz* small world models.

In the experiments that we present in this section we see that in almost all cases our approach gives better results than the ones given in [59] in the number of dimensions. Additionally, even in the cases where our results are similar with respect to the number of dimensions used we point out that our solutions have no fips, while in [59] there are both cases with few and many fips. If fips exist in any solution then positive reachability can be determined only after running the $O(n+m)$ time check for each query [59]. On the other hand, the approach described in [59] gives the user the possibility to reduce the number of dimensions which results, of course, in increasing the Fip Ratio, while in our approach the number of dimensions used is equal to the width of the graph. So, in this sense our results on the number of dimensions are not directly comparable to the results given in [59] (since ours contain no fips). However, if the number of required dimensions is "affordable" then our approach may prove to be very useful since it requires no extra time to check positive reachability.

Indeed, our experiments show that the width of the graph in many cases can be very small with respect to the number of nodes, n, of the graph and in some cases, when n increases the width remains (almost) constant. Hence, in these cases the number of required dimensions is "affordable". Therefore, in those cases our approach is better than the one described in [59]. For all other cases, the choice of an approach depends upon the parameters of the application (i.e., time vs. space). In some cases it might be desirable to save space, by reducing the number of dimensions, at the expense of having fips (which implies having to run the $O(n+m)$ time check for each positive query). In many other cases one might prefer to use w_G dimensions in order

to have 0 fips, which implies immediate response in constant time to any reachability query.

In order to be consistent with the experimental setting used in [59], we use NetworkX [76] to generate our graphs using the same parameters. While generating a graph, when an edge between two vertices x and y of the graph is added, where x and y are two integers, the methods of NetworkX add the edge (x, y) to the graph if $x < y$ or the edge (y, x) otherwise. Hence, the generated graphs are always DAGs. The experimental results show that the value of the width for many DAGs can be really small, especially for dense ones. Additionally, the experimental results show that sometimes as the number of nodes increases, the width increases at a much slower pace and sometimes it does not even increase at all, especially for dense DAGs. In order to analyze this aspect we performed experiments for different values of the number of nodes of our graphs ($n = 1000, 2000, 5000, 10000$). Notice that in [59] the value of n is fixed to 10000. Our experimental results report the average number of dimensions over five graphs for every category of graphs that we analyze.

5.3.1 The Erdős-Rényi model [77]:

It is a model of graph where every edge has a probability to exist equivalent to a given parameter p (obviously, since p is a probability, we have that $p \in [0, 1]$). The other parameter of this model is the number of nodes of the graph, n. We consider the cases where $n = 1000, 2000, 5000, 10000$ and $p = 0.05, 0.1, 0.15, 0.2, 0.25$.

The results of our experiments on $k = w_G$, the number of dimensions, obtained for this model are presented in Figure 5.1. For this model the number of dimensions required in order to have a dominance drawing is at most 21. According to the results showed in Figure 2 of [59] concerning the same model the number of dimensions required is up to 50 in order to have a weak dominance drawing (i.e., fips exist). Recall that the experiments reported in [59] are only for $n = 10000$. In our experiments and in the experiments in [59] it is shown that when the graphs are rather dense (i.e.,

as p increases) the number of dimensions needed to have a dominance drawing, for us, or a weak dominance drawing with a very small Fip Ratio for [59], drastically decreases. We conclude the analysis on this model with an interesting observation: The number of nodes of the graph does not significantly influence the number of required dimensions. The value of k, indeed seems to remain stable or even decrease as the number of nodes increases. In general, in our experiments the width ($k = w_G$) of the graph seems to scale very well for this particular model and the effectiveness of our technique, in this case, depends almost exclusively on the density of the graphs. In other words, for this model the density of the graphs is of paramount importance, whereas the node-size does not affect the resulting k that much.

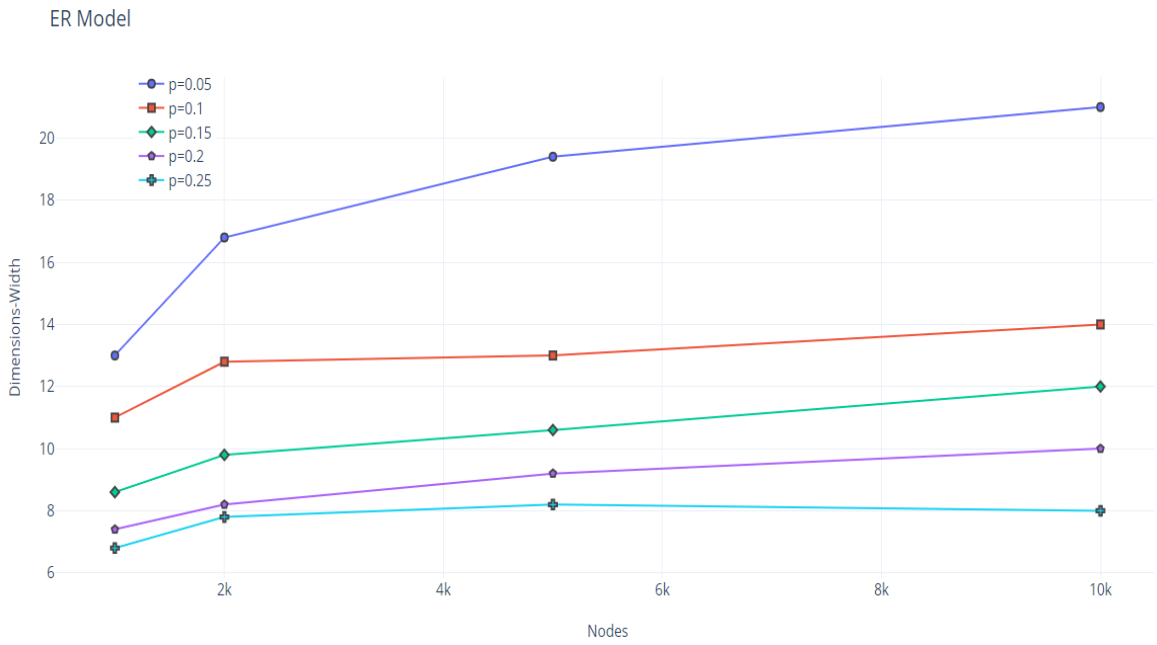


Figure 5.1: Results of the experiments for the Erdős-Rényi model showing the number of dimensions with respect to nodes.

5.3.2 Barabasi-Albert model [78]:

This model is well suited for creating scale-free graphs, which, roughly speaking, are graphs having few nodes having high degree and many nodes with small degree. More formally, in this model the fraction of nodes $P(x)$ having degree x is proportional to

x^{-3} (i.e., $P(x) \sim x^{-3}$). The graph is built by attaching nodes one by one, following a preferential attachment strategy, that is, new nodes are attached to high degree nodes with high probability. The parameters of this model are the number of nodes n and the number of edges initially attached to a newly node inserted in the graph m. Clearly, as far as m increases the density of the graph increases. In our experiments we consider $n = 1000, 2000, 5000, 10000$ and $m = 10, 20, 30, 50, 100$.

In this model the first m nodes are added so that they are incomparable (with no edge connecting them). Hence, by definition, the width of any BA graph in our experiments can not be lower than m. This fact does not influence the experiments in this setting, since we choose values of m that are very low and, consequently, we experiment graphs with a very low density. We discuss more in deep the importance of this initialization step at the end of the section.

The results of our experiments on $k = w_G$ performed on this model are presented in Figure 5.2. In order to have a dominance drawing we need less than 200 dimensions for $m = 50, 100$. In Figure 2 of [59] we can see that with a similar number of dimensions the Fip Ratio is (relatively) small, but higher than 0. Hence, for dense BA graphs our approach seems to give more competitive results than the one described in [59] in terms of both (a) the number of dimensions (i.e., space required to store the information) and (b) the Fip Ratio. On the other hand, as m decreases ($m = 10, 20, 30$) the width of the graphs seems to increase linearly with respect to the number of nodes. Similarly, in [59], when the graph is very sparse ($m = 10, 20, 30$) the Fip Ratio is considerably higher. We conclude this analysis with an interesting observation: Notice that for $m = 100$ in our experiments the width of the graph seems to slowly decrease when the number of nodes increases. In other words, the experiments tell us that - for this (not so) particular case - for dense BA graphs, for all practical purposes, the width can be considered a constant. Additionally, it follows that our approach scales very well for graphs that follow this model.

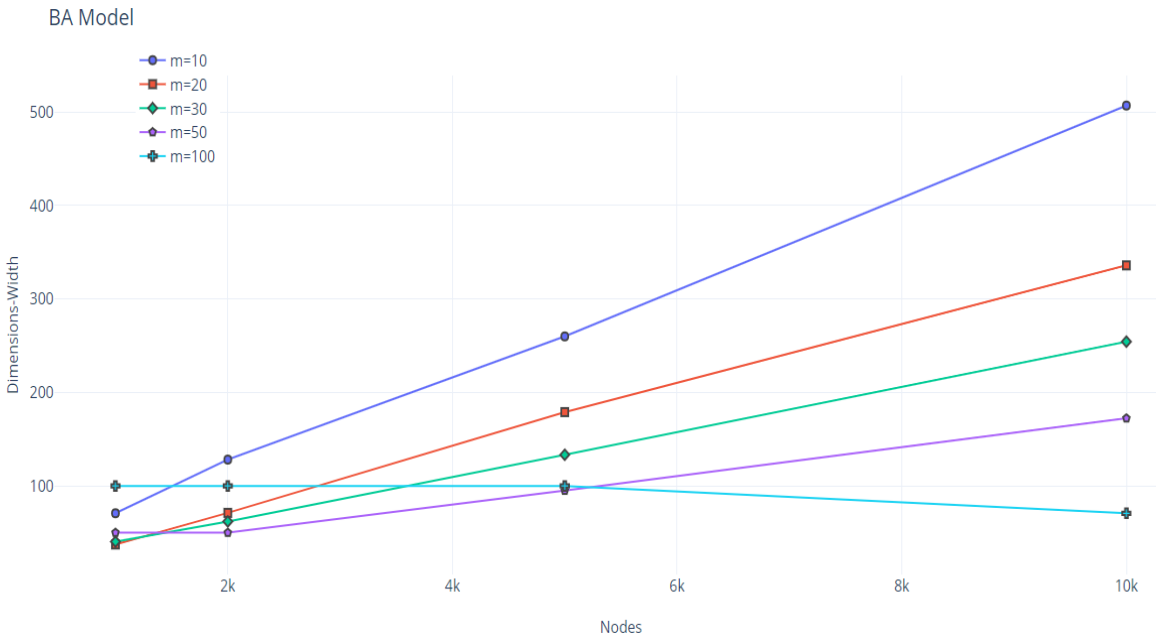


Figure 5.2: Results of the experiments for the $Barabasi - Albert$ model showing the number of dimensions with respect to nodes.

5.3.3 Watts-Strogatz model [79]:

This model represents small world graphs, that is, many sets of nodes are highly connected between them. This model requires a parameter k, which represents the average number of neighbors of the nodes, to be significantly larger than $ln(n)$ to guarantee that a random graph will be connected. Another important parameter is b. It is used to control the "randomness" of the graph, and it influences the average length, $L(G)$ between two nodes (the length of the shortest path connecting them). When b approaches 0 $L(G)$ approaches $n/2k$, whereas when b approaches 1 $L(G)$ approaches $ln(n)/ln(k)$. The random network at $b = 1$ is a poorly clustered, small world where $L(G)$ grows only logarithmically with n [79]. Hence, for this model we need three parameters. The first one is k, and second parameter is $0 \leq b \leq 1$, which measures the randomness of the graph, that is, it affects directly the value of $L(G)$. The last parameter is, of course, the number of nodes n. Our results show that the number of dimensions needed to have a dominance drawing in the case of $b = 0.9$ or $b = 1$ is very high and that it grows linearly with n, as shown in Figure

5.3. Similarly, the Fip Ratio in such cases is much higher than in other cases as indicated in [59]. In order to add to our understanding we should stress again the fact that the WS model has a structure close to the ER model when b is (almost) one. Hence, one would expect that the WS graphs would have a behaviour similar to the ER graphs. Obviously, this is not the case, since the results for the ER graphs are significantly better than the ones for the WS graphs for $b = 1$ and $b = 0.9$ (in both our and their experiments). The reason for this behavior is probably due to the fact that the two models with the specified parameters for the graphs have significantly different density: Indeed, for the values of p (in ER) and k (in WS) considered in [59] and in this work, the WS graphs are significantly less dense than the ER graphs. For example, the sparsest ER graph considered having $n = 10000$ is the one having $p = 0.05$, which has 2.5 million edges, while the most dense WS graph considered for $n = 10000$ has only 250000 edges (which is a factor of 10 less edges). Hence, as a last experiment of this section we decided to experiment with WS and ER graphs that have the same density. As expected, the experiments for the two models give very similar results, as will be discussed later, see Figure 5.5.

In contrast to [59], we believe that analyzing the WS for very high values of b is not interesting, since such graphs are basically random (similar to ER) and not "small worlds" graphs. Hence, we will focus our attention on values of b that are not as large in order to show the different behaviour for $b = 0.3$, $b = 0.5$, and $b = 0.7$. Specifically, we compare our results as shown in Figure 5.4, with the ones presented in Figure 2 of [59] only for $b = 0.5, 0.7$, since in that paper there are no experimental results for the choice $b = 0.3$ (we recall that in that paper the authors presented experimental results only for $n = 10000$). For $k = 10$ and $k = 20$ the Fip Ratio in [59] is equivalent to 0.5, which is a high value, even when using 200 dimensions. Our experimental results show that we obtain dominance drawings with no fips using at most (a) 250 dimensions if $b = 0.7$ and 50 dimensions if $b = 0.5$ when $k = 10$; and (b) 25 dimensions if $b = 0.7$ and 5 dimensions if $b = 0.5$ when $k = 20$. For $k = 30$ and $k = 50$ we can

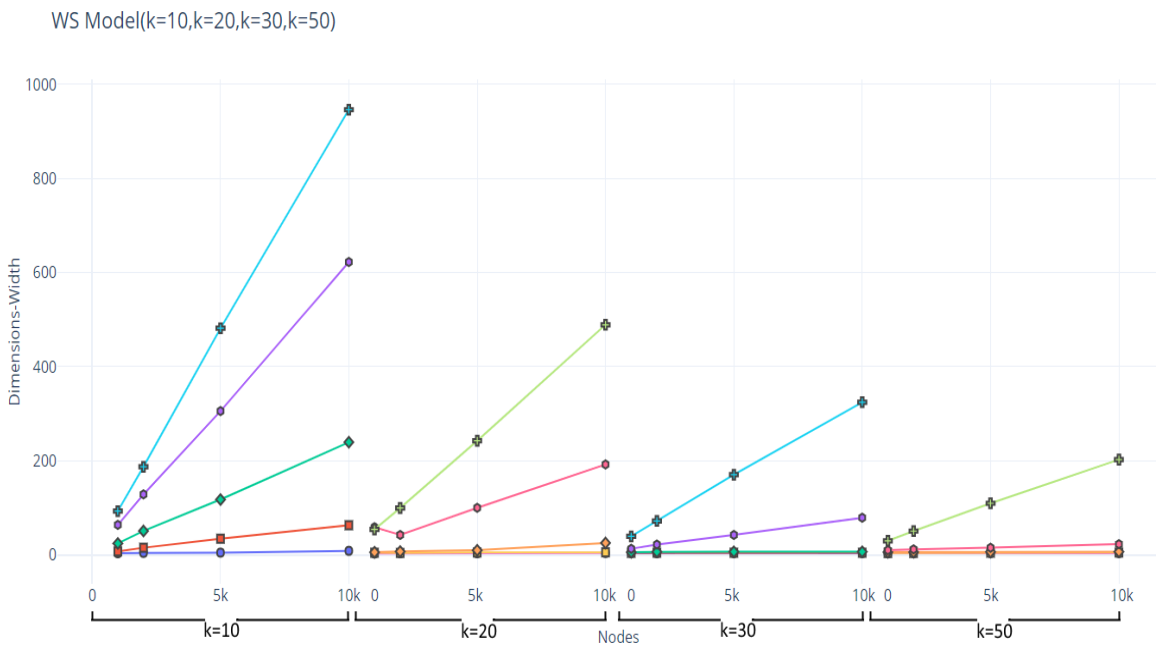


Figure 5.3: Results of the experiments for the Watts-Strogatz model for $k = 10, 20, 30$ and 50 showing the number of dimensions with respect to nodes. The five curves correspond to $b = 1, 0.9, 0.7, 0.5, 0.3$ as seen from top to bottom.

achieve 0 fips by using at most 3 and 7 dimensions, respectively. The corresponding results presented in [59] are also very good, since they obtain a small Fip Ratio, close to 0, for any number of dimensions.

We conclude the discussion on the results for the WS model with the analysis on the scalability of our results as shown in Figure 5.3. For $k = 10$ and $k = 20$ the width grows linearly with n for $b = 0.7, 0.9, 1$, while it is very close to be a constant for $b = 0.5, 0.3$. For $k = 30$ and $k = 50$ the curves seem to be constant for $b = 0.3, 0.5, 0.7$, whereas they seem to grow linearly (with a smaller slope) for $b = 0.9, 1$. These plots show that when the graphs are rather sparse (small k) our techniques scale well for small values of the randomness variable b whereas, when the graphs have a high level of clusterization (high b) they scale well for all the considered densities.

In the results of the experiments depicted in Figure 2 in [59] the Fip Ratio obtained for the ER model is much lower than the Fip Ratio obtained for the same

number of dimensions for the BA model and the WS model. For example, for 50 dimensions the highest Fip Ratio in the ER model is 0.001, while for the BA model it is 0.2 and for the WS model it is 0.7. Also in our experiments, as shown above, the width of the ER graphs is much lower than the width of the BA and WS graphs. However, this is caused because the density of the graphs of these three models using the parameters of [59] is vastly different. In fact, in order to have almost the same density between ER with $p = 0.05$ and the other two models we need to set $m = 250$ for the BA model and $k = 500$ for the WS model. For this reason we decided to perform one more set of experiments in order to validate that, for the same density (i.e., same number of nodes and about same number of edges), the ER graphs have a width comparable to the width of the BA and WS graphs. This experiment is described in the next subsection.

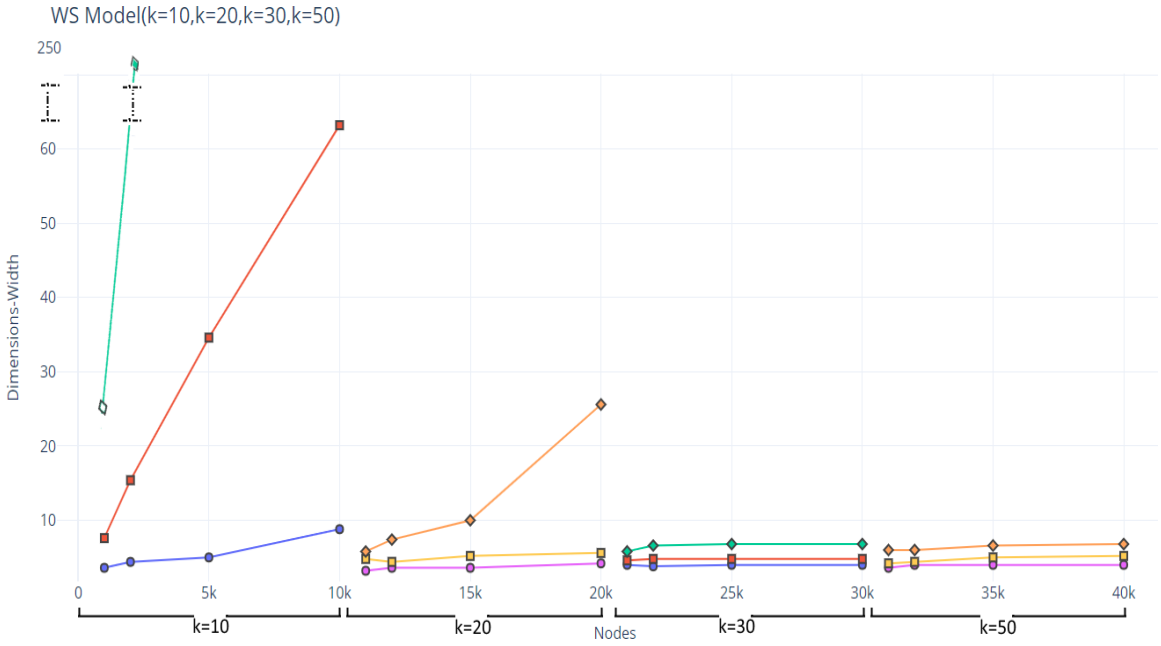


Figure 5.4: Results of the experiments for the Watts-Strogatz model for $k = 10, 20, 30$ and 50 showing the number of dimensions with respect to nodes. The three curves correspond to $b = 0.7, 0.5, 0.3$ as seen from top to bottom.

Same Density Experiment: In order to understand the meaning of the previous experiments, we chose parameters that would create the same density graphs for the three models. Refer to Figure 5.5, where we show experiments on the width of the graphs where the number of nodes is 10000 and the number of edges is 250000 and 2.5 millions.

Before doing a comparison between the BA graphs and the ER graph, where the two graphs have the same density, we highlight the following observation concerning the initialization step of the BA graphs. The BA graphs shown in Figure 5.2 produced in the experiment above are initialized with m incomparable nodes. Hence, for those graphs the width cannot be lower than m, which in this case is equivalent to 250. Notice that for the BA model the initialization step can be performed in many different ways. We performed our experiments initializing the BA graphs slightly differently than before, in order to see if, for a different initialization step, the width of the graph can be lower than m. Namely, in this last experiment we initialize the BA graphs by adding a path connecting the first m nodes. For this possible initialization, the width of the BA graphs can be even lower than 250. This is the case when the number of edges is 2.5 millions, but this is not the case when the number of edges is 250000. In this last case (sparser graphs), the initialization step does not influence the width of the graphs.

Concerning the comparison between the BA and the ER models, we see that for the same value of density and for both types of initialization that we used in our experiments (with $m = 250$ incomparable nodes, where the width cannot be lower than 250, and with a path containing the first m nodes), the width of the BA graphs is higher than the width of the ER graphs.

Concerning the comparison between the WS graphs and the ER graphs, Figure 5.5 shows that the two models have a very similar behaviour when the value of b is close to 1. This fact is expected, since for $b = 1$ the WS graph is, almost the same as an ER graph. The WS graphs have a much better behavior for lower values of b. Our

experiments validate the fact that the much better behaviour of the ER graphs with respect to the WS graphs shown by our experiments and by Figure 2 in [59] is due only to the different value of density used for the two models.

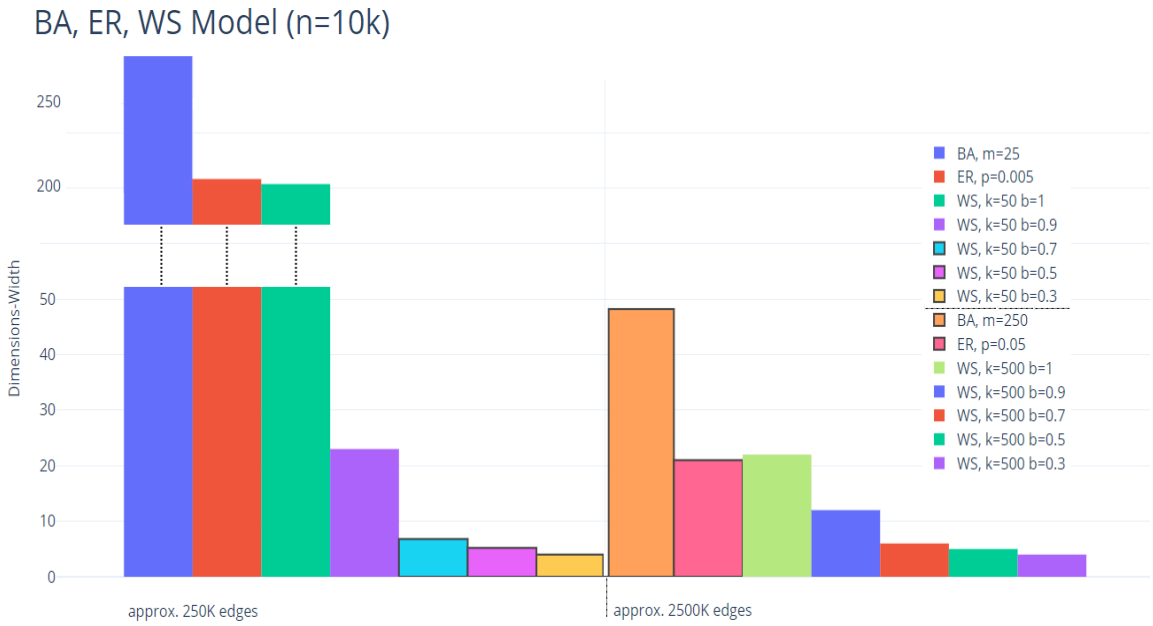


Figure 5.5: Results of the experiments on the width of the graphs where the number of nodes is 10000 and the number of edges is 250000 and 2.5 millions respectively.

The above experimental results on the three graph models were presented as a comparison with the experimental results of [59]. Our experimental results offer on one hand a validation of their results in some cases, but they also give better results in many other cases. However, these models are interesting mostly for undirected graphs or for directed graphs with cycles (i.e, not DAGs). Indeed, DAGs are usually used to describe processes containing some long paths, such as in PERT applications see for example [7, 8], while in the graphs discussed above paths of significant length are not guaranteed by construction. Therefore, in the next section we introduce a new model, the Path-Based DAG model, where the graphs are constructed starting from a number of randomly created paths and where the rest of the edges are added also randomly.

5.4 The Path-Based DAG Model

We introduce a new graph model which is more suitable to DAGs and their applications [80]. We believe that this model is more representative of DAGs that are used in many applications. In this model, graphs are randomly generated based on a number of predefined but randomly created paths. By construction this model has the clear advantage that we know in advance an upper bound on the number of dimensions since the predefined paths clearly provide a set of dimensions. A second advantage is that we can create graphs with significantly different structure in order to explore how structure influences the number of dimensions of DAGs. We shall refer to this model as the *path-based (DAG) model*.

The purpose of this model is to generate a random graph $G = (V, E)$ with n nodes and m edges having a predefined number k of paths. This implies that the number of paths is a clear upper bound on the number of dimensions for these graphs. However, as we add more edges randomly the number of required dimensions decreases, sometimes significantly. The strategy is described by the following two steps: (i) First a set of k paths is randomly generated, where the number k is given as input. (ii) Next we enrich the generated paths by randomly inserting edges one by one until the total number of edges is equal to the requested number of edges m given as an input parameter, i.e., a given density.

In this model we add the edges following a strategy similar to the *Erdős-Rényi model*: the parameters of the construction of a path-based DAG are the number of nodes n, the probability p of the existence of an edge between two nodes, and the number of paths, k. Notice that the total number of edges m in the DAG is implied by n and p, ranged from approximately 25.000 to 12.500.000 edges. In order to obtain a DAG, all edges are oriented from a "lower" numbered node to a "higher" numbered node. In our experiments we consider cases with $n = 1000, 2000, 5000, 10000$ and $p = 0.05, 0.1, 0.15, 0.2, 0.25$. Based on our experience from the experiments performed

in [13] it seems natural to choose the initial number of paths k to be centered at $\sqrt{n}$. That is, we perform experiments assuming that a realistic initial channel/path decomposition of a given DAG $G = (V, E)$ with n nodes would have a number of paths that would be equal to a fraction or a multiple of $\sqrt{n}$.

We performed two types of experiments using DAGs described previously. In the first type the chosen number of predefined paths is fixed to $\sqrt{n}$. We want to assess how the number of dimensions varies with the density of the input graph while the structure remains "fixed". For each case we performed five runs on different randomly generated DAGs with the same parameters starting with $\sqrt{n}$ paths. The results of our experiments, presented in Figure 5.6, show the average number of required dimensions. Our results show that for the case where the parameters create the lowest density (sparsest DAG), i.e., for $p = 0.05$ and $10,000$ nodes, we need at most 21 dimensions in order to have a dominance drawing. A similar observation as for the ER model applies also here: the number of nodes of the graph does not influence drastically the number of required dimensions. This experiment highlights the fact that the number of dimensions is mostly influenced by the density of the graph. Another observation is that the number of dimensions is a lot lower than $\sqrt{n}$, the number of predefined paths used to construct the graphs. Additionally, notice that the results on this experiment are very similar to the results for the ER graphs, a fact that provides a further validation of our model and choice on the number of predefined paths.

5.4.1 Number of predefined paths as a parameter of the model.

In the second type of experiments the chosen number of predefined paths is a multiple or a fraction of $\sqrt{n}$ while the number of nodes fixed at $n = 5000$. Here we want to assess how the number of dimensions varies with a varying structure of the input graph while the node size remains fixed. The number of paths influences heavily the structure of G and it presents a different point of view to the problem of determining

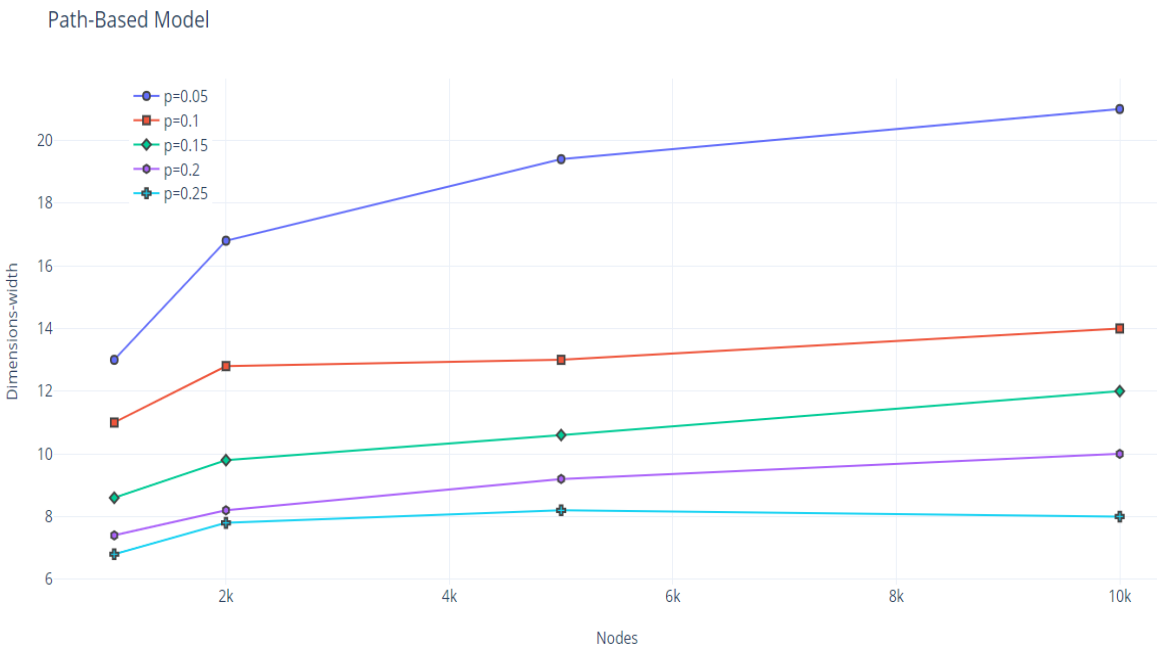


Figure 5.6: Results of the experiments for the path-based model as created with $\sqrt{n}$ predefined paths, showing the number of dimensions with respect to nodes.

a small number of dimensions for a dominance drawing. Specifically, we investigate DAGs constructed following the path-based model with $n = 5000$ nodes considering the following parameters for DAGs: Various small values of p (in order to still assess the influence of density in even sparser DAGs than before) and the number of predefined paths varying from $\sqrt{n}/8$, $\sqrt{n}/4$, $\sqrt{n}/2$, $\sqrt{n}$, $2\sqrt{n}$, $4\sqrt{n}$ to $8\sqrt{n}$ in order to assess how structure influences the number of dimensions. The experimental results of this study are shown in Figure 5.7. Again, for each case we performed five runs on different randomly generated DAGs with the same parameters and the results shown in Figure 5.7, record the average number of dimensions. The curves in this figure provide clear evidence that the structure of a DAG plays an important role in the number of dimensions of a dominance drawing. Of course the different curves shown for the different values of p also show that density is still very important. Clearly, the number of paths (i.e., structure) influences the number of dimensions significantly when dealing with very spare graphs (e.g., $p = 0.005, 0.01$), as shown in Figure 5.7.

On the other hand, if the graphs are rather denser (e.g., $p = 0.03, 0.05$), then the

graph structure seems to be less important. The density is clearly a determining factor. This is demonstrated by another experiment that we performed in order to determine the importance of various parameters in the preliminary stage of our research. We investigated the case where the graph is rather dense i.e., $n = 2000$ and $p = 0.25$. In this scenario, the number of dimensions ranged from 6.2 to 7.8 for all cases of varying number of paths given. This shows that the structure does not play an important role in dense DAGs, which is the opposite to what happens when the graph is rather sparse, as discussed in the previous paragraph.

By examining again the curves in Figure 5.7, we observe that, generally speaking, the number of dimensions required is increasing as the number of predefined paths is increasing, for all p values tested. However, when the number of paths is very large this behaviour becomes almost a constant for the denser graphs of this experiment. Also note that in some extreme cases the number of dimensions may even slightly decrease. For example for $p = 0.05$ we see that when having 140 and 280 paths the number of required dimensions is 20.6 and 23, respectively, while the number drops to 21.6 when the number of paths is 560. Our explanation of this phenomenon is that when the number of predefined paths is very large the average number of nodes in each path is a small constant. Then the other edges, between nodes that are on different paths, are most influential in building longer (and fewer) channels which leads to smaller number of required dimensions.

Moreover, another interesting fact that needs to be highlighted is that this behavior is inversely proportional to the density of the graph. In other words, in denser graphs the curves flatten out faster whereas in sparser graphs the curves continue to rise.

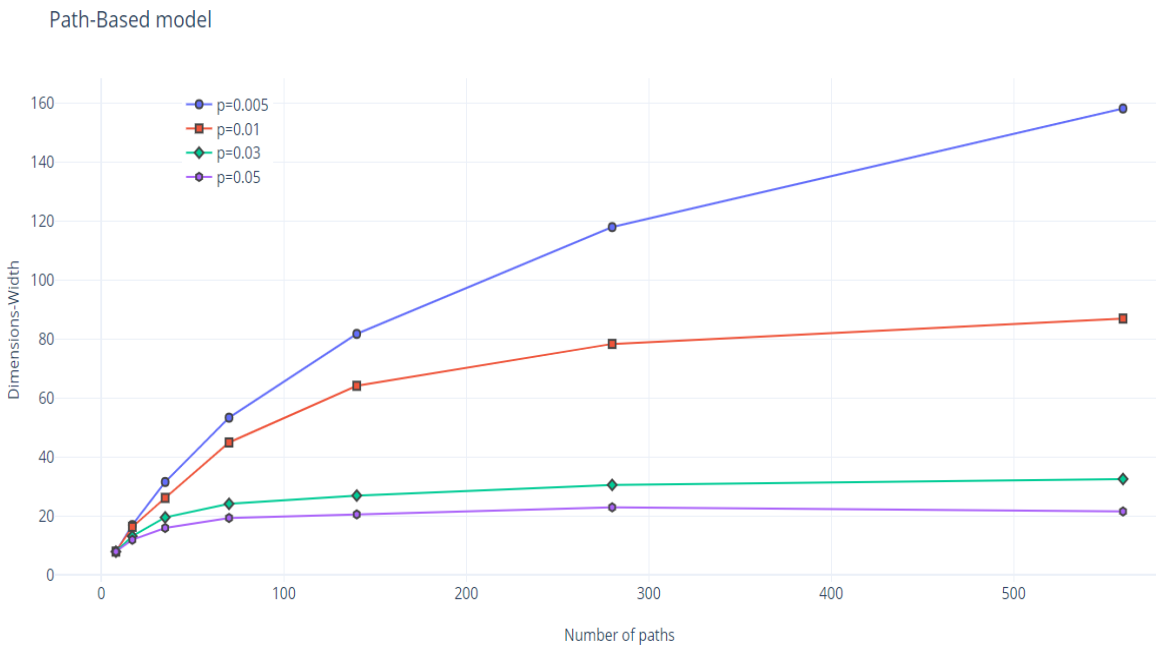


Figure 5.7: Results of the experiments in the number of paths for the path-based model for $n = 5000$ and various p values, showing the number of dimensions with respect to nodes

Discussion: In this chapter we present algorithms that construct a data structure in the k-dimensional space that is based on the concept of Graph Dominance Drawing. Our algorithms construct this data structure in $O(km)$ time, while it can be stored in $O(kn)$ space. Any reachability query can be answered in $O(1)$ time, since it suffices to check only one specific dimension. Furthermore, no "falsely implied paths (fips)" are introduced. The experimental results show that our techniques perform better than the state of the art. Namely, we provide experimental evidence that shows that the number of dimensions k is usually equal to a small fraction of $\sqrt{n}$ (for very sparse graphs) and sometimes it grows as slow as $O(\log n)$. Our extensive experiments on graph models with different structure indicates that when the DAGs are dense k seems to be almost a constant. Furthermore, our experimental results validate the choice on the maximum number of dimensions used in the experimental results of [59].

An interesting key point is that we introduce a new graph model that is more suitable to DAGs and their applications. We show that this model has a clear advantage

that we know in advance an upper bound on the number of dimensions since the predefined paths clearly provide a set of dimensions. A second advantage is that we can create graphs with different structure in order to explore how structure influences the number of dimensions of DAGs.

It will be important that future research investigate fast heuristics that further reduce the number of required dimensions. Since these techniques have significant applications in very large graphs, like in large databases, it is important to investigate this problem when the graph changes dynamically, i.e, insertions of new edges/nodes, deletions, etc. New techniques need to be developed specifically designed for dynamic graphs since the cost of recomputing the data structures every time after an update may be prohibitive. For example, it seems that the insertion of a single edge may not alter the structure of the graph significantly, but in other cases it does, e.g., when it creates a new cycle.

Chapter 6

Graph abstraction techniques for visualizing DAGs based on the Context-Aware Graph

The emergence of social networks and graph databases has led to an increasing interest in analyzing and visualizing networks (graphs) over the recent years. A social network can be defined by a (homogeneous or heterogeneous) graph consisting of a set of nodes (people) linked according to different types of edges (relationships). For example, a heterogeneous graph can have nodes and edges of different types, in contrast to a homogeneous graph. The enhanced information which describes the relationships and the people within the network composes the "semantic" information of the social network or the **context of the graph**. The tendency of people with similar preferences within a social network leads to the formation of clusters or communities. Identifying such communities [81] and the interactions between them [82] is of crucial importance and can be beneficial for numerous applications such as recommendations [83]. In such cases, users are considered similar if there is an overlap in the items consumed. There is a number of works for community detection such as in [84] where the authors proposed new methods for community discovery, including methods based on "betweenness" measures and methods based on modularity optimization. Other approaches, such as in [85] differ completely from the traditional clustering techniques and use in a conjoint way, the information from the social network, represented by

the points of view, and its structural information.

6.1 Which graph to visualize?

The idea of interoperability of social networks and graphs in general, has received significant attention. The purpose of such process is to understand the information flow within the graph, identify various aspects and select specific interpretations of the input graph that can be used for processing or visualization. Toward that direction, the authors of [86] highlight the significance of identifying the correct graph model that users should choose when dealing with large scale graph database (e.g., in database services such as AWS). They emphasize the fact that it is very common to see users make the wrong choice regarding the selection of the suitable graph model, which is a critical action with no easy way to reverse it later. To this respect, they discuss challenges and different aspects and point out a number of properties that one should take into account when dealing with such decisions. Based on their analysis, they conclude that edge properties (multiple edge), graph abstraction and graph partitioning, are the key aspects and criteria in order to understand the underlying graph structure. We also believe that this is the way forward. To this respect, we elaborate on these criteria in order to analyse the original graph, extract a part or a subgraph of the original graph and visualize it accordingly based on a specific use case.

The purpose of this chapter is to address the challenges of interpreting the information (*semantics*) of the graph that can be used also as a graph visualization aid. To this respect, we define the problem and the requirements of that direction, and we present and analyze various techniques to cover such needs. Although we highlight some interesting research questions to be addressed by the broader graph community, our direction is with respect to graph visualization. For this reason, we apply the proposed techniques over our hierarchical drawing framework, as a use case and present some experimental results to verify the validity of such an approach that can

be applied on various applications depending on the different scenarios.

6.1.1 Multiple Network Models

Heterogeneous information networks (HINs), also called heterogeneous graphs, are composed of multiple types of nodes and edges, and contain comprehensive information and rich semantics. Although heterogeneous information networks are ubiquitous, there are not many standard datasets for study, since such heterogeneous information usually exists in different data sources [87]. Intuitively, most real systems include multi-typed interacting objects. For example, a social network website (e.g., Facebook) contains a set of object types, such as users, posts, and tags. Likewise, in a bibliographic database, like DBLP [88], papers are connected together via authors, venues and terms; and in Flickr, photos are linked together via users, groups, tags and comments. In general, these interacting systems can all be modeled as heterogeneous information networks.

6.1.2 Challenges

Although, it is important to show all the information represented by the edges of a graph, there are cases where this is either impossible due to the complexity and the size of the graphs or even misleading since the graph could be very dense. Moreover, human ability to identify patterns is inversely proportional to the size and complexity of graphs. Other reasons are related to privacy protection (e.g., social network graphs). Although it is crucial to visualize the graph in order to have a better understanding of the underlying structure of the network, as already mentioned there are cases which this is almost impossible. To this respect, it is reasonable to validate the cost of graph visualization but with respect to which graph? In such cases, it makes sense to replace the original graph with a subgraph (e.g., using clustering, supernodes, etc.) or a summary, which removes unnecessary details about the original graph topology but retains the mental map of the user. To overcome this, different

approaches imply to hide the unnecessary information [5] by displaying them on demand, skip the redundant information or to apply graph summarization techniques in order to reduce the complexity of the graph on top of sophisticated graph drawing layouts.

Graph summarization has extensive applications such as clustering [89], classification [90], community detection [91], outlier detection [92], [93], pattern set mining [94], finding sources of infection in large graphs [95] and visualization [96], [97], among others. The notion of summarization over graphs or graph summary is not yet well defined. In general graph summarization has five main challenges [98]:

- Data volume.
- Complexity of data.
- Definition of interestingness.
- Evaluation.
- Change over time.

6.2 Context-Aware Graph Abstraction techniques for visualizing Graphs

In this chapter, we propose sophisticated methods for grouping nodes with similar *relationships (i.e., features)* that naturally decompose the graph based on the set of features applied. The proposed approach is motivated by the observation that people within the same community (clusters) have strong feature similarity [99]. According to our hypothesis, a large number of semantically different features implies a co-existence of several overlapping clusters, since a node may exist in more than one clusters based on the context applied. The underlying assumption is that objects within the same clusters are more likely to have similar features than objects in different clusters. By that, we expect to provide answers to the following problems:

- How can we find significantly different clusters for the same graph?
- Can we find unexpected (hidden) clusters that emerge when a specific combination of features is considered based on the context applied?

Let us assume the network schema of a simple social network as illustrated in Figure 6.1c. In such cases, one could categorize edges based on the type of the k relationships and extract the corresponding k simple graphs and then apply the clustering process accordingly. **This approach though, ignores the different simultaneous roles each user may have and can yield misleading results**. More specifically, in our example there are users that are not only co-workers but also friends so a clustering based on a single attribute e.g., work, is a crude approximation of reality. On the contrary, our proposed approach has the ability to recognise the full social structure of the underlying graph and offer a deeper understanding of social interactions if compared to the corresponding graph without attributes. **Moreover, in contrast to similar approaches, our approach significantly differs by the fact that our abstraction techniques will reveal hidden information. By that, we expect to have nodes within the same clusters or summaries although there is no edge between them in the original graph.**

To summarize, our contributions are as follows:

- We focus on summarization and clustering techniques specifically on heterogeneous graphs-networks.
- We create Context-Aware (Social) Graph, defined as an augmented (social) network enhanced by the semantics of the relationships expressed as edge features.
- We offer the ability to analyse the network not only from a structural view but from a semantic perspective.
- We offer a method that changes the semantic structure of the same network based on the context applied.

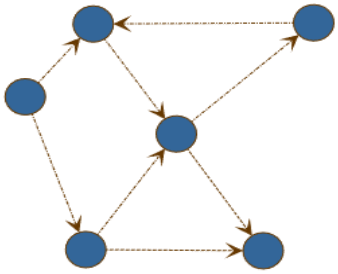
- Our model can detect hidden information by providing clustering results with nodes that are not directly connected.

6.2.1 Problem definition

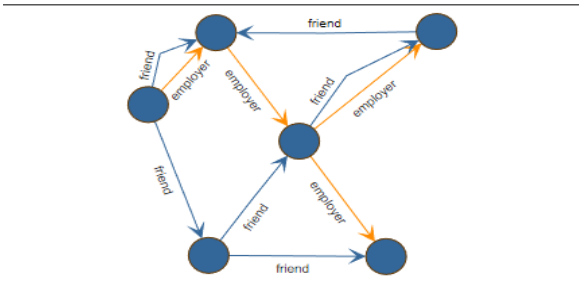
Context of the graph

Let $G = (V, E)$ be a graph of a social network and $V(G)$, $E(G)$ denote the set of vertices and the set of edges of G, respectively. In general, each vertex $v \in V$ can be represented as a distinct point p_v while each edge $e = (v, u) \in E(G)$ as a simple curve connecting p_v and p_u. A common approach is to consider binary relations, i.e., edges between vertices are either present or not, using the corresponding adjacency matrix. In other examples such as signed graphs edges can have a positive or negative sign. According to Wasserman and Faust [100], a graph is consider to be *complex* or *multigraph* if a graph contains loops and/or any pair of nodes is adjacent via more than one line. An example of such graphs are heterogeneous social networks *(HSN)*. More specifically, a heterogeneous social network can have a set of typed nodes (e.g., movies or actors) and typed edges as relations (e.g., friends, colleagues). In contrast to that, manny techniques consider only simple graphs by taking into account only one type of relation which can be derived from complex graphs or by merging multiple edges into single edges and by removing the loops. Although this can be helpful in some cases, we can easily understand that different edges (**i.e., features of the edges**) should be consider simultaneously in order to understand the inter-graph behaviour. For instance, in a social network an important amount of information can be derived indirectly from the different context. Figures 6.1a, 6.1b highlight a deeper understanding of social interactions if compared to the corresponding simple graph without the features of edges.

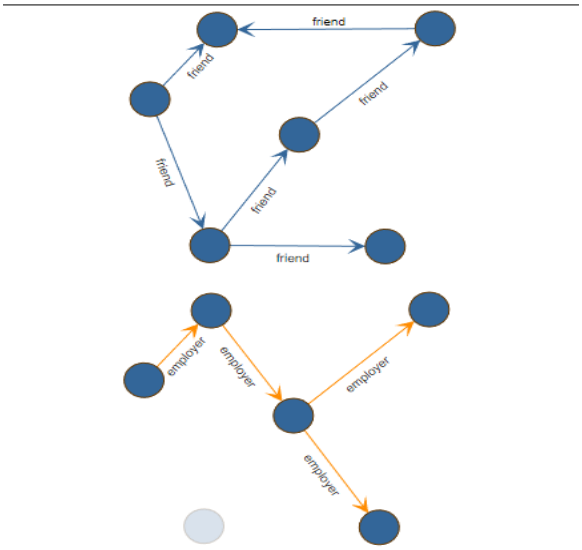
In such graphs the edges may also contain conflicting information e.g., friendship or antagonism which leads to totally different interpretation of the network. More specifically, it is more than common that within the same graph, multiple interactions



(a)

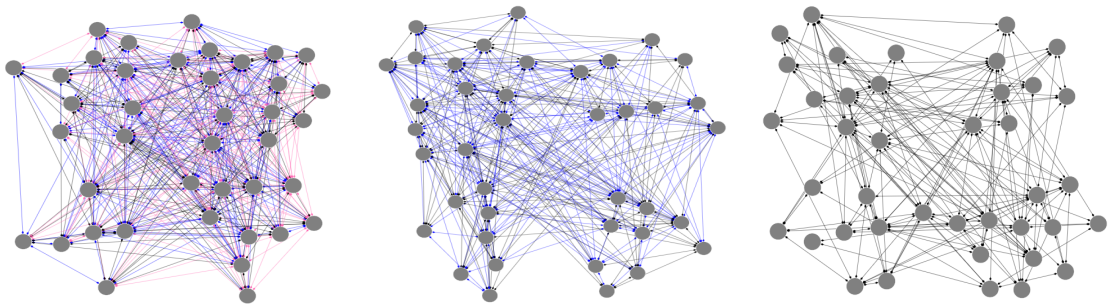


(b)



(c)

Figure 6.1: Graph (a) is a simplified representation of a social network. Graph (b) illustrates the underlying structure by taking into account all the features. Part (c) is an alternative representation of the simple graphs based on the features.



(a) (b) (c)

Figure 6.2: An example of a small social network graph: Part (a) shows the representation of a social network based on 3 different features. Part (b) and (c), show the resulting graph by selecting only two and one feature, respectively.

of different kinds of relations coexist simultaneously e.g., rating, degree of confidence, and can be mapped for the same vertex pair. In Figure 6.1c we see such a case where the simplified graph can be extended to represent the different features of the edges.

In order to better understand the various relationships than coexist within the same graph we highlight the following example. Let us assume a small social network consisting of 40 nodes and a number of edges with 3 different features f_1, f_2, f_3 indicating friendship, family and working relationship, as shown in Figure 6.2. The different colors of the edges represent the various features that coexist within this graph. Based on the features selected, we can extract the corresponding graphs, as shown in Figures 6.2b, 6.2c where we have the original graph as extracted when taking into account features f_2, f_3 and f_3 respectively. To this respect, by using the semantic information we can define the *context* of the graph as the aggregation of the $features$ of the edges. As expected, the different features change the semantic state of the network, since the various features can create different *semantic graphs* of the same original graph.

Features of edges of the graph

We model a social network as a graph $G(V, E)$, where V is a non-empty set of vertices representing users and E is a set of edges representing the relationships among the

Table 6.1: Various features f of F_E associated with each edge e of E.

	Features			
Edges	$Feature_1$	$Feature_2$	...	$Feature_k$
$Edge_1$	$score_{1,1}$	$score_{1,2}$	...	$score_{1,k}$
$Edge_2$	$score_{2,1}$	$score_{2,2}$	...	$score_{2,k}$
...	...	...	...	...
$Edge_i$	$score_{i,1}$	$score_{i,2}$	...	$score_{i,k}$

$\rightarrow$ multi-contextual edge $s(e)$.

users. Let v, u be two vertices of V and $e(v, u)$ be the edge defined by v, u. We denote v, u *neighbors* if $e \in E$. Moreover, let us also denote F_E as a non-empty set of features of the edges. More specifically, by using a $features$ set F_E, that contains k features, each edge $e \in E$ is associated with some number of k' features $f_e \in F_E$, where $k' <= k$, as shown in Table 6.1.

Moreover, we assign a $score(e, f)$, which can be either a binary value (in cases where a feature f_j is present, or not, in edge e_i) or can be expressed as an edge-weight w for a specific feature, where $w \in W\{w_1, ..., w_z\}$. More specifically, the *score* is given by:

1. $Score(e_i, f_k)$=$\{E \times F\}$, where no weight about a feature is given.

2. $Score(e_i, f_k, w_z)$= $\{E \times F \times W\}$, where feature weight is explicitly given.

In the simplest scenario, we have only the first case, where a feature can be expressed only as binary value. Let us recall the previous example of Figure 6.1. Note that each edge of the original graph, may have a number of semantic edges i.e., relationships based on the corresponding features. To this respect, an edge can be also defined as a *multi-semantic edge* or a *multi-contextual edge*.

Multi-contextual edge

Given the semantic edges $s(e)$ as shown in Table 6.1, we can model the multi-contextual edge as follows: As explained in the previous section, each edge $e \in E$ in the graph has a number of features that define the various semantic edges $s(e)$. In other words, each semantic edge $s(e)$, corresponds to a specific feature related to that edge e. To this respect, the edge e can be modeled as a $multi - contextual$ edge, by taking into account the various semantic edges that coexists simultaneously.

Definition 10 (Context-Aware Social Graph) *Given a set of vertices V, a set of edges E and the set of semantic edges SE, the multi-contextual edge graph is the semantically enriched graph of the original graph G. The* ***Context-Aware Social Graph*** *$G(C)$ can be defined as the semantic sub-graph of the multi-contextual edge graph by selecting only a subset of the semantic edges, based on the context C applied.*

Context selection

Choosing the correct context C, is of crucial importance. We highlight the following approaches:

a: *Baseline approach.* The creation and evaluation of clustering techniques over datasets is a critical and difficult task, since no correct answer can be given [101]. The Baseline approach implies that in order to use the full knowledge of the Context-Aware Social Graph, we need to take into account all the features for each edge $C = SE$. As expected, this approach would be extremely inefficient and expensive in execution time for large scale graphs.

b: *User defined context.* As an alternative to the baseline approach, the user can select only a subset of the semantic edges $SE' \subset SE$. This subset can be changed accordingly based on the context applied. By that, one can easily understand that the same graph can give different clustering results based on its Context-Aware Social Graph and the context applied.

c: *Randomly selected context.* Similar to User defined context, we can choose a random subset of the semantic edges $SE' \subset SE$. This strategy can be also applied in cases where no such information is explicitly provided. Furthermore, we also introduce this approach to evaluate its performance compared to the user defined context.

Methodology

Our basic idea is motivated by the hypothesis that semantic clusters can emerge when a specific combination of semantically extracted graphs are taken into account. This indeed is valid when we deal with a large number of semantically different features which may lead to several overlapping clusters. Using the notion of Context as defined previously, we aim to provide an interesting perspective on discovering representative

clustering in complex graphs. More specifically, we use such information in an aggregated way in order to construct the Context-Aware Social Graph and create semantic clusters. The diagram of the architecture is shown in the Figure 6.3.

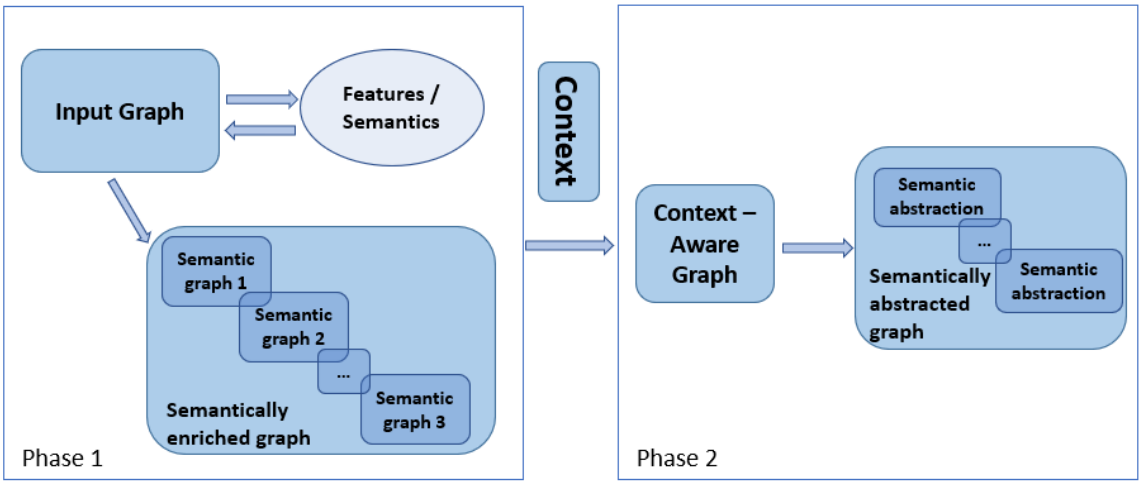
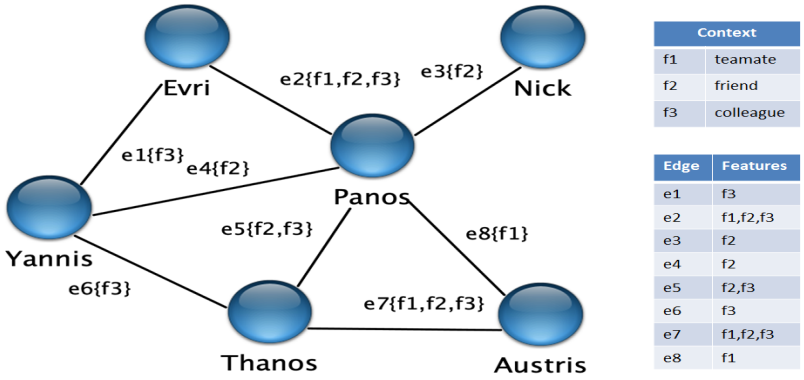


Figure 6.3: The semantics of the graph are identified and the semantically enriched graph is created. Based on the semantic graphs and context applied, we create the Context-Aware Social Graph and extract the semantically abstracted graph.

Algorithmic Example

For this example we use a small social network consisting of 6 users (nodes) and 8 relationships (edges), as shown in Figure 6.4. This network contains a context with features related to $\{f1$:teamate, $f2$:friend, $f3$:colleague$\}$. Note that each edge in the social graph may have more than one feature. For example edge $e5$, which connects users *Panos* and *Thanos*, contains information about features $f2$ and $f3$ while edge $e8$, which connects users *Panos* and *Austris*, contains information related only to $f1$.



Context	
f1	teamate
f2	friend
f3	colleague

Edge	Features
e1	f3
e2	f1,f2,f3
e3	f2
e4	f2
e5	f2,f3
e6	f3
e7	f1,f2,f3
e8	f1

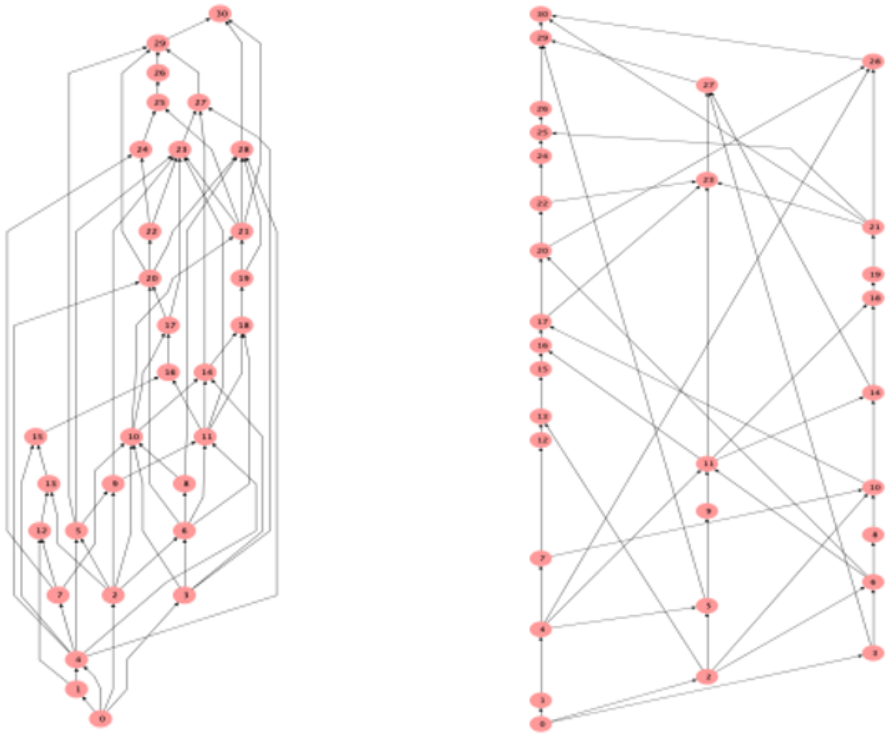
Figure 6.4: A social network with a set of features as context.

6.3 A use case over hierarchical drawings

In chapter 4, we presented and evaluated a set of drawing variants over the *Path based Framework* (PBF). The idea behind these variant techniques, is to progressively abstracting the edges in order to have different visualization results while having the same transitive closure as the input graph. For example, in Figure 6.5b we see the drawing result of the baseline variant, Variant 0. Let us recall these variants. In abstract, Variant 1 is where we denote by jumping cross edge an edge e= (u,v) such that $|X(v) - X(u)| > 1$ and also place a bend on every jumping cross edge of Γ. Refer to Figure 6.6a where, for example, the jumping cross edge e= (7,10) has a bend. Other examples such as Variants 2 and 3 attempt to abstract edges of nodes that are in the same path and edges with common source or target. **This set of variants was the main motivation for our proposed approach.** Similar to these variants we will use the same methodology combined with the semantic information. Figure 6.6b shows a graph drawing as computed by Variant 4.

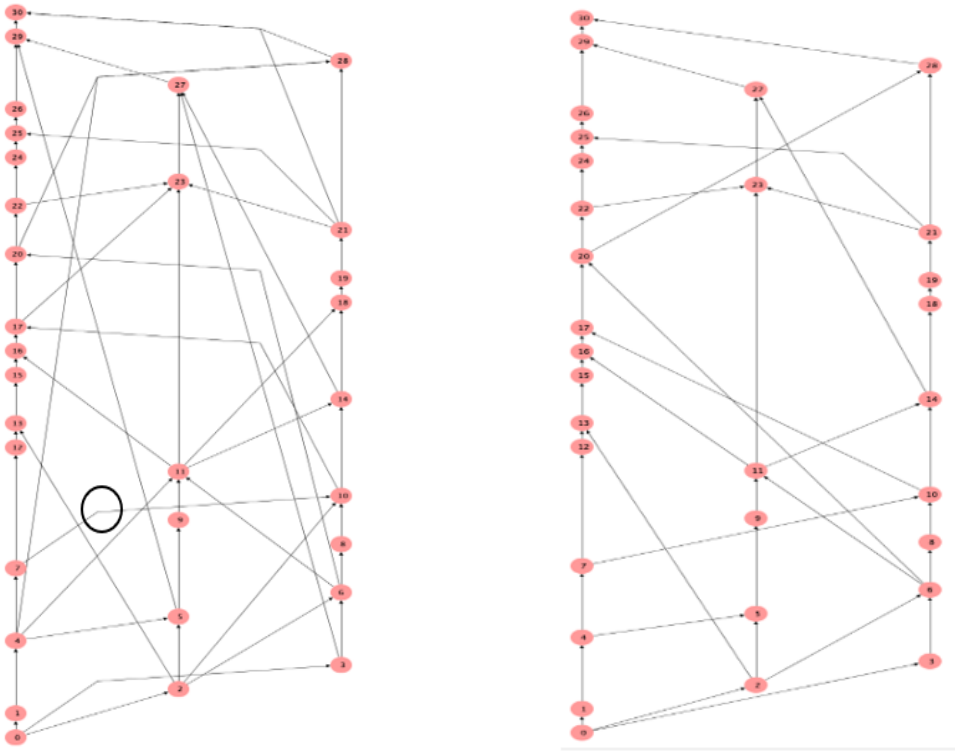
6.3.1 Visualization in Reverse

Although it is may be desirable to show all the information represented by the edges of graphs, such drawings may be confusing or even misleading due to the size. For example, in a vast majority of applications domains, such as graph databases and



(a) (b)

Figure 6.5: In (a) we show the drawing of graph G, as computer by Tom Sawyer Perspectives. In (b) we show the drawing Γ based on G computed by Algorithm PB-Draw.



(a) (b)

Figure 6.6: In (a), (b) we show the drawings Γ based on G computed by variant1, variant4 respectively of Algorithm PB-Draw.

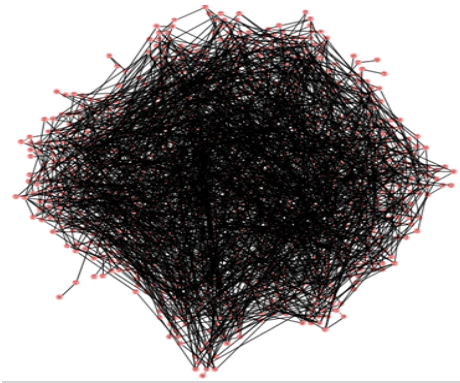


Figure 6.7: The "Hairball Effect" of a graph consisting of 1000 nodes and 2752 edges.

big data, the graphs are very large and the traditional visualization techniques are not applicable due to the complexity and the size and density of the graphs. This indeed is valid, not only in web-scale but also in smaller cases, such as in citations graphs, small social networks, and real-world graphs. The visualization of such graphs produces drawings similar to a "hairbal" (see Figure 6.7). The "Hairball Effect" is a commonly encountered phenomenon in graph visualization. Moreover, human ability to identify patterns is inversely proportional to the size and (visualization) complexity of graphs. In such cases, it makes sense not to show the original graph but replace it with a new representative graph based on specific points of interest. This new graph is a subset of the edges and/or nodes, thus it reduces the visual complexity of the resulting drawing, and the memory requirements such as for storing the transitivity information in the case where the selected point of interest is transitivity. In other words, applying such methods (i.e., abstraction techniques) can be a prepossessing step for further reduction of the visual complexity of the original graph and then visualizing this new graph instead of the original.

6.3.2 Apply semantic Summaries over PBF

Let us focus on challenge of *Definition of interestingness* [98], as described in a previous section. By taking into account this challenge, a summary or a cluster should involve extracting interesting information. As expected, the definition of interest can

vary significantly since most of the cases are subjective and dependent on the application. To this respect, we focus on generating summaries by identifying the most interesting nodes within an area, based on the topology, or produce summaries by exploiting semantic information using the hierarchical drawings as introduced in the previous chapters.

We introduce the following semantic abstraction strategies:

1. Based on the topology of a drawing Γ of G. *(Semantic Summaries based on neighborhoods)*

2. Based on the semantics of the edges *(Semantic Clustering based on cross and jumping cross edges)*

The idea behind our techniques is based on identifying what is important. To this respect, we extract this information by using the characteristics of the graph. Note that we do not simply use the topology of the graph but we take into account the different aspects of an edge or a node. For example, an edge can express friendship or working relation. More specifically, in PBF framework as shown in Figure 6.6b, we see that some edges were removed (or hidden) based on the transitivity, while also maintaining the complete reachability information of a graph. In our case, we choose the semantics of the graph as *Point of Graph Interest (PoGI)*, which can be system based or be specified by the user. Let us focus again on the PBF framework which will be used as a baseline. In this scenario we understand that both of the strategies are based on the same semantic information which in our case is reachability (or transitivity). More specifically the aforementioned strategies are detailed as follows:

1. Semantic Summaries based on neighborhoods. We use the topology of the produced drawing Γ in order to detect/extract summaries using the relative distance of nodes in the same path. Towards that direction, we introduce the following rule: *"All sequential nodes in the same path can create a new summary"*. The idea behind

this is simple and can be justified by the fact that such nodes naturally create a group (summary) since they belong to the same neighborhood. The produced summary will replace the original nodes, thus it can be also called a semantic supernode. The supernode will include all the nodes and will be placed in the original Cartesian position of the highest node (in topological order), which contains a cross edge. Any previous cross edges will be placed in the new position of the supernode. Algorithm 12, $Define\_topology\_neighborhoods$, describes this technique, as shown in Figure 6.8.

Algorithm 12 Define_topology_neighborhoods

Input: A path decomposition $S_p = (P_1,...,P_k)$ of a DAG G; a path based hierarchical drawing Γ_o according to Variant 0
Output: A path based hierarchical drawing Γ_s according to Summaries.

//Defining the summaries
For any vertex v drawn in Γ_o:
Let P_i be the path of v and X(v) , Y(v) its x, y-coordinates respectively
For every vertex $u, v \in V$ belonging to the decomposition path P_i

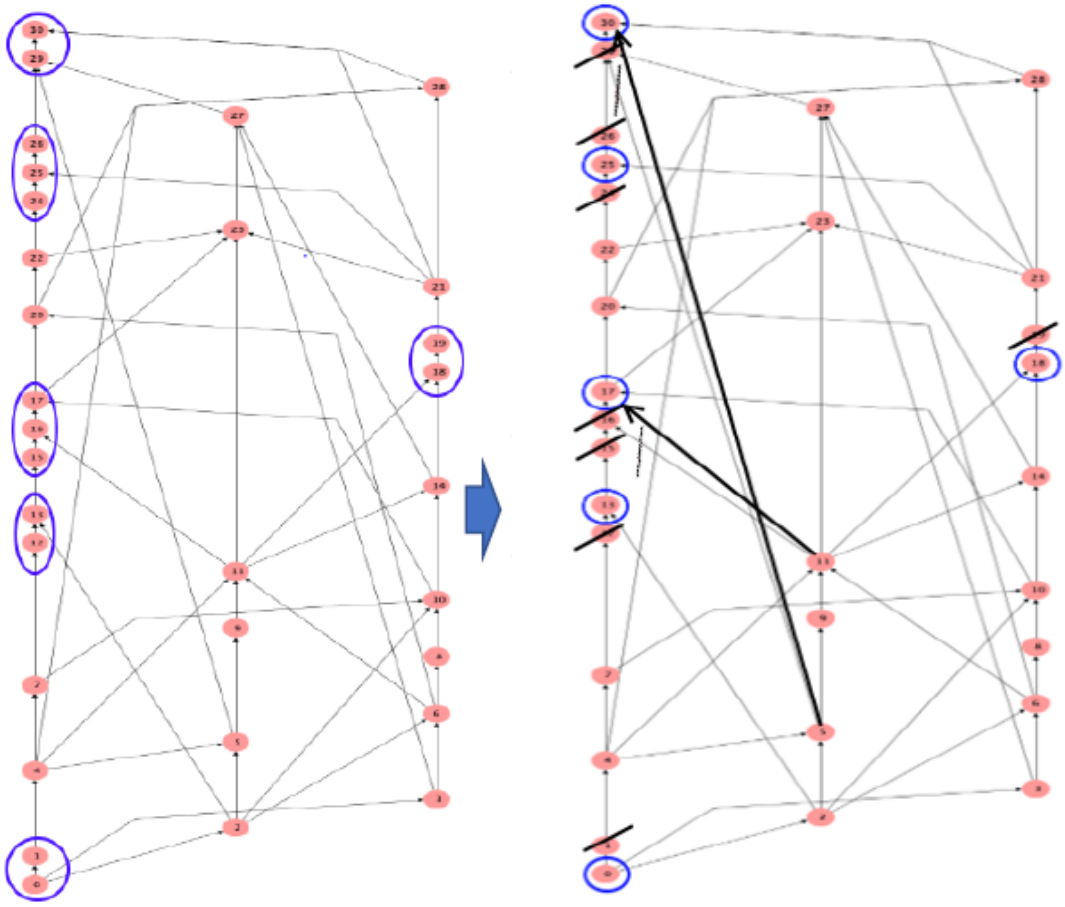
a: If Y(u) – Y(v) < 1: add u,v to same summary

b: Else: create new summary

//Create the position of the summaries
For each summary S_n in S:
For every vertex v in S_n
Let X(Sn), Y(Sn) be the x, y-coordinates of summary S_n

a: If X(Sn), Y(Sn) are undefined: X(Sn)=X(v0), Y(Sn)=Y(v0), v0 is the first vertex in S_n

b: Else: v has cross edge: Y(Sn)=Y(v)



(a) (b)

Figure 6.8: In (a) we show the drawing Γ based on G and the summaries based on neighborhoods technique marked as blue circles. In (b) we show the final drawing based on computed summaries.

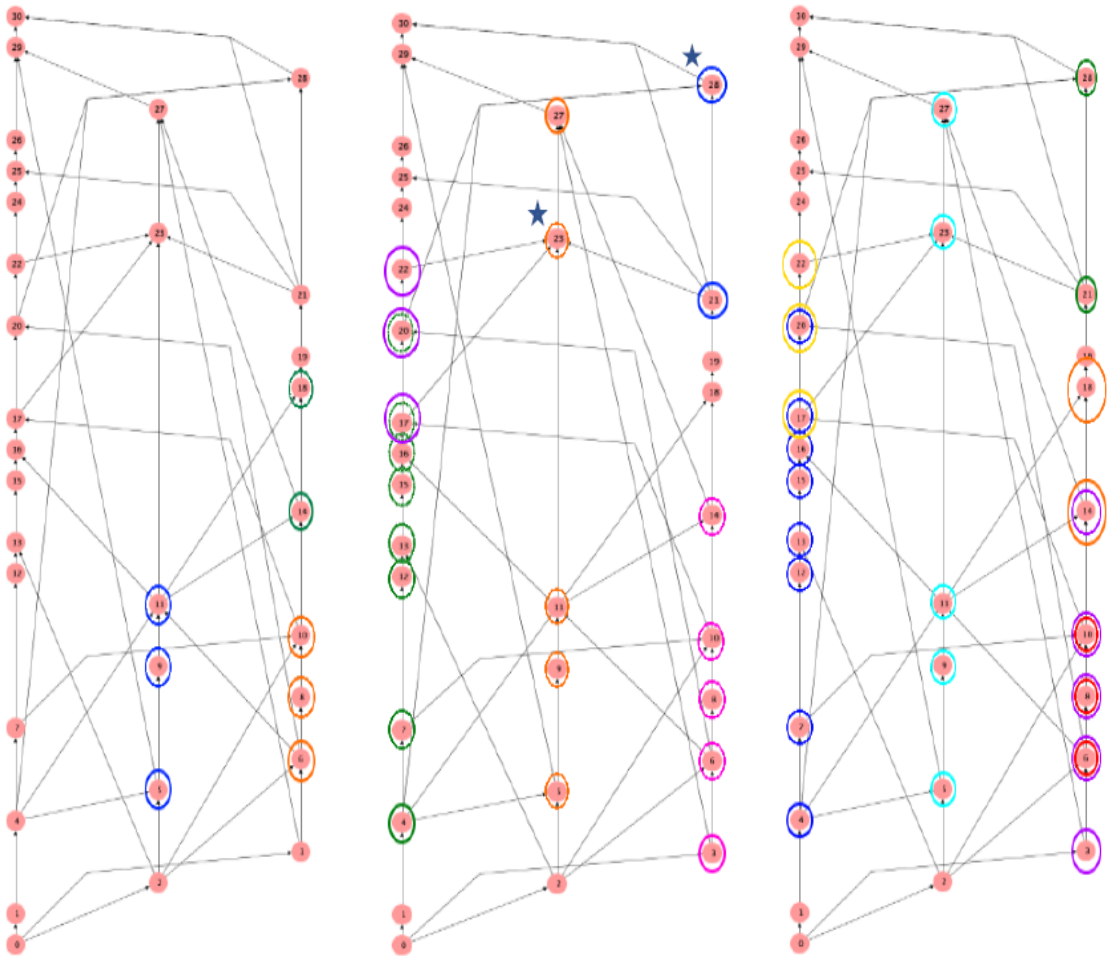
2. Semantic Clustering based on reachability. Based on the concepts of Variants 2 and 3 we use the reachability criterion and transitivity as semantics, in order to introduce the new abstraction strategy. In this strategy we can distinguish two different techniques:

a: One for edges with common source.

b: One for edges with common target nodes as implied by the corresponding variants.

Let us focus on case (a). We introduce the following rule: *"All sequential nodes in the same path that share common source node can create a new cluster"*. Similar to previous abstraction techniques, the idea is to create clusters that semantically group nodes that belong to the same semantic neighborhood, which in our case is expressed by the transitivity. In the simplest scenario, the cluster will distinguish the original nodes, thus it can be also called a semantic cluster. This will include all the nodes and mark them accordingly based on the cluster they belong, revealing their source node. Figure 6.9a shows an example where we have 3 different semantic clusters based on this technique. Technique b, can be defined in a similar manner by taking into account the edges with common target nodes, as shown in Figure 6.9b. Moreover, we can also apply both of these two techniques, as shown in Figure 6.9c, where we see that nodes may exist in more than one cluster.

Choosing the correct cluster is completely depended on the *Point of Graph Interest (PoGI)* and can extract different information upon request, using the same input graph. For example different semantic strategies or techniques, can give different semantic clusters. In more advanced scenarios, we can also define a different semantic rule than transitivity for graphs with edges that contain this information. The idea behind this, has a vast of application domains such as fraud detection and exploitation of hidden information. It also worth mentioning that the semantic clusters can reveal hidden information, as shown in Figure 6.9b. For example *nodes 23* and *28* may have



(a) (b) (c)

Figure 6.9: The 3 different semantic clusters as computed by the semantic techniques of rule 1. In (a) we show the semantic clusters over drawing Γ as computed by the semantic technique (a), in b as computed by the semantic technique (a), while in c as computed by the semantic technique (a) and (b).

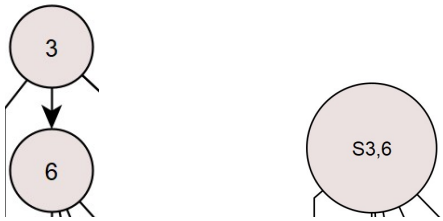


Figure 6.10: Grouping of consecutive nodes into a super node.

some underlying connection although this is not clear by investigating the first figure. This indeed is valid since both nodes are target nodes and their semantic clusters based on technique b contain overlapping nodes.

6.3.3 Experimental analysis

Semantic Summaries based on neighborhoods (Topology neighborhoods)

In order to apply our strategy, first we need to introduce a way to display them according to our use case. In this scenario we introduce the notion of *super nodes*. A super node, contains all the consecutive nodes that might appear in each path, plus it has twice the size of the regular node. Super node's label demonstrates the nodes within the summary. For instance, in the Figure 6.10, the super node contains the nodes 3 and 6. Additionally, Super node's coordinates are calculated based on the last node that has a *cross* or *jumping cross* edge, as described previously.

In the first step of the pseudo code (Algorithm 13), Topology Neighborhoods are defined. The algorithm iterates through the nodes of the graph and searches for nodes that belong in the same path. If these nodes are consecutive then we add them in the same summary, if not, we create a new summary for each node.

Algorithm 13 Finding Topology Neighborhoods

Input a path decomposition Sp = (P1,...,Pk) of a DAG G; a path based hierarchical drawing Γ_0 according to Variant 0

Output a path based hierarchical drawing Γ_s according to Semantic Abstraction technique

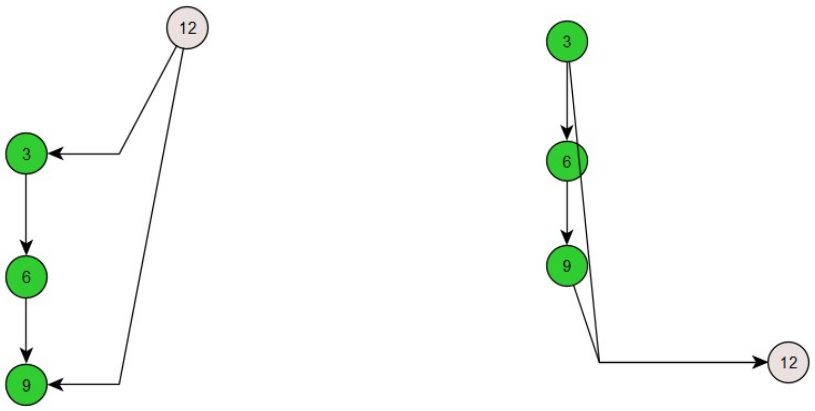
```
// 1.Define Neighborhoods
for every Path P in G do
    for each pair (u,v) of vertices in P do
        Let (X(v),Y(v)),(X(u),Y(u)) be the x-y coordinates of vertices v,u respectively
        if |Y(v) − Y(u)| == 1 then
            Add u,v in the same neighborhood
            if u have a cross edge and Y(u) > Y(v) then
                Update Summary y-coordinate
        else
            Create new Summary

// 2. Fix edges and bends
for each edge e=(u,v) in Γ0 do
    Let S_u, S_v be the Topology Summaries of u,v respectively
    Add in S'_u s targets the Summary S_v
    Add in S'_v s sources the Summary S_u
    if X(S_u) ≠ X(S_v) then
        Add a bend since this is a crossing edge
    else if |Y(S_v) − Y(S_u)| > 1 & there are internal Summaries then
        Add two bends since this is a path transitive edge
```

In the second path of the pseudo code we fix the position of the edges. Based on that, we add bends so there is no overlap between nodes and edges. This is done in a similar manner to the original algorithm, by checking the highest node in topological order that a cross edge.

Semantic Clustering based on reachability

This strategy is based on cross and jumping cross edges. More specifically, a semantic cluster contains two nodes that exist in the same path and have a common cluster or target. For simplicity, let us refer to these nodes as boundaries, as they indeed define an area since they contain the first and last node of the cluster. We also refer to the



(a) Cluster defined by common source. (b) Cluster defined by common target.

Figure 6.11: Various examples of semantic clusters.

common source/target as the **core** node of the semantic cluster. A semantic cluster also contains the internal nodes of these boundaries. For instance, Figure 6.11a, shows an example where nodes 3, 6 and 9 create a semantic cluster because nodes 3 and 9 (boundaries) are connected with node 12 (common source). Similar to that, Figure 6.11b shows an example where nodes 3, 6 and 9 create a semantic cluster since nodes 3 and 9 reach node 12. In both examples, the core node of both Semantic Clusters is node 12.

By taking into account the intercluster nodes defined by the semantic clusters, we define the *Semantic cluster Channel*. More specifically, a Semantic Cluster Channel is an ordered list containing all the Semantic clusters existing in a path. The clusters are sorted based on the Y-coordinate of their first node. Semantic Cluster Channels are very important in finding overlapping clusters mentioned below.

Summaries	Color
0	Gray (default)
1	Green
2-3	Red
4-5	Brown
6-7	Yellow
8-9	Orange
10-11	Blue
12+	Purple

Table 6.2: Various colors assigned to nodes based on the number of Clusters.

Algorithm 14 Find Semantic Clusters

Input a path hierarchical drawing Γ_0 according to Variant 0, a DAG G
Output a path based hierarchical drawing Γ_s according to Semantic Clusters

// Defining Semantic Clusters
for every vertex u in G **do**
Let S be a Map $< K, V >$ where K is the x-coordinates of nodes that reach to u (sources) and V is a Semantic Cluster containing all those nodes
Let T be a Map $< K, V >$ where K is the x-coordinates of nodes that u points to (targets) and V is a Semantic Cluster containing all those nodes
for every vertex s that reaches to u (source) **do**
Add s in Cluster $S[X(s)]$
for every vertex t that u points to (target) **do**
Add t in Cluster $T[X(t)]$
for every Cluster sum in S,T **do**
if sum contains more than one nodes **then**
Find internal nodes between sum.FirstNode and sum.LastNode
Add this sum to the Semantic Path

In our case, nodes that belong to a semantic Cluster are represented with a different colour. The colour is assigned based on the **number of semantic Clusters, each node belongs to**. In more details, Table 6.2 below, shows the various colors as a factor of the node and the number of Clusters it belongs.

Figure 6.12: An example of two overlapping semantic clusters.

Overlapping Clusters

Overlapping Clusters are the pairs of semantic Clusters that have common nodes. For instance, Figure 6.12 demonstrates two overlapping semantic Clusters. $Cluster_1$ with nodes 3, 6, 9 and 2 and $Cluster_2$ with nodes 6, 9, 2, 4 and 0. These two Clusters are overlapping since they have three common nodes, nodes 6, 9 and 2.

We also filter out Semantic Clusters using a threshold. The algorithm loops through all of the Semantic Clusters and finds all of the colliding cluster for each Semantic Cluster. Moreover, the algorithm introduces two options on how to filter the outcome. The first option (relaxed search) prints the pairs of Semantic Clusters that at least one of them satisfies the given threshold. The second option, (rigorous search) searches for pairs of Semantic Clusters that both satisfy the threshold given by the user. The complete pseudocode is described in Algorithm 15.

Overlapping Clusters can reveal some underlying connection between nodes, we refer to this connection as **"hidden connection"**. Two **core** nodes form a hidden connection, based on two criteria:

- The nodes are not connected.
- The corresponding Semantic Clusters satisfy the threshold based on the search (relaxed / rigorous).

Algorithm 15 Find Overlapping Clusters

Input a path hierarchical drawing Γ_s with the Semantic Clusters, a threshold percentage, a boolean value denoting if the algorithm should search for rigorous connections or not

Output Pair of clusters that satisfy the overlapping threshold

for every Semantic Cluster C **do**
  Let P be the Semantic Path of C
  for every other Semantic Cluster T in P **do**
    if C.LastNode doesn't collide with T.firstNode **then**
      move C to the next Cluster in P.
    else
      Let $C_e \leftarrow$the Cluster that has $min(Y(C.LastNode), Y(T.LastNode))$
      Find common nodes starting from T.FirstNode and ending at $C_e.LastNode$
      if Common nodes between C & T satisfy the overlapping threshold **then**
        Use DFS with Priority Queue to check if the core nodes form a hidden connection.

A priority queue is used in a DFS implementation to determine whether two nodes are connected directly or indirectly. An example of hidden connection is demonstrated in Figure 6.13, where nodes 16 and 9 form a hidden connection as Target nodes in two semantic clusters containing the nodes <6,4,2>. Because the two semantic clusters in the example below are identical, they satisfy any threshold given by the user.

Threshold

In our experiments we elaborated on various threshold values. More specifically, Figure 6.14a, shows the influence threshold as it applies to a graph that is 6 percent complete, whereas Figure 6.14b shows the influence threshold as it applies to a denser graph that is 24 percent complete. As the threshold rises, we observe that, the number of hidden connections gradually decreases in both figures.

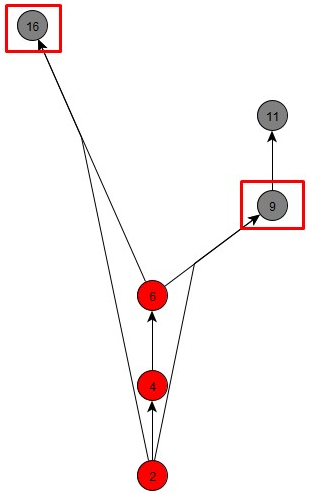
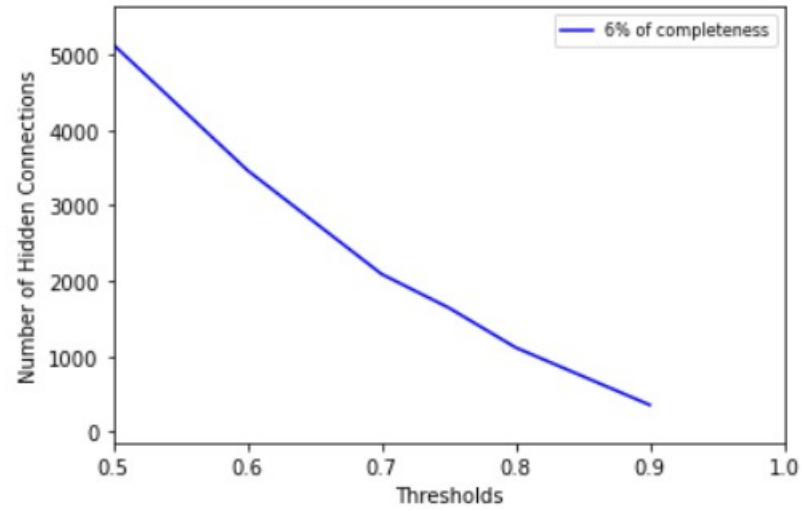


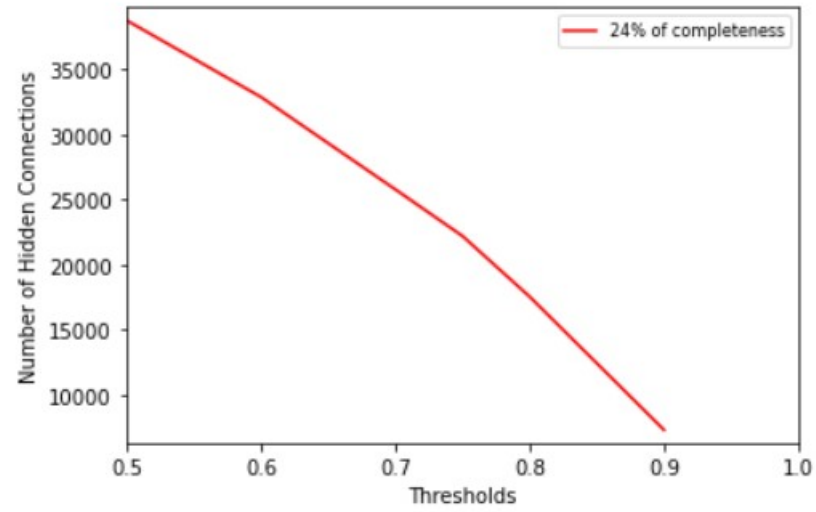
Figure 6.13: An example of hidden information detected by our algorithm.

Results Comparison

In this subsection we provide insights about the reduction of nodes and edges based on our techniques over 9 different graphs. The randomly generated graphs were constructed by our graph generator on a predefined number of 60 paths, using 3 different number of nodes (7000, 10000 and 15000) and a number of edges based on various percentages of completeness as shown below. The results show that for a threshold value of 8%, our technique achieves an average nodes reduction of approximately 48%. On the other hand, the edge reduction is not so promising since the percentage of reduction is less than 6%. Nevertheless, this is an expected behaviour since we do not provide any improvement as regards the edge abstraction based on our proposed techniques. Tables 6.3 and 6.4 show the overall results with respect to percentage of completeness (POC), number of hidden connections, number of semantic clusters created in the graph and the number of semantic pairs compared in order to find hidden connections. More specifically, Table 6.3 highlights the hidden connections with respect to semantic clusters, whereas Tables 6.4 shows the nodes and edges reductions.



(a)



(b)

Figure 6.14: Number of Hidden Connections to Threshold factor based on different Percentage of Completeness (P.o.C.).

Graph	Threshold	nodes	P.o.C	hidden_-connec-tions	semantic_-clusters	semantic_-pairs
graph1	0.6	7001	0.01	86511	46517	7778216
graph2	0.6	7001	0.005	6777	10523	377602
graph3	0.6	7001	0.0025	519	2440	19457
graph4	0.6	10001	0.01	307843	112006	36733762
graph5	0.6	10001	0.005	22387	25496	1653174
graph6	0.6	10001	0.0025	1744	5668	80067
graph7	0.6	15001	0.01	1294838	305219	215450169
graph8	0.6	15001	0.005	82930	66125	8356360
graph9	0.6	15001	0.0025	6834	15192	415393

Table 6.3: Results (hidden information) over 9 graphs for a specific threshold.

Graph	Threshold	nodes_-before	nodes_-after	nodes_-perc	edges_-before	edges_-after	edges_-perc
graph1	0.6	7001	4080	41.72261	243710	237917	2.3770056
graph2	0.6	7001	3433	50.964146	122179	118457	3.04635
graph3	0.6	7001	2642	62.262535	61162	57712	5.6407576
graph4	0.6	10001	5975	40.255974	497388	486192	2.250959
graph5	0.6	10001	5111	48.89511	249363	243274	2.4418218
graph6	0.6	10001	4138	58.624138	124814	119943	3.902607
graph7	0.6	15001	9393	37.384174	1119276	1098947	1.8162634
graph8	0.6	15001	8212	45.256985	561007	550880	1.8051468
graph9	0.6	15001	6700	55.336308	280869	273451	2.641089

Table 6.4: Results (nodes-edges reduction) over 9 graphs for a specific threshold.

Discussion: In this chapter, we present and address various challenges toward to direction of graph analysis with respect to graph visualization. We introduced the notion of Context-Aware Graph as a semantically enhanced representation of the original graph. To this respect, we described the technique of context-aware summarization, which uses the context of the graph. Additionally, we efficiently used the context of the graph to reveal hidden information and discover communities and patterns underneath the original (graph) network, on top of PBF. The experimental results, show that our methods can achieve a node reduction of 48%. Although the results show that this technique offers an interesting approach toward the direction of node reduction an open problem is to investigate what is the trade-off in terms of losing information. Moreover, our approach can be used as a pre-processing step, prior to running any visualization or rendering algorithm, since it exploits the coordinates of the graph to reduce the visual complexity. On the other hand, as expected a reduction in the visual complexity of the graph, implies that information is lost from the original graph. For example, if a super node is created, there is no direct way to distinguish the original source node of an (incoming or outgoing) edge, that belongs to this super node, without expanding it.

Chapter 7

Conclusion

7.1 Synopsis of Contributions

In this thesis, we focused on proposing advanced techniques and algorithms for covering the needs of visualizing and analyzing directed graphs. In particular, the objective was to design, apply and evaluate a suite of new layout techniques over DAGs using efficient algorithms in order to reduce the visual complexity of the graph drawings. Moreover we retained the mental map of the user and improved the clarity of the produced drawings. The above include the following milestones:

We proposed new techniques for finding a minimum Feedback Arc Set (FAS). More specifically, we presented a new heuristic algorithm for computing a minimum FAS in directed graphs. The new technique produces solutions that are better than the ones produced by the best previously known heuristics, reducing the FAS size by more than 50% in some cases. It is based on computing the PageRank score of the nodes of the directed line graph of the input directed graph. Although the time required by our heuristic is heavily influenced by the size of the produced line graph, our experimental results show that it runs very fast even for very large graphs used in graph drawing.

For visualizing such graphs, we presented a detailed general-purpose hierarchical graph drawing framework that is based on the *Path Based Framework (PBF)*. Extensive edge bundling is applied to draw all edges of the graph and the height of the

drawing is minimized using compaction. The drawings produced by this framework are compared to drawings produced by the well known Sugiyama framework in terms of area, number of bends, number of crossings, and execution time. The new algorithm runs very fast and produces drawings that are readable and efficient. Since there are advantages (and disadvantages) to both frameworks, we performed a user study and the results show that the drawings produced by the new framework are well received in terms of clarity, readability, and usability. Hence, the new technique offers an interesting alternative to drawing hierarchical graphs, and is especially useful in applications where user defined paths are important and need to be highlighted.

To further extend the previous model, we presented a set of variant algorithms (suite of algorithms) over PBF. This approach, attempts to draw DAGs hierarchically with few bends and crossings, and by abstracting edges in order to improve the clarity of the drawings while reducing substantially the amount of "ink" required. Additionally, our algorithms have direct applications to the important problem of showing and storing transitivity information of very large graphs and databases. Only a subset of the edges is drawn, thus reducing the visual complexity of the resulting drawing, and the memory requirements for storing the transitivity information. Our algorithms require almost linear time, $O(kn + m)$, where k is the number of paths/channels, n and m is the number of vertices and edges, respectively. They produce progressively more abstract drawings of the input graph.

Concerning the problem of answering reachability queries in directed acyclic graphs, we presented efficient algorithms to construct and search a space efficient data structure in the k-dimensional space that is based on Graph Dominance Drawing. Our algorithms construct this data structure in $O(km)$ time, while it can be stored in $O(kn)$ space. Any reachability query is answered in constant time, since no "falsely implied paths (fips)" are introduced. We also presented experimental results that show that the number of dimensions, k, in the solutions produced by our techniques is low. Additionally, we presented a new model of random DAGs with a prespecified

number of paths and density. The analysis of our experimental results revealed an interesting interplay between density and structure.

Finally, we introduced sophisticated methods for grouping nodes with similar features that naturally decompose the graph into summaries and clusters. This technique is based on the information derived by the edges of the graph and can be used as a visualization aid. We applied this technique on top of the Path Based Framework and efficiently used the context of the graph to reveal hidden information and discover communities and hidden patterns underneath the original graph (network).

7.2 Directions for Future Work and Research

There are several aspects that are worth further work and research. In abstract, these are the following:

Computing a Feedback Arc Set Using PageRank

Since it is NP-hard to compute the minimum FAS the optimum solution for the webgraphs used in our experiments is unknown. Hence, we do not know how close our solutions are to the optimum. It would be interesting to investigate techniques to speedup PageRankFAS in order to make it more applicable to larger webgraphs.

Adventures in Abstraction: Reachability in Hierarchical Drawings

Our study assumes that the path decomposition is given as part of the input, or a minimum size decomposition is computed by one of the known algorithms. However, it is interesting to study the problem of computing a path decomposition and placement of the paths of G which implies the minimum number of cross edges in our drawings. The use of such a decomposition and placement would considerably reduce the number of edges drawn, bends, and crossings in our drawings. Similarly, exploring the possibility to compute a path decomposition and placement of the paths of G which implies the minimum number of jumping cross edges in our drawings seems

interesting. Finally, with respect to the user evaluation performed, it would be interesting to conduct an in-lab user study with both *experts* and *users* in a more controlled manner.

Reachability queries in directed acyclic graphs (DAGs)

An interesting open problem is to elaborate on approaches that further reduce the number of required dimensions. Since these techniques have significant applications in very large graphs, like in large databases, it is important to investigate this problem when the graph changes dynamically, i.e, insertions of new edges/nodes, deletions, etc. New techniques need to be developed specifically designed for dynamic graphs since the cost of recomputing the data structures every time after an update will be prohibitive. For example, it seems that the insertion of a single edge does not alter the structure of the graph significantly, unless of course it causes a cycle. With respect to our proposed graph model for DAGs, it would be interesting to further analyse its behaviour in terms of density and structure and how this can be applied to specific use cases in order to validate its performance and provide insights about its usefulness in real world scenarios.

Graph abstraction techniques for visualizing DAGs based on the Context-Aware Graph

The goal of this work, was not to present a complete formalization of graph analysis techniques but how this can combined with visualization. To this respect, we introduce a framework and highlight the key questions that need to be answered in order to to investigate how graph abstraction can be used as a visualization aid. As expected, edge or node abstraction has a significant reduction in the visual complexity of the graph, since we group nodes and we reduce the edges. However, by doing this, a lot of information is lost from the original graph. The root cause for this ambiguity is that we get a simplified graph, which is a dimensionally reduced view of the original graph. Therefore, it may be important to define a level of abstraction

which can be user defined e.g., a percentage of abstraction, or application specific. Moreover, as further future work it would be interesting to elaborate and evaluate on real world use cases and scenarios toward to the direction of pattern recognition and hidden information. Based on the analysis of our techniques we believe that the open questions that arise, imply lots of interesting research problems that will have significant practical relevance.

Bibliography

[1] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999, ISBN: 0-13-301615-3.

[2] M. Kaufmann and D. Wagner, "Drawing graphs: Methods and models," *LNCS vol. 2025*, 2001.

[3] N. S. Nikolov and P. Healy, *Hierarchical Drawing Algorithms, in Handbook of Graph Drawing and Visualization, ed. Roberto Tamassia*. CRC Press, 2014, ch. 13, pp. 409-453.

[4] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph drawing*. Prentice Hall, Upper Saddle River, NJ, 1999, vol. 357.

[5] G. Ortali and I. G. Tollis, "A new framework for hierarchical drawings," *Journal of Graph Algorithms and Applications*, vol. 23, no. 3, pp. 553–578, 2019. DOI: 10.7155/jgaa.00502.

[6] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[7] G. Di Battista, E Pietrosanti, R. Tamassia, and I. G. Tollis, "Automatic layout of pert diagrams with x-pert," in *[Proceedings] 1989 IEEE Workshop on Visual Languages*, IEEE, 1989, pp. 171–176.

[8] D. L. Fisher and W. M. Goldstein, "Stochastic pert networks as models of cognition: Derivation of the mean, variance, and distribution of reaction time using order-of-processing (op) diagrams," *Journal of Mathematical Psychology*, vol. 27, no. 2, pp. 121–151, 1983.

[9] G. Ortali and I. G. Tollis, "Algorithms and bounds for drawing directed graphs," in *International Symposium on Graph Drawing and Network Visualization*, Springer, 2018, pp. 579–592.

[10] E. R. Tufte, *The visual display of quantitative information*. Graphics press Cheshire, CT, 2001, vol. 2.

[11] V. Geladaris, P. Lionakis, and I. G. Tollis, *Computing a feedback arc set using pagerank*, 2022. DOI: 10.48550/ARXIV.2208.09234. [Online]. Available: https://arxiv.org/abs/2208.09234.

[12] P. Lionakis, G. Kritikakis, and I. G. Tollis, "Algorithms and experiments comparing two hierarchical drawing frameworks," *CoRR*, vol. abs/2011.12155, 2020. arXiv: 2011.12155. [Online]. Available: https://arxiv.org/abs/2011.12155.

[13] P. Lionakis, G. Ortali, and I. Tollis, "Adventures in abstraction: Reachability in hierarchical drawings," in *Graph Drawing and Network Visualization: 27th International Symposium, GD 2019, Prague, Czech Republic, September 17–20, 2019, Proceedings*, 2019, pp. 593–595.

[14] P. Lionakis, G. Ortali, and I. G. Tollis, "Constant-time reachability in dags using multidimensional dominance drawings," *SN Computer Science*, vol. 2, no. 4, pp. 1–14, 2021.

[15] C. Budak, D. Agrawal, and A. El Abbadi, "Limiting the spread of misinformation in social networks," in *Proceedings of the 20th international conference on World wide web*, 2011, pp. 665–674. DOI: 10.1145/1963405.1963499.

[16] X. He, G. Song, W. Chen, and Q. Jiang, "Influence blocking maximization in social networks under the competitive linear threshold model," in *Proceedings of the 2012 siam international conference on data mining*, SIAM, 2012, pp. 463–474.

[17] M. Simpson, V. Srinivasan, and A. Thomo, "Clearing contamination in large networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1435–1448, 2016. DOI: 10.1109/TKDE.2016.2525993.

[18] D. S. Johnson, "The np-completeness column: An ongoing gulde," *Journal of Algorithms*, vol. 3, no. 4, pp. 381–395, 1982.

[19] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[20] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg, "An exact method for the minimum feedback arc set problem," *ACM J. Exp. Algorithmics*, vol. 26, 2021, ISSN: 1084-6654. DOI: 10.1145/3446429. [Online]. Available: https://doi.org/10.1145/3446429.

[21] P. Eades, X. Lin, and W. F. Smyth, "A fast and effective heuristic for the feedback arc set problem," *Information Processing Letters*, vol. 47, no. 6, pp. 319–323, 1993.

[22] F. J. Brandenburg and K. Hanauer, "Sorting heuristics for the feedback arc set problem," in *Technical Report MIP-1104*, University of Passau Germany, 2011.

[23] M. Simpson, V. Srinivasan, and A. Thomo, "Efficient computation of feedback arc set at web-scale," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 133–144, 2016. DOI: 10.14778/3021924.3021930.

[24] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: Ranking and clustering," *Journal of the ACM (JACM)*, vol. 55, no. 5, pp. 1–27, 2008. DOI: 10.1145/1411509.1411513.

[25] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Comput. Networks*, vol. 30, no. 1-7, pp. 107–117, 1998.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Stanford InfoLab, Tech. Rep., 1999.

[27] S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica ®*. Cambridge University Press, 1990.

[28] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[29] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th international conference on World Wide Web*, S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, Eds., ACM Press, 2011, pp. 587–596.

[30] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, Manhattan, USA: ACM Press, 2004, pp. 595–601.

[31] G. Di Battista *et al.*, "Drawing directed acyclic graphs: An experimental study," in *Graph Drawing, Symposium on Graph Drawing, GD '96, Berkeley, California, USA, September 18-20, Proceedings*, S. C. North, Ed., ser. Lecture Notes in Computer Science, vol. 1190, Springer, 1996, pp. 76–91.

[32] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981. DOI: 10.1109/TSMC.1981.4308636. [Online]. Available: https://doi.org/10.1109/TSMC.1981.4308636.

[33] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput.*, vol. 2, no. 4, pp. 225–231, 1973. DOI: 10.1137/0202019. [Online]. Available: https://doi.org/10.1137/0202019.

[34] A. Kuosmanen, T. Paavilainen, T. Gagie, R. Chikhi, A. I. Tomescu, and V. Mäkinen, "Using minimum path cover to boost dynamic programming on dags: Co-linear chaining extended," in *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, April 21-24, 2018, Proceedings*, 2018, pp. 105–121.

[35] J. B. Orlin, "Max flows in O(nm) time, or better," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 765–774. DOI: 10.1145/2488608.2488705. [Online]. Available: http://doi.acm.org/10.1145/2488608.2488705.

[36] C. Schnorr, "An algorithm for transitive closure with linear expected time," *SIAM J. Comput.*, vol. 7, no. 2, pp. 127–133, 1978. DOI: 10.1137/0207011. [Online]. Available: https://doi.org/10.1137/0207011.

[37] *Tom Sawyer Software*. [Online]. Available: www.tomsawyer.com.

[38] *yWorks*. [Online]. Available: www.yworks.com.

[39] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel, "The open graph drawing framework (OGDF)," in *Handbook on Graph Drawing and Visualization*. 2013, pp. 543–569.

[40] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo, "A technique for drawing directed graphs," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 214–230, 1993.

[41] G. Sander, "Layout of compound directed graphs," Universität des Saarlandes, Tech. Rep., 1996.

[42] M. T. Goodrich and R. Tamassia, *Algorithm Design and Applications*, 1st. Wiley Publishing, 2014, ISBN: 1118335910.

[43] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale, "Empirical studies in information visualization: Seven scenarios," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 9, pp. 1520–1536, 2011.

[44] J. Heer, F. v. Ham, S. Carpendale, C. Weaver, and P. Isenberg, "Creation and collaboration: Engaging new audiences for information visualization," in *Information visualization*, Springer, 2008, pp. 92–133.

[45] C. Plaisant, "The challenge of information visualization evaluation," in *Proceedings of the working conference on Advanced visual interfaces*, 2004, pp. 109–116.

[46] E. Wall *et al.*, "A heuristic approach to value-driven evaluation of visualizations," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 491–500, 2018.

[47] R. Amar, J. Eagan, and J. Stasko, "Low-level components of analytic activity in information visualization," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, IEEE, 2005, pp. 111–117.

[48] E. Di Giacomo, W. Didimo, G. Liotta, F. Montecchiani, and I. G. Tollis, "Exploring complex drawings via edge stratification," in *International Symposium on Graph Drawing*, Springer, 2013, pp. 304–315.

[49] M. Ghoniem, J.-D. Fekete, and P. Castagliola, "A comparison of the readability of graphs using node-link and matrix-based representations," in *IEEE symposium on information visualization*, Ieee, 2004, pp. 17–24.

[50] H. C. Purchase, J. Hamer, M. Nöllenburg, and S. G. Kobourov, "On the usability of lombardi graph drawings," in *International symposium on graph drawing*, Springer, 2012, pp. 451–462.

[51] W. Didimo, E. M. Kornaropoulos, F. Montecchiani, and I. G. Tollis, "A visualization framework and user studies for overloaded orthogonal drawings," *Comput. Graph. Forum*, vol. 37, no. 1, pp. 288–300, 2018. DOI: 10.1111/cgf.13266. [Online]. Available: https://doi.org/10.1111/cgf.13266.

[52] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.

[53] H. V. Jagadish, "A compression technique to materialize transitive closure," *ACM Trans. Database Syst.*, vol. 15, no. 4, pp. 558–598, Dec. 1990, ISSN: 0362-5915. DOI: 10.1145/99935.99944. [Online]. Available: http://doi.acm.org/10.1145/99935.99944.

[54] R. Jin, N. Ruan, S. Dey, and J. X. Yu, "SCARAB: scaling reachability computation on large graphs," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, 2012, pp. 169–180. DOI: 10.1145/2213836.2213856. [Online]. Available: https://doi.org/10.1145/2213836.2213856.

[55] S. J. van Schaik and O. de Moor, "A memory efficient reachability data structure through bit vector compression," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, 2011, pp. 913–924. DOI: 10.1145/1989323.1989419. [Online]. Available: https://doi.org/10.1145/1989323.1989419.

[56] R. R. Veloso, L. Cerf, W. M. Jr., and M. J. Zaki, "Reachability queries in very large graphs: A fast refined online search approach," in *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, 2014, pp. 511–522. DOI: 10.5441/002/edbt.2014.46. [Online]. Available: https://doi.org/10.5441/002/edbt.2014.46.

[57] H. Yildirim, V. Chaoji, and M. J. Zaki, "GRAIL: a scalable index for reachability queries in very large graphs," *VLDB J.*, vol. 21, no. 4, pp. 509–534, 2012. DOI: 10.1007/s00778-011-0256-4. [Online]. Available: https://doi.org/10.1007/s00778-011-0256-4.

[58] E. R. Gansner, E. Koutsofios, S. C. North, and K. Vo, "A technique for drawing directed graphs," *IEEE Trans. Software Eng.*, vol. 19, no. 3, pp. 214–230, 1993. DOI: 10.1109/32.221135. [Online]. Available: https://doi.org/10.1109/32.221135.

[59] L. Li, W. Hua, and X. Zhou, "HD-GDD: high dimensional graph dominance drawing approach for reachability query," *World Wide Web*, vol. 20, no. 4, pp. 677–696, 2017. DOI: 10.1007/s11280-016-0407-z. [Online]. Available: https://doi.org/10.1007/s11280-016-0407-z.

[60] J. Su, Q. Zhu, H. Wei, and J. X. Yu, "Reachability querying: Can it be even faster?" *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 683–697, 2017.

[61] H. A. ElGindy, M. E. Houle, W. Lenhart, M. Miller, D. Rappaport, and S. Whitesides, "Dominance drawings of bipartite graphs," in *Proceedings of the 5th Canadian Conference on Computational Geometry, Waterloo, Ontario, Canada, August 1993*, 1993, pp. 187–191.

[62] G. Di Battista, R. Tamassia, and I. G. Tollis, "Area requirement and symmetry display of planar upward drawings," *Discrete & Computational Geometry*, vol. 7, pp. 381–401, 1992. DOI: 10.1007/BF02187850. [Online]. Available: https://doi.org/10.1007/BF02187850.

[63] J. M. Six and I. G. Tollis, "Automated visualization of process diagrams," in *Graph Drawing*, P. Mutzel, M. Jünger, and S. Leipert, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 45–59, ISBN: 978-3-540-45848-7.

[64] J. Zhou, S. Zhou, J. X. Yu, H. Wei, Z. Chen, and X. Tang, "Dag reduction: Fast answering reachability queries," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, Chicago, Illinois, USA: ACM, 2017, pp. 375–390, ISBN: 978-1-4503-4197-4. DOI: 10.1145/3035918.3035927. [Online]. Available: http://doi.acm.org/10.1145/3035918.3035927.

[65] R. M. McConnell and J. Spinrad, "Linear-time transitive orientation.," *SODA*, pp. 19–25, 1997.

[66] M. Yannakakis, "The complexity of the partial order dimension problem.," *SIAM J. Algebraic and Discrete Methods*, vol. 3, pp. 303–322, 1982.

[67] Y. Chen and Y. Chen, "On the dag decomposition," *British Journal of Mathematics and Computer Science*, 2014, 10(6): 1-27, 2015, Article no.BJMCS.19380, ISSN: 2231-0851. [Online]. Available: https://www.researchgate.net/publication/285591312_Pre-Publication_Draft_2015_BJMCS_19380.

[68] Y. Peng, Y. Zhang, X. Lin, L. Qin, and W. Zhang, "Answering billion-scale label-constrained reachability queries within microsecond," *Proceedings of the VLDB Endowment*, vol. 13, pp. 812–825, Feb. 2020. DOI: 10.14778/3380750.3380753.

[69] H. V. Jagadish, "A compression technique to materialize transitive closure," *ACM Trans. Database Syst.*, vol. 15, no. 4, pp. 558–598, 1990. DOI: 10.1145/99935.99944. [Online]. Available: http://doi.acm.org/10.1145/99935.99944.

[70] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998, pp. 112–127, ISBN: 978-0-13-301615-4.

[71] E. M. Kornaropoulos and I. G. Tollis, "Weak dominance drawings and linear extension diameter," *CoRR*, vol. abs/1108.1439, 2011. arXiv: 1108.1439. [Online]. Available: http://arxiv.org/abs/1108.1439.

[72] E. M. Kornaropoulos and I. G. Tollis, "Weak dominance drawings for directed acyclic graphs," in *Graph Drawing - 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, 2012, pp. 559–560. DOI: 10.1007/978-3-642-36763-2_52. [Online]. Available: https://doi.org/10.1007/978-3-642-36763-2_52.

[73] E. M. Kornaropoulos and I. G. Tollis, "Algorithms and bounds for overloaded orthogonal drawings," *Journal of Graph Algorithms and Applications*, vol. 20, no. 2, pp. 217–246, 2016. DOI: 10.7155/jgaa.00391.

[74] R. P. Dilworth, "A decomposition theorem for partially ordered sets," *Ann. Math. Ser.*, vol. 251, pp. 161 –166, Jan. 1950. DOI: 10.1007/978-0-8176-4842-8_10.

[75] G. Ortali and I. G. Tollis, "Multidimensional Dominance Drawings," *arXiv e-prints*, arXiv:1906.09224, arXiv:1906.09224, 2019. arXiv: 1906.09224 [cs.DS].

[76] A. Hagberg, P. Swart, and D. Chult, "Exploring network structure, dynamics, and function using networkx," Jan. 2008.

[77] P Erdős and A Rényi, "On random graphs i," *Publ. Math. Debrecen*, vol. 6, no. 290-297, p. 18, 1959.

[78] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[79] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *Nature*, vol. 393, no. 6684, p. 440, 1998.

[80] G. Di Battista *et al.*, "Drawing directed acyclic graphs: An experimental study," *International Journal of Computational Geometry & Applications*, vol. 10, no. 06, pp. 623–648, 2000.

[81] J. Cruz, C. Bothorel, and F. Poulet, "Layout algorithm for clustered graphs to analyze community interactions in social networks," Aug. 2012. DOI: 10.1109/ASONAM.2012.120.

[82] V. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics Theory and Experiment*, vol. 2008, Apr. 2008. DOI: 10.1088/1742-5468/2008/10/P10008.

[83] E. Ntoutsi, K. Stefanidis, K. Norvag, and H.-P. Kriegel, "Fast group recommendations by applying user clustering," Oct. 2012, ISBN: 978-3-642-34001-7. DOI: 10.1007/978-3-642-34002-4_10.

[84] M. Newman, "Detecting community structure in networks," *Eur Phys J*, vol. 38, Mar. 2004.

[85] J. Cruz, C. Bothorel, and F. Poulet, "Semantic clustering of social networks using points of view.," *CORIA 2011: COnference en Recherche d'Information et Applications - Conference on Information Retrieval and Applications*, pp. 175–182, Mar. 2011.

[86] O. Lassila *et al.*, "Graph? yes! which one? help!" *CoRR*, vol. abs/2110.13348, 2021. arXiv: 2110.13348. [Online]. Available: https://arxiv.org/abs/2110.13348.

[87] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu, "A survey of heterogeneous information network analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 17–37, 2017. DOI: 10.1109/TKDE.2016.2598561.

[88] Y. Sun and J. Han, "Mining heterogeneous information networks: A structural analysis approach," *Acm Sigkdd Explorations Newsletter*, vol. 14, no. 2, pp. 20–28, 2013.

[89] M. van Leeuwen, J. Vreeken, and A. Siebes, "Compression picks item sets that matter," in *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2006, pp. 585–592.

[90] R. Cilibrasi and P. M. Vitányi, "Clustering by compression," *IEEE Transactions on Information theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[91] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos, "Fully automatic cross-associations," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 79–88.

[92] K. Smets and J. Vreeken, "The odd one out: Identifying and characterising anomalies," in *Proceedings of the 2011 SIAM international conference on data mining*, SIAM, 2011, pp. 804–815.

[93] L. Akoglu, H. Tong, J. Vreeken, and C. Faloutsos, "Fast and reliable anomaly detection in categorical data," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 415–424.

[94] J. Vreeken, M. Van Leeuwen, and A. Siebes, "Krimp: Mining itemsets that compress," *Data Mining and Knowledge Discovery*, vol. 23, no. 1, pp. 169–214, 2011.

[95] B. A. Prakash, J. Vreeken, and C. Faloutsos, "Spotting culprits in epidemics: How many and which ones?" In *2012 IEEE 12th International Conference on Data Mining*, IEEE, 2012, pp. 11–20.

[96] C. Dunne and B. Shneiderman, "Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 3247–3256.

[97] L. Jin and D. Koutra, "Eco: Comparative visualization of time-evolving network summaries," Visualization, 2017.

[98] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–34, 2018.

[99] J. D. U. Anand Rajaraman Jure Leskovec, *Mining of Massive Datasets*. 2010, http://infolab.stanford.edu/~ullman/mmds.html.

[100] S. Wasserman, K. Faust, *et al.*, "Social network analysis: Methods and applications," 1994.

[101] P. Arabie, L. Hubert, and G. De Soete, *Clustering and classification*. World Scientific, 1996.

Appendix Acronyms

DAG Directed Acyclic Graph

FAS Feedback Arc Set

PBF Path Based Hierarchical Drawing Framework

PB-Draw Path Based - Draw

OGDF Open Graph Drawing Framework

PERT Program Evaluation Review Technique

FIP Falsely Implied Paths

ER Erdős-Rényi

WS Watts-Strogatz

BA Barabasi-Albert

HINs Heterogeneous Information Networks

POC Percentage of Completeness

POGI Point Of Graph Interest