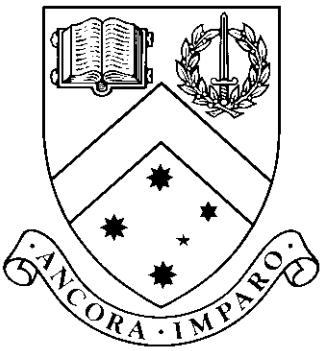
A Human-Centred Approach to Network Layout Algorithm Design

by

Steve Kieffer, BSc, MSc, MSc



Thesis

Submitted by Steve Kieffer

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Dr. Kim Marriott

Associate Supervisor: Dr. Michael Wybrow

Associate Supervisor: Dr. Tim Dwyer

Caulfield School of Information Technology
Monash University

May, 2017

Contents

List of Tables

List of Figures

A Human-Centred Approach to Network Layout Algorithm Design

Steve Kieffer, BSc, MSc, MSc
[redacted]
Monash University, 2017

Supervisor: Dr. Kim Marriott
[redacted]
Associate Supervisor: Dr. Michael Wybrow
[redacted]
Associate Supervisor: Dr. Tim Dwyer
[redacted]

Abstract

Scientists and engineers often use a type of information graphic called a node-link diagram to represent the complex systems they study. Here, labelled boxes connected by lines show how elements of the system relate to one another. It is challenging to create a good layout for such a diagram, positioning the boxes and lines for maximum clarity. Existing software can help with the job, but the results often look artificial, or fail to capture a lab-specific style.

Insofar as node-link diagrams communicate information, like a language does, it is suitable to refer to the special layout styles employed by various labs as "dialects". This thesis develops a human-centred methodology for teaching computers to "speak" such dialects, i.e. to lay out node-link diagrams in a manner appropriate to one specialised style or another.

The methodology has three steps: (I) a formative user study, in which we examine both the process and the product of hand-made layout in a given dialect; (II) the translation of the formative study's findings into a new layout algorithm that mimics human design behaviour, at least in product if not necessarily in process; (III) a normative user study, in which we examine user preference and performance (on standard node-link diagram tasks like path following) on layouts generated by: (a) the new algorithm, (b) human beings, and (c) an existing state-of-the-art algorithm.

The three-part, human-centred methodology is one of the main contributions of this thesis. Another contribution is the "DiAlEcT Layout Framework," which is a set of instructions and guidelines for carrying out Step (II) of the human-centred methodology.

While the methodology and framework may be thought of as tools intended for use by computer scientists and developers, other contributions of this thesis include tools intended for end users, developed using the DiAlEcT framework. (1) a new layout algorithm called "HOLA" for orthogonal diagrams, which performed comparably to human layout and outperformed the state-of-the-art algorithm (Topology-Shape-Metrics) in a normative study of the kind described above; (2) a new layout algorithm called "MCGL" for SBGN (Systems Biology Graphical Notation) diagrams, also developed using the DiAlEcT framework; (3) interactive tools with which the user may perform both high-level and low-level editing of the diagrams produced by these algorithms.

The reason for developing interactive tools is that while the new layout algorithms succeed in making the same sorts of design decisions that human designers would make within a given layout dialect, these algorithms are nevertheless bound to eventually make a choice that a particular user would not have made. The interactive tools provide easy ways for the user to alter such design choices after the fact, and working in high-level terms: simple interaction gestures with the mouse allow the user to make requests like "switch these two paths," or "rotate this subgraph". Together the automatic and interactive techniques make a pair of complementary layout tools helping to make the computer into an ideal layout assistant.

A Human-Centred Approach to Network Layout Algorithm Design

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Steve Kieffer
May 20, 2017

Acknowledgments

This thesis has been improved by a great deal of insightful feedback from my supervisors Kim Marriott, Michael Wybrow, and Tim Dwyer, and enriched by content from our published papers. The ideas represented here are the product of collaborative efforts among us.

Advice on SBGN and MetaCrop layout from Tobias Czauderna, Falk Schreiber, and Matthias Klapperstück has also been of great help. The techniques for handling layout with ports were developed in productive and enlightening collaboration with Ulf Rüegg.

This work would not have been possible without the generous scholarships granted by Monash University and by National Information and Communications Technology Australia (NICTA, now Data61).

I am grateful both to my supervisors and other collaborators, from whom I learned something every time we met to puzzle over the next algorithm, and to my family and friends, who helped to keep life fun amid all the hard work.

Steve Kieffer

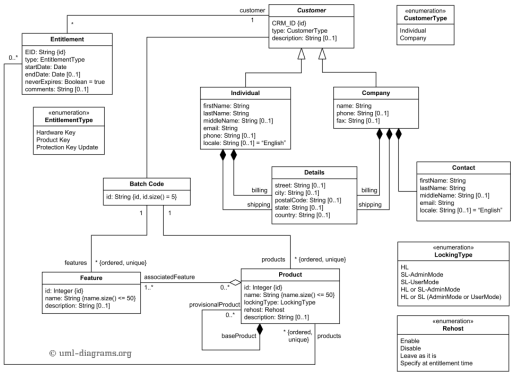
Monash University
May 2017

Chapter 1

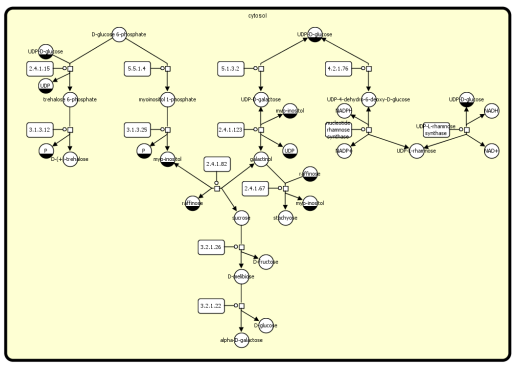
Introduction

When we study the world in which we live we are often confronted by complex systems consisting of many interconnected components. These can arise in nature or be human-made. An ecologist might wish to understand the relationships between the plant and animal species living in a certain biome. A neuroscientist might wish to understand the connections between various regions in the human brain. Industrial engineers need to understand the flow of materials between processes on a manufacturing floor, and software engineers the connections between software components. In hopes of halting the onset of diseases like Parkinson's and Alzheimer's, biologists seek to understand the flow of biochemicals in metabolic pathways and gene-regulatory networks within the cells of the human body.

Node-link diagrams help us to understand such systems by depicting the entities in the system as *nodes*, and the relationships between the entities as *links* between those nodes (Figure 1.1). Such diagrams can help us to understand "the big picture," to comprehend how all the parts of the system fit together, to better recall the various parts by mentally associating them with various locations in a static image, and to quickly switch our attention back and forth between the big picture and the details of one or another part of the system by zooming in and out. (See for example Munzner [Mun14], Chapter 9.) Whether it is a biologist seeking to block a disease pathway with a new medicine, an engineer wishing to improve an industrial process or a software design, or an ecologist evaluating the sustainability of fishing or farming practices, all need to first understand the systems they are studying, and then convey the same understanding to others. For such purposes a picture is worth a thousand words.



(a) Node-link diagram using the UML language (image credit: `http://uml-diagrams.org`)



(b) Diagram using SBGN notation to show biochemical reactions involved in sugar metabolism (image credit: IPK Gatersleben)

Figure 1.1: Node-link diagrams

With the raw data that describe such a complex system in hand, it is natural to ask whether a computer can be made to automatically, or semi-automatically (i.e. with some user interaction), create a clear and understandable node-link diagram from the data, and this has been the subject of *network layout research* for approximately the last fifty years.

1.1 A Human-Centred Approach to Network Layout Algorithm Design

Let us consider the trade-offs between getting a computer to create a node-link diagram automatically, and having a human being create the diagram by hand. Two important factors are the sheer volume of data, and the desire to have creative control. If a network consists of just ten nodes then a nice drawing might quite easily be made by hand in a vector graphics editor like Inkscape or PowerPoint, and users are then able to make all the design decisions themselves. If the number of nodes is closer to a hundred then it is easy to imagine most busy professionals eagerly welcoming special software that would help them with the job, despite its tendency to take away some creative control. Once available, such software might even be preferred for the ten-node case, at least as a way to create an initial layout, which could then be altered manually if desired. Ideally, the computer should help users to get through large jobs, but without forcing them to accept layouts they would not have created themselves, had they possessed the time and patience to do so unassisted.

This suggests that the computer should play the role of *assistant* in the production of node-link diagrams. If we imagine a workshop where master and assistant collaboratively produce an artefact (of any kind), the process might go something like this: (1) Master says to assistant: "You know essentially what I would do. Go off and get the project started, then show me what you have." (2) The assistant shows the master the initial work, and the master says, "This is mostly good, but I would make the following changes...." (3) The assistant is able to make the requested changes working alone, or perhaps requiring some small amount of guidance. This process may repeat through several iterations, at the end of which the master approves of the final product, but was spared much of the labour involved in producing it.

The reason for considering the master-assistant metaphor is twofold, in that it motivates both our goals and our methodology in this thesis. In terms of goals, the metaphor suggests that the computerised assistant be able to do two things: (A) create a reasonable first attempt at a layout, and (B) understand a high-level description of desired changes, in order to improve the initial attempt. This thesis is concerned primarily with ability (A), but in the next-to-last chapter some preliminary work on ability (B) is also presented.

Secondly, since each new assistant must study and learn from the skills of the existing masters of the craft, the metaphor also suggests the methodology that has been taken in this thesis, namely, a human-centred approach. Here the techniques of living, human "masters" of network layout are studied and translated into algorithms, and then these living experts are asked to evaluate the results.

To be precise, the human-centred methodology has three steps: (I) a "formative" user study, in which we examine both the process and the product of hand-made layout in a given style; (II) the translation of the formative study's findings into a new layout algorithm that mimics human design behaviour, at least in product if not necessarily in process; (III) a "normative" user study, in which we examine user preference and performance (on standard node-link diagram tasks like path following) on layouts generated by: (a) the new algorithm, (b) human beings, and (c) an existing state-of-the-art algorithm.

1.2 Network Layout Languages and Dialects

To continue with the metaphor, different workshops may produce different types of artefacts—for example the blacksmiths produce iron work, and the basket weavers produce baskets—while two workshops of the same kind may produce their artefact in different styles—for example one basket weaver may produce round baskets while another weaver produces square ones.

So too in the many industries that use node-link diagrams, different disciplines may produce different types of diagrams, while two labs belonging to the same discipline may produce their diagrams in different styles. But node-link diagrams are essentially a communicative tool, like a language. This is why special diagramming systems are given names like Unified Modelling *Language* (UML, Figure 1.1 left), and Systems Biology Graphical *Notation* (SBGN, Figure 1.1 right). Therefore in classifying the various types and styles of diagrams in use in the world today, we find it suitable to use terms like *language family*, *language*, and *dialect*.

In these terms, we may say that different types of lab (e.g. software engineering versus biology) use different node-link diagram *languages*, of which the aforementioned UML and SBGN are examples. Different labs of the same kind (e.g. two biology labs) are apt to develop their own "house style" when it comes to layout and design decisions, and we refer to these as different node-link diagram *dialects*. Thus two biology labs may both use SBGN, but each will develop its own dialect. For example one lab may like to lay out the nodes of cyclic metabolic processes on the circumference of a circle, while another lab may prefer to lay them out on the perimeter of a square.

Meanwhile, it often makes sense to group similar node-link diagram languages into *language families*. For example both UML and VLSI circuit notation belong to what in this thesis we dub the *Orthogonal Language Family*, since both employ *orthogonal connectors*.[1] This means that the lines connecting nodes in the diagrams are drawn using alternating horizontal and vertical segments, like in the diagram on the left of Figure 1.1.

In this thesis we take an interest in both the Orthogonal Language Family, and in the SBGN language, contributing a new layout algorithm for the former in Chapter 5, and one for the latter in Chapter 8.

However, in keeping with our overall purpose and human-centred methodology, we cannot *directly* develop a layout algorithm for a whole language, let alone an entire language family, but can only do so *indirectly* by studying a given dialect of the language or family in question. This must be the case if our assistant algorithms are to be informed by observations of existing masters' diagram layouts; at least, this is so if we confine our attention to the diagrams produced by just one lab or workshop, where a single "house style" is in effect. While an ambitious future effort might attempt to combine observations across multiple labs, in this thesis I have stuck to the rule of one lab, one algorithm.

In the case of SBGN layout, the chosen lab was the Leibniz Institute of Plant Genetics and Crop Plant Research in Gatersleben, Germany, whose SBGN diagrams are gathered in a system they call MetaCrop. Accordingly I refer to theirs as the "MetaCrop dialect" of SBGN diagrams.

In the case of orthogonal diagrams, the word "lab" has to be interpreted loosely. I conducted an online experiment (described in Chapter 4), by designing a web site where participants were asked to lay out orthogonal node-link diagrams, and the editing process was recorded. Later their diagrams were ranked by a new set of participants, so that

[1] While not strictly mandated, orthogonal connectors are extremely common in UML.

I could identify the "masters" of this particular craft as those who created the highest-ranked diagrams. Thus in this case the "lab" was a temporary one, with anonymous members. I refer to this as the "Orthowontist dialect" of orthogonal diagrams.[2]

1.3 The DiAlEcT Layout Framework

One of the main contributions of this thesis is a *method* for solving the problem set out in Section 1.1, i.e., to observe an existing layout style, and, based on these observations, create layout tools to aid in the creation of layouts of that style. In terms of our running metaphor, DiAlEcT is a method for observing a master and creating an assistant.

The capital letters D, A, E, T in the name DiAlEcT stand for the four *layout phases* of this method: *Distribute*, *Arrange*, *Emend*, and *Transform*. The first three are automatic and the fourth is interactive. The *Distribute* phase uses *stress-minimisation* (see Sections 1.4 and 2.2) to give the nodes of the network a reasonable distribution in the plane. The *Arrange* phase then applies constraints (again see Section 2.2) to arrange nodes in the sorts of patterns that people create when designing node-link diagrams by hand. The *Emend* phase removes lingering defects. The *Transform* phase allows the user to make broad changes in "high-level" terms, as well as "low-level" changes to the final layout. The system is described in full in Chapter 7.

Perhaps the clearest way to understand the purpose and nature of any tool is to think about who would be likely to use it, and what they would use it for. Here then is a conceivable scenario for the use of DiAlEcT:

> Firm A uses node-link diagrams, which its employees create by hand, and they have a distinct "house style". Firm B offers consulting and software solutions. Firm A hires Firm B to create custom layout tools to help its employees create node-link diagrams faster and more easily.
>
> An engineer from Firm B examines a set of representative examples of node-link diagrams created in Firm A's house style, and maybe even pays a visit to interview the most experienced employees of Firm A and ask them about design decisions they make when creating diagrams.
>
> The engineer then applies the DiAlEcT framework in order to develop two layout tools for Firm A. The first of these is an algorithm that automatically generates node-link diagrams in Firm A's house style. If this algorithm has one major drawback it is that, while it succeeds in making the same *kinds* of design decisions employees of Firm A would make, it is nevertheless bound to eventually make a *particular* choice that a particular employee will disapprove of.
>
> This is why the DiAlEcT framework also helps the engineer of Firm B to generate a second tool, this one an interactive layout editor. Employees of Firm A can use this editor to easily alter the layouts generated by the automatic algorithm in high-level terms. For example, we could imagine a corrective statement such as, "I'm happy that these nodes are arranged in a crescent shape, but I want that crescent to bow downward, not upward." The editor facilitates high-level changes of this kind in just a few mouse clicks, while also allowing arbitrary low-level changes.

[2] This word, "Orthowontist" is a pun, and it was used as the title of the experiment, greeting participants on the front page of the site. It refers to the experiment's goal of determining designers' "wont" or set of habits, in laying out "ortho" diagrams.

It is important to emphasise however that, while in concept the DiAlEcT framework stands complete, only the automatic phases D, A, E have been thoroughly examined in this thesis. Only initial explorations have been done with the interactive phase T (see Chapter 9), and full testing of that part of the process awaits future work.

1.4 Blending Observed Behaviour with the Constrained Layout Tradition

While the human-centred methodology of this thesis may be a novel contribution to the field of node-link diagram layout research, that is no reason why we cannot draw on the existing tools and techniques of the field; on the contrary, we would surely be lost if we were to try to start from scratch. For, after we have observed the design patterns of existing masters of a given layout dialect, we need a way to build an algorithm that can mimic their behaviour. The question then is which of the existing techniques provides the most suitable foundation on which to build, and the answer is that we build on a technique called *Constrained Stress-Minimising Layout* or CSML. If each generation of researchers stands on the shoulders of giants, CSML is our colossus.

Why do we use CSML? Is there some reason why it is a natural choice when attempting to make computers simulate observed human layout behaviour? A few things may be said briefly here in order to justify the choice, but the justification will emerge in full as the thesis progresses. To begin with, a 2008 study by Dwyer et al. [DLF+09] demonstrated, among other things, that the best-liked hand-made layout was the one that came the closest to the computer-made stress-minimal layout. This showed not only that people like low-stress layouts, but that among hand-made layouts low-stress ones are favoured.

Chapter 2, on background, provides further reasons why the use of CSML is motivated. On the one hand (Section 2.2) this includes a demonstration of how the stress function emerged naturally from two separate lines of network layout research: force-directed layout [FCW67, Ead84], and multidimensional scaling [KS80]. On the other hand (Section 2.3) we examine how constrained optimisation in general was identified in the early 90s as the proper arena in which to try to develop layout algorithms that can recreate the results of human graphic designers [Mar91a, KMS94].

1.5 Sparse Networks

There are various ways to measure the *density* of a network, the simplest of which is to divide the number of links by the number of nodes. Networks with high density are called *dense* while those with low density are called *sparse*. For an example of a dense network consider the pages of Wikipedia and their links to one another. For examples of sparse networks consider a family tree, a map of a metropolitan train network, or a typical flow chart.

In this project we are interested exclusively in sparse networks. After all, we can only hope to make computers lay out networks the way people do if people actually *do* make hand-made layouts of the kind in question. When it comes to dense networks people typically do not draw these at all, not because it is impossible but because the job is usually felt to be too complex and difficult. Pictures of dense networks have begun appearing in recent years only because computers have been made to draw them.

On the contrary, the specific domains of interest to us are ones in which sparse networks are typically used, and hand-made layouts are abundant. To begin with, consider orthogonal diagrams. When pioneers Di Battista et al. (see Section 2.4) performed an experimental comparison in 1995 of three orthogonal layout algorithms, they wrote that,

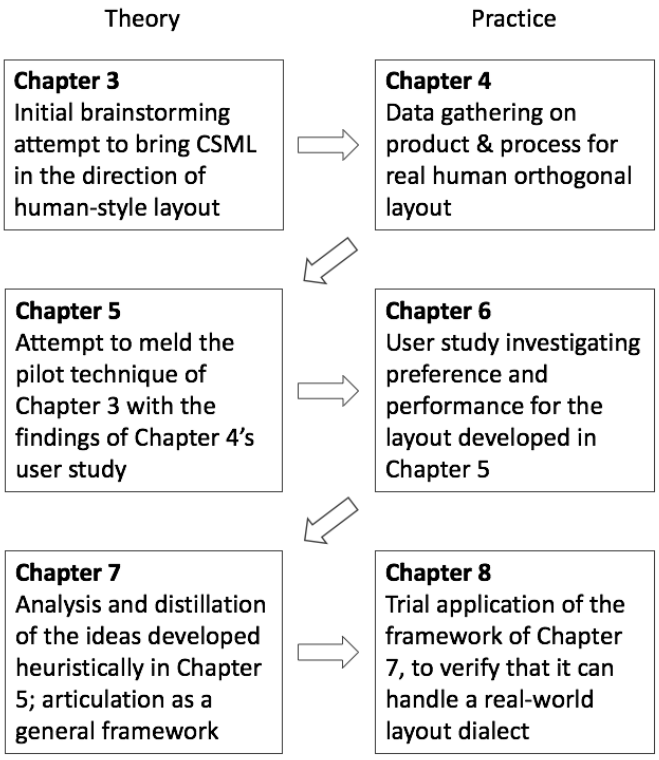


Figure 1.2: Alternation between theory and practice in the core chapters of the thesis

"Sparsity and 'near-planarity' are typical properties of graphs used in software engineering and database applications." [DBGL95]. Consequently the graphs used in their study had densities between 1.2 and 1.3, again using the simple metric of number of links divided by number of nodes.

As for biological network diagrams, in SBGN it is customary to use a technique called *cloning* (see Section 6.4.2) to lower the density of the network to near unity. The practice is so common that the SBGN standard even includes a special way of indicating cloned nodes.

If network layout research can be classified according to the density of the networks considered, this thesis decidedly belongs to the category concerned with sparse networks.

1.6 Outline

Chapters 1, 2, and 10 provide introduction, background, and conclusions, respectively. Chapter 9 covers preliminary work on interactive layout methods. The heart of the work lies in Chapters 3 through 8, wherein we repeatedly bounce back and forth between theory and practice, as illustrated in Figure 1.2. See also Figure 1.3.

Chapter 3 (INVENTION node in Figure 1.3) represents an intuitive first attempt to bring CSML in the direction of orthogonal and SBGN layout. We experiment with various ways of adding constraints to position nodes suitably for these types of diagram. In particular we examine the effectiveness of both goal-function methods (NS = Node-Snap, and GS = Grid-Snap), and a simple greedy strategy (ACA = Adaptive Constrained Alignment) for generating constraints on an otherwise stress-minimised layout. We find that such a technique looks promising for the achievement of layouts conforming to the aesthetic demands of various layout languages or dialects.

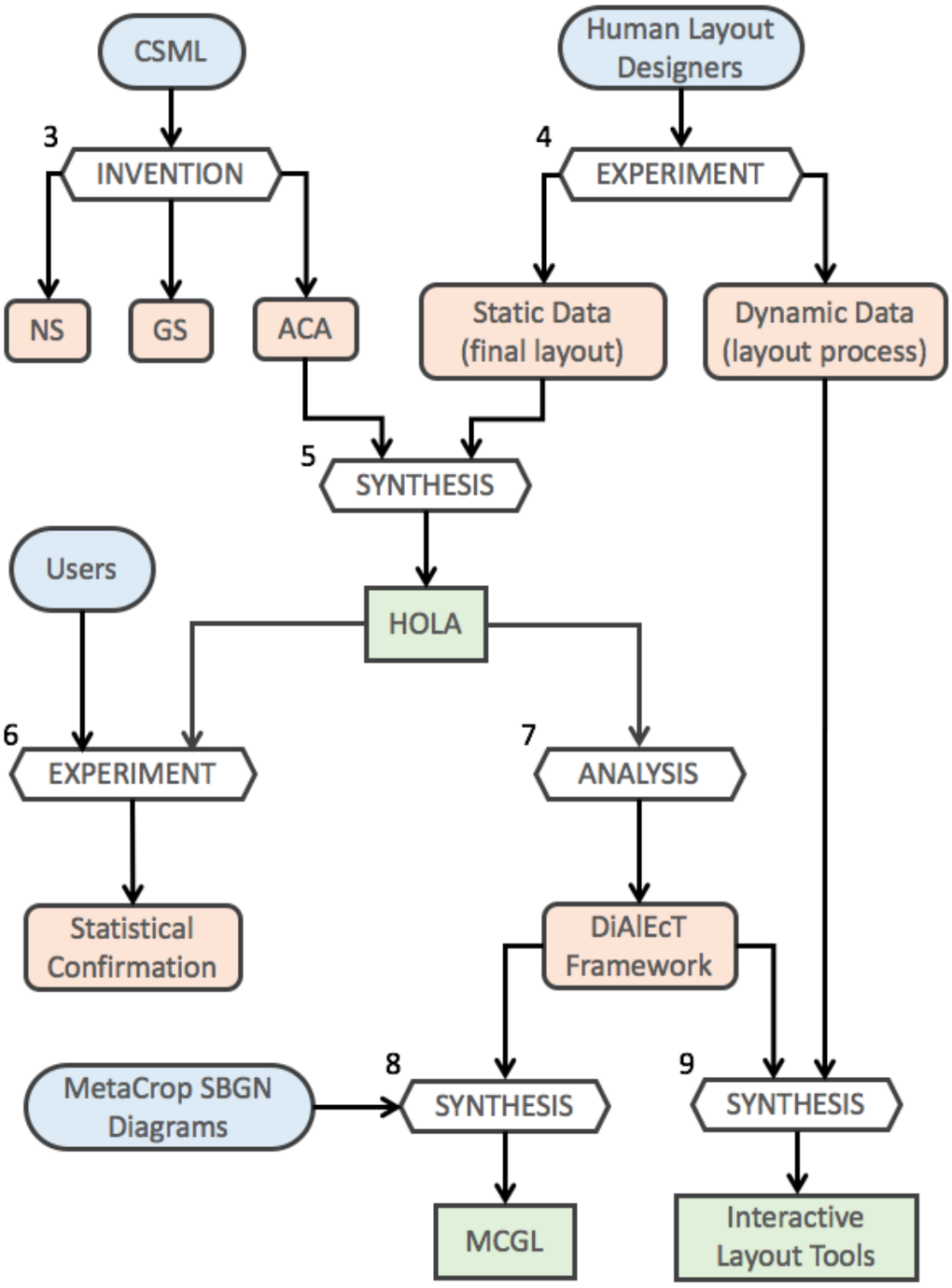


Figure 1.3: This flow chart represents the inputs (blue lozenges), processes (white hexagons), theoretical products (pink rounded rectangles), and concrete products (green rectangles) of this thesis. Each process node corresponds to one of the chapters 3 through 9, and the chapter numbers are indicated in the figure.

In Chapter 4 (EXPERIMENT node top-right in Figure 1.3) we move from theory to practice and begin to gather data through a user study, namely, the Orthowontist study mentioned in Section 1.2. This is the "formative" study of our human-centred methodology (Section 1.1). The experiment produced two kinds of data, which we call "static" and "dynamic". The former refers to the final layouts produced by the study participants, while the latter refers to their editing processes. To give the reader a sense of the information represented by the dynamic data, a flip-book has been drawn on the lower-right corner of the pages starting in Chapter 4. This shows the playback of "Layout 8A" as defined in Section 9.1.

It is in Chapter 5 (SYNTHESIS node, middle of Figure 1.3) that the two strands of theory and practice first come together. The basic ideas of Chapter 3 are augmented in an effort to achieve the special layout patterns that were observed in the user study of Chapter 4. This results in an algorithm called HOLA, for Human-like Orthogonal Layout Algorithm. However, like Chapter 3, the effort here is still intuitive; that is, while we create an algorithm, we are not yet in a position to say *how* we created it. That must wait until Chapter 7.

Chapter 6 (EXPERIMENT node lower-left in Figure 1.3) evaluates the effectiveness of HOLA, through another user study – the "normative" study of our human-centred methodology. Layouts created by HOLA are compared to hand-made ones, as well as to layouts generated by the state-of-the-art orthogonal layout algorithm available in the YFILES library [WEK04]. The latter uses the Topology-Shape-Metrics approach to orthogonal layout (see Section 2.4), so we refer to these diagrams as TSM layouts. The study shows that both in terms of user preference and performance on standard tasks, HOLA's layouts are comparable to human-made ones, and preferable to TSM layouts.

In Chapter 7 (ANALYSIS node in Figure 1.3) we distil out of HOLA's design a general method for the creation of automatic layout algorithms approximating human performance on various network layout dialects. This is the DiAlEcT framework introduced in Section 1.3. Thus, Chapter 7 represents the culmination of the theoretical side of the thesis, in that it crystallises the *intuitive* work from Chapters 3 and 5 into something *methodical* that can potentially be applied again and again in the future.

In Chapter 8 (SYNTHESIS node bottom-left in Figure 1.3) we verify the methodical applicability of the DiAlEcT framework by applying it to the MetaCrop dialect of the SBGN language, to generate a new algorithm called MCGL[3], for MetaCrop Graph Layout.

In Chapter 9 (SYNTHESIS node bottom-right in Figure 1.3) we study the human designers' layout processes from the Orthowontist experiment. Inspired in part by this, we formulate interactive tools for Phase-T (*Transform*) of the DiAlEcT framework.

The exposition is thus somewhat heuristic, in that instead of jumping straight to the DiAlEcT framework, we take the time to reach it in a natural way. Chapter 3 represents purely intuitive thinking; Chapter 5 revises those ideas in light of facts from the real world, but still guided by intuition; finally Chapter 7 systematises. This approach has the virtue of reflecting the actual course of development of the ideas presented here.

Miniature versions of Figure 1.3 appear at the start of Chapters 3 through 9.

1.7 Contributions

- We develop a new, human-centred methodology for network layout algorithm design, as described in Section 1.1, and demonstrated in depth in Chapters 4, 5, and 6. This starts with observation of the techniques of living, human experts, which are then translated into algorithms, and finally the performance of the algorithms is

[3]MCGL is pronounced like the name, "McGill". Alternatively, Melburnians may prefer to call it "MCG Layout", in honour of their beloved Melbourne Cricket Ground.

assessed by user study. This method gives us a way to try to make computers into ideal assistants for laying out node-link diagrams. Since we need to observe the techniques of human layout experts, we have restricted attention to the kinds of sparse networks (as defined in Section 1.5) that people commonly lay out by hand, using special layout styles.

- The empirical findings on human layout of orthogonal node-link diagrams gathered in the experiment of Chapter 4 are a contribution in themselves, in that they may be useful to other researchers, beyond the way in which they have been interpreted and used in this thesis.

- In terms of tools for end users, three have been designed and implemented in the course of this work:

  - The HOLA algorithm of Chapter 5 for orthogonal layout: This is the first layout algorithm to be designed based on a study of hand-made layout, and then shown in testing (Chapter 6) to perform comparably to human designers. It also outperformed the current state-of-the-art orthogonal layout algorithm.
  - The MCGL algorithm of Chapter 8 for SBGN layout in the MetaCrop dialect
  - The interaction tools developed in Chapter 9: These techniques provide a way to overcome the limitations of the automatic layout algorithms.

- Finally, the most far-reaching contribution of this work may be the DiAlEcT framework of Chapter 7, since this provides guidelines for future developers to create more tools of the kind just mentioned, for other layout dialects not considered in this thesis.

Chapter 2

Background

In this chapter we begin with a brief look at the field of graph drawing, or network layout, in general (Section 2.1). After this there are three main threads in the history of network layout that must be understood as background to the present work. The first of these is constrained stress-minimising layout, or CSML, which serves as the foundation for the techniques developed in this thesis (Section 2.2). The second is rule-based layout, where attempts were made in the early 1990s to make computers design network layouts as humans do (Section 2.3). The third is Topology-Shape-Metrics (TSM) layout, which is the current state-of-the-art approach to creating orthogonal diagrams (Section 2.4). While work on the evaluation of layouts is another important part of the background for this thesis, discussion of that topic is postponed until Section 4.1, where it can be put in closer proximity to our own evaluation work.

2.1 Graph Drawing

A *graph* $G = (V, E)$ is a mathematical abstraction, consisting of a set V of *vertices*, and a set $E \subseteq V \times V$ of *edges*. It can represent any system in which things (the vertices) are connected to one another (by the edges). A graph is therefore the underlying mathematical model when we are interested in *networks*, as discussed in Chapter 1.

In particular, when we are interested in *visualising* a given network, there are various techniques that can be used, but in this thesis we are interested only in *node-link diagrams*.[1] The problem of generating good node-link diagrams may be known as the *graph drawing*, *graph layout*, or *network layout* problem. The last term, network layout, may be used to emphasise a practical interest in computing diagrams for visualisation purposes, while the first term, graph drawing, refers more to the underlying mathematical theory. While this thesis is primarily concerned with the practical production of node-link diagrams, we may take a moment here to consider its mathematical roots.

The field of graph drawing itself is rooted in the more general study of *graph theory*, which is generally regarded as having been inaugurated in two papers by Euler: one presented to the St. Petersburg Academy in 1736, in which he solved the famous "Bridges of Königsberg" problem by representing the landmasses and bridges of Königsberg as the vertices and edges of a graph [Eul41], and another paper presented to the Berlin Academy in 1750, in which he established his now-famous formula

$$v - e + f = 2$$

[1] Other techniques include for example matrix representations, starting with Bertin [Ber83], or hybrid approaches [HF06, HFM07].

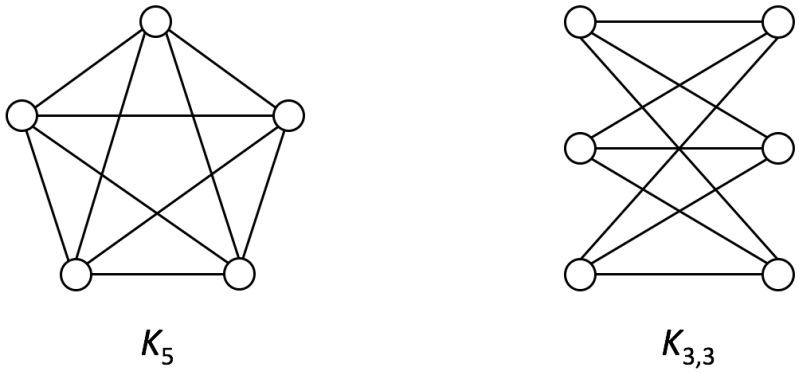


Figure 2.1: For each positive integer n we denote by K_n the complete graph on n vertices, i.e. the unique graph with n vertices every pair of which is connected by an edge. K_5 is depicted on the left. For positive integers m, n we denote by $K_{m,n}$ the complete bipartite graph with m vertices on one side and n vertices on the other side; this means that each vertex on one side is connected to each vertex on the other side. $K_{3,3}$ is depicted on the right.

relating the numbers v of vertices, e of edges, and f of faces of a polyhedron [Eul58].[2]

Since Euler's time mathematicians have studied the properties of graphs as a field of pure mathematical research, and among the questions they have explored have been ones regarding the embedding of graphs in the two-dimensional plane. It is this particular focus in the field of graph theory that is called graph drawing, in reference to the idea of "drawing" the graph in the plane, as we may draw a picture of the graph on a page.

Formally, a *planar embedding* of a graph $G = (V, E)$ is a map $\varphi : G \to \wp(\mathbb{R}^2)$ sending each vertex $v \in V$ to a point $\varphi(v)$ in the plane, and each edge $e = (u, v) \in E$ to a simple curve $\varphi(e)$ in the plane connecting the points $\varphi(u)$ and $\varphi(v)$, in such a way that none of the curves $\varphi(e)$ for $e \in E$ intersect one another, i.e. there are no edge crossings.[3] (Meanwhile to say that the curves are *simple* means that they do not intersect themselves either.)

Foremost among the mathematical questions we can ask about planar embeddings is the question whether they are possible for all graphs, and, if not, for which graphs they do exist. It turns out that not all graphs have planar embeddings. Those that do are called *planar graphs*, and they are characterised negatively by a theorem of Kuratowski, which states that a graph is non-planar if and only if it contains a subgraph isomorphic to a subdivision of K_5 or $K_{3,3}$ (see Figure 2.1).[4]

In the tomes of graph drawing many further properties about planarity have been studied. Just to give a flavour of these, we may ask for example whether a given graph admits a *convex embedding*, which is a two-dimensional embedding in which all of the faces are convex. Myriad questions of this kind have been investigated.

The first graph drawing *algorithms* seem to have appeared in the year 1963: one in the paper, *How to Draw a Graph*, by Tutte [Tut63]; and one in the paper, *Computer-Drawn Flowcharts*, by Knuth [Knu63].[5] The two algorithms point out nicely the distinction

[2]Note that Euler wrote in Latin, so that he spoke not of vertices, edges, and faces, but of *angulorum solidorum*, *acierum*, and *hedrarum*, respectively; nevertheless, it would seem to be due to its roots in this paper of Euler that the modern subject of graph theory continues to use the geometrical terms *vertex* and *edge*, despite graphs being abstract, not geometrical objects. Meanwhile the use of the term *graph* has an entirely separate history, having been introduced (according to Biggs et al. [BLW76]) by the English mathematician J.J. Sylvester in 1878 [Syl78], by analogy to molecular diagrams that were already in use at the time, known as "chemicographs".

[3]We use $\wp$ to denote the power set. Thus φ maps each vertex to a singleton subset of the plane, but for convenience we say that it simply maps each vertex to a point.

[4]The reader is referred to Harary [Har] for the formal definitions of graph isomorphism and subdivision.

[5]On the question whether these are indeed the earliest known graph drawing algorithms, see Eades and Hong [EH12] who consider the possible primacy of Tutte, and Di Battista et al. [DBETT99] on that of Knuth.

between *pure* and *applied* graph drawing. Tutte, an English-born mathematician who spent most of his life and career in Canada, gave an algorithm to compute a convex embedding for a planar graph, and the algorithm was given in service of a *proof* that such a convex embedding *exists*. Knuth, on the other hand, a giant in the history of computer science who needs no introduction, was concerned with the *practical* computation of flow chart diagrams to aid in the understanding of computer programs. In the present work we are concerned with the applied side of the subject. Accordingly, we mostly abandon the term "graph drawing" in favour of the term "network layout" in the remainder of the thesis.

Another distinction between the pure and applied sides is that in pure graph drawing we use the terms "vertex" and "edge", and vertices are zero-dimensional points. In applied network layout we still speak of vertices and edges when considering the underlying mathematical model, but tend to use the terms "node" and "link" for the graphical objects representing these, and nodes are drawn as shapes—most often boxes—with width and height. Links are also sometimes called "connectors", especially in the context of routing, i.e. computing a path along which to draw the link or connector.

2.2 Constrained Stress Minimising Layout

In this section we trace the origins of stress-minimising layout from two sources: force-directed layout, and multidimensional scaling. We then examine the introduction of constraints. Thus we are now concerned with practical network layout in the tradition we associated with Knuth [Knu63] in the previous section. Interestingly however, we will see that even here there is a connection with Tutte [Tut63] (see page 15).

2.2.1 Physical Analogies

The idea of *force-directed network layout* is to treat a graph as though it were a physical system, the nodes connected by springs, so that a layout may be computed by allowing the system to settle into an equilibrium position. We will examine the multiple, independent origins of this idea, as this lends weight to the idea being a very natural one. The technique was first invented by researchers working on the automatic layout of electronic circuit boards, and again about twenty years later by a researcher in the area of data visualisation. Understandably, the latter was not familiar with the former's literature.

In the spring of 1964 the Sandia Corporation, a United States laboratory in Albuquerque, New Mexico, initiated a large project in collaboration with the Thomas Bede Foundation in California to automate the design and production of printed circuit boards. As a part of this project the technique of force-directed layout was developed, and was published in a paper by lead author Fisk [FI65] (see also [FCW67]). It was written in Fortran II for the IBM 7090 computer. The authors referred to the method as "force placement" and the metaphor of the network as a physical system is found in their paper:

> Imagine that each of the actual components represented on this schematic has elastic leads which are in a state of tension until they contract to some arbitrarily short length. ... If the components were suddenly released from their pinned positions, each would accelerate according to the instantaneous resultant force exerted on it. The components would move into a cluster and stop moving only after resultant forces acting on each component provided enough deceleration to stop all movement. [FI65]

The metaphor is fine as a form of motivation, but, as the authors went on to observe, to perform an accurate computer simulation of such a physical system obeying the true

physical laws of our universe would be highly computationally expensive. On the contrary, they noted that in the world of simulation we are free to write our own physical laws, which can be much simpler, and yet still achieve the desired results. To that end Fisk et al. formulated two forces to act between nodes, one attractive and one repulsive. In order to state the force formulae that they used, let us define a *weighted distance* function

$$d(\alpha, \beta; u, v) = \sqrt{\alpha^2 (x_u - x_v)^2 + \beta^2 (y_u - y_v)^2}$$

where α, β are positive real numbers, u and v are two nodes in a graph, and for any node t the coordinates of its centre point are (x_t, y_t). If the width and height of node t are w_t and h_t then the force formulae in Fisk et al. were

$$F_A(u, v) = K_A d(w_u + w_v, h_u + h_v; u, v) \quad (2.1)$$

$$F_R(u, v) = -K_R d(w_u + w_v, h_u + h_v; u, v)^{-1}. \quad (2.2)$$

Note: they did not present these formulae in terms of the weighted distance function, and I have only introduced that as a notational convenience here. Nor did they discuss the reason for the weightings. But it is clear that, irrespective of the distance between the centres of two nodes, Fisk et. al. wanted the attractive force between them to decrease if the nodes were larger, almost as if the nodes were made of some medium that only partially transmitted the force. On the other hand, the repulsive force increased with the thickness of the nodes.

Twenty years later Peter Eades, an academic researcher who had been working on a graph editing system called TYGES (Typed Graph Editing System), independently reinvented the force-directed technique, this time for the purposes of data visualisation. He too described a physical metaphor:

> To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system.... The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state. The algorithm outputs the positions of the vertices in this stable state. [Ead84]

Eades too realised that his "springs" did not actually have to obey Hooke's law,

$$F = -k(x - \ell_0)$$

(k the spring constant, ℓ_0 the natural length of the spring, x the current length) and he could instead take the liberty to define whatever forces would work best. In fact his paper shows that he initially tried Hooke's law but found that the attractive forces became too strong when the nodes were far apart. In its place he defined a force with logarithmic form to operate between neighbouring nodes (those connected by an edge), and again for purposes of easy comparison I present it here using the weighted distance function, though Eades did not:

$$F_1(u, v) = C_1 \log [d(1, 1; u, v)/C_2] \quad (2.3)$$

while another repulsive force was required to operate between all other pairs of nodes, to keep them from coming too close together:

$$F_2(u, v) = \frac{C_3}{d(1, 1; u, v)^2}. \quad (2.4)$$

The constants C_1, C_2, C_3 were to be determined empirically, and Eades reported values that had been successful on many graphs so far. Note that in the force F_1 the constant C_2

is similar to the natural length ℓ_0 in Hooke's law, since the sign of the force goes to zero here and flips between positive and negative on either side of it, so that C_2 represents the natural distance by which two nodes "want" to be separated.

While we can certainly say that the idea of force-directed layout was invented at least twice, by Fisk's group and independently by Eades, it is sometimes claimed that the idea was already present in Tutte's landmark paper in pure graph drawing, *How to Draw a Graph* (see Section 2.1); however, I would suggest that this claim is far too strong. Tutte's method is known as the *Barycentre Method* and, as demonstrated for example in Section 10.2 of the graph drawing textbook by Di Battista et al. [DBETT99], the Barycentre Method *yields the same layout as* a force-directed system in which the springs obey Hooke's law and have a natural length of zero, and in which there are no repulsive forces. But to go from equivalent layouts to the *idea* of force-directed layout actually having been present in Tutte is far too generous. While we saw in the quotations above that both Fisk and Eades explained their idea through the explicit use of a metaphor of physical forces, such a metaphor is completely absent from Tutte's paper. Instead, a fairer assessment is that the Barycentre Method of Tutte, while being conceptually distinct from the idea of force-directed layout, remarkably turns out to be *mathematically equivalent*.

Energy, and Faithful Springs

After Eades's inauguration of the idea in the field of data visualisation in 1984, force-directed layout became a subfield of the discipline in its own right, with many other researchers exploring variations on the force model and convergence conditions, in works that are surveyed in Chapter 10 of Di Battista et al. [DBETT99].

Among these later variants the approach begun in 1989 by Kamada and Kawai [KK89] is foundational for the techniques employed in this thesis. Recall that in physics energy is the integral of force over distance, and this time the physical metaphor was realised not in terms of forces but in terms of an *energy function*:

$$E = \sum_{i<j} \frac{1}{2} k_{ij} \left(|p_i - p_j| - l_{ij} \right)^2 \tag{2.5}$$

where p_i was the position of node i, and thus $|p_i - p_j|$ the distance between nodes i and j, and where k_{ij} and l_{ij} were the rigidity constant and natural length for the spring connecting nodes i and j. The move to an energy function was in itself an important change, paving the way for the use of classical optimisation techniques, as we will examine in Section 2.2.3. Moreover, while the energy was ostensibly just that of a system of springs obeying Hooke's law, Kamada and Kawai introduced two crucial innovations in their definition of the spring constants, so that the energy would encode more information about the structure of the graph.

Their first innovation was in the choice of the natural length l_{ij} of each spring, and their idea was to let this length reflect the connectedness of each pair of nodes in the graph. Namely, for any pair of nodes connected by an edge, there should be some basic spring length L; for any pair of nodes whose shortest connection in the graph was a path of two edges, the spring length should be twice as much, or $2L$; in general, the spring length should be nL for any pair of nodes separated by a minimal path of n edges. Formally, if d_{ij} is the *graph-theoretic distance* between nodes i and j, that is, the length in edges of the shortest path between those nodes in the graph, then we define

$$l_{ij} = d_{ij} L$$

for some *ideal edge length* L, presumably to be chosen in reference to the average size of the nodes, or the desired size of the drawing, or both. The ideal edge length L will be an important parameter in all the algorithms to be developed in this thesis.

The importance of defining the spring lengths l_{ij} in this way has been perhaps best expressed only decades later by Nguyen, Eades, and Hong [NEH13], who have characterised the layout resulting from such a spring system as a "faithful" representation of the graph. The layout is called faithful because, ideally, the geometric distances reflect the abstract graph-theoretic distances.

The idea of stress-minimal layout as faithful layout is central in this thesis.

The second innovation of Kamada and Kawai was in their choice of the rigidity constant k_{ij} for each spring. They realised that the contribution of each pair of nodes to the total energy of the system must not be allowed to grow with the graph-theoretic distance d_{ij} between those nodes. For supposing all spring constants k_{ij} were equal to a single value k, then consider a pair of nodes u, v separated by a shortest path of ten edges. Their ideal separation would be $10L$, so that a real separation of $11L$ would contribute $\frac{kL^2}{2}$ to the total energy of the system. Meanwhile a pair of neighbouring nodes s, t with a real separation of $2L$ would contribute exactly the same amount of energy. In principle, this seems wrong. The deviation of u, v from their ideal separation may be barely perceptible to the eye, whereas the separation of s and t by *twice* the desired amount will be glaring. What becomes apparent is that it is the percentage error in each ideal separation that matters, not the absolute error, and therefore Kamada and Kawai defined k_{ij} to be inversely proportional to the square of the graph-theoretic distance d_{ij}:

$$k_{ij} = \frac{K}{d_{ij}^2}$$

for some constant K.

Plugging the definitions of l_{ij} and k_{ij} into the energy formula (2.5) gives

$$\begin{aligned} E &= \sum_{i<j} \frac{1}{2} \frac{K}{d_{ij}^2} \left(|p_i - p_j| - d_{ij}L\right)^2 \\ &= \frac{K}{2} \sum_{i<j} \left(\frac{|p_i - p_j|}{d_{ij}} - L \right)^2 \end{aligned}$$

and since we are only interested in *minimising* this function, we may as well drop the constant factor, leaving us with a layout that is determined when the quantity

$$\sum_{i<j} \left(\frac{|p_i - p_j|}{d_{ij}} - L \right)^2 \tag{2.6}$$

is minimised. It is in this form that the innovations of Kamada and Kawai are most readily apparent, while the origin in Hooke's law is now scarcely visible. Their use of percentage error rather than absolute error, and their realisation that a "faithful" layout will make geometric distances proportional to graph-theoretic ones, are both immediately apparent in this form. For we now have an ideal edge length L which, put differently, is the ideal ratio between geometric and graph-theoretic separation for each pair of nodes, and the costs accumulated in (2.6) are precisely the squared deviations from this ideal.

We see in the next section how this particular energy function came to be known as "stress".

2.2.2 Multidimensional Scaling

Just as the force-directed layout method was pioneered by Fisk et al. long before a later rediscovery of the idea by Eades would capture the imagination of mainstream researchers in graph drawing, the very same thing happened with the "faithful" layout of Kamada-Kawai. This time the earlier work was presented at the First General Conference on Social Graphics[6] in October 1978 by Joseph B. Kruskal[7] and Judith B. Seery [KS80], about a decade before Kamada-Kawai. In this case, however, there was no spring model, or any physical metaphor at all. Kruskal and Seery instead approached the problem of laying out a graph from the standpoint of *multidimensional scaling*.

Multidimensional scaling is an approach for visualising high-dimensional data in 2- or 3-space, the idea being to represent each higher dimensional point by a point in lower-dimensional space in such a way as to best preserve the distances between the original points. (This allows the detection of clusters or other trends via human inspection of a 2-D or 3-D plot.) In other words, if we are given a set of n-dimensional points $X_1, X_2, \ldots, X_m$, and the distance in n-space between points X_i, X_j is d_{ij}, and if each X_i is to be mapped to a point Y_i in 2- or 3-space, then we want to choose the points Y_i so as to minimise value of the expression

$$\sum_{i<j} \left(|Y_i - Y_j| - d_{ij}\right)^2 \tag{2.7}$$

which bears a striking resemblance to the later spring energy (2.6) of Kamada-Kawai. Here then is yet another example of a single mathematical form, arrived at by conceptually distinct routes by different researchers, lending a sense of necessity and naturalness to that form.

The idea of a "faithful" representation is inherent in the idea of multidimensional scaling, or MDS, and Kruskal and Seery saw that this was a suitable way to lay out the nodes of a network for the purpose of creating a diagram. They deserve great credit for perceiving this in an era when the very idea of the computer-aided layout of node-link diagrams was new.

The newness of this idea in 1978 is evident in the way Kruskal and Seery motivated their paper. For example their abstract allows that,

> The process of designing such diagrams has not generally been recognised as a problem....

Later, after admitting that interactive computer graphics could be one approach to creating network diagrams (while noting the scarcity of access to the appropriate equipment), they explained that they instead took an algorithmic approach:

> ...we describe a fully algorithmic method which starts from a description of which nodes are connected to one another and yields suggested locations for the nodes.

in a description which, for its careful enunciation of something now so utterly commonplace, reads as if it might have been among the very first statements ever made of the algorithmic approach to network layout.

Credit is also due to Kruskal and Seery for anticipating many of the important aspects of network layout research which would begin to be examined only decades later, and which are still being examined in this thesis. They observed that,

[6] The conference was organised by the U.S. Bureau of the Census.

[7] This is the same Kruskal of the famous minimum-spanning-tree algorithm, and not to be confused with his brother William, after whom the Kruskal-Wallis statistical test is named in part.

> Although judgment of quality is necessarily a subjective process, quality could be studied by the methods of psychology.

and they indicated that they were not aware of any such studies in this area. To my knowledge, the first formal studies into what makes a good diagram only began in the 1990s with the work of Helen Purchase and others [PCJ96, Pur97, PCJ97, Pur98, Him95].

Kruskal and Seery also considered whether anyone had yet documented the way in which people create network diagrams, but stated that,

> We are not aware of any published discussion of the "conventional" process of designing such diagrams....

Formal studies of both these kinds, that is, into the process whereby people create network diagrams, and into the qualities that make good diagrams, are carried out in Chapters 4 and 6 of this thesis.

The MDS statistical technique itself dates from the 1950s [Tor52] and was further developed in the 1960s by several researchers [Sam69, Kru64], among them Kruskal, who introduced the term "stress" for a variant[8] of the expression in (2.7). Whereas Kamada and Kawai were unaware of MDS and the stress function, Cohen [Coh97] seems to have been the first graph drawing researcher after them to revive the term "stress" and make the connection with Kruskal and MDS. The idea of the stress function was then picked up in subsequent papers on network layout [KH03, GKN04, DKM05] and remains current today.

2.2.3 Constraints

The paper by Kamada and Kawai on what we may now call the "stress energy" (see Section 2.2.1) was based on the former's PhD work completed at the University of Tokyo around the same time, in 1988. Concurrent with this, Kamada published a book describing his work more fully [Kam89] and containing a whole chapter on the subject of *constraints*.

By constraints we simply mean any requirements placed on the coordinates of the nodes. For example a constraint may require that several nodes have the same y-coordinate (so that they are horizontally aligned) or that one node have smaller x-coordinate than another (so that it lies to the left of the other).

That constraints should be added atop basic force-directed layout is indeed in keeping with the spirit of Eades's paper on the subject in 1984, where he concluded that the method was at least promising as "a first approximation to a layout," [Ead84] which could then be "fine tuned". Constraints provide a way of fine-tuning an initial force-directed layout, and this is the basic idea of CSML (i.e. constrained stress-minimising layout, a subset of constrained force-directed layout).

In his book, Kamada considered what he called "low-level" constraints such as alignments and orderings on x- and y-coordinates, as well as "high-level" constraints obtained by combining these. For example, he had a constraint called "CIRCULARLISTING" which would constrain a list of nodes to be arranged evenly spaced on a circle around a central node.[9] Kamada's constraints were given as linear *equations*—inequalities were not possible—and the constrained stress-minimisation problem was solved using a technique based on a simple equation solver [VW82].

[8] The history has been nicely surveyed by Cohen [Coh97] in a paper in which he presented a faster incremental algorithm than that of Kamada-Kawai, which also had less tendency to get caught in suboptimal local minima. Among other things, Cohen sorts out the different functions that have been called "stress."

[9] This may be compared with the VHUB directive of Marks discussed in Section 2.3.2 as well as with our regular polygonal face arrangements in Section 8.4.2.

Kamada's system was called COOL (for COnstraint-based Object Layout), and was perhaps the first ever CSML layout system. From this starting point, technological innovations would come in both the way the constraints were formulated, and the way the stress function was minimised subject to these.

Where Kamada and Kawai used a Newton-Raphson iteration to minimise the stress function, He and Marriott [HM96] improved on this by using *active set methods* [Fle13] to minimise the stress function subject to arbitrary *linear constraints*.

A linear constraint is an equation *or inequality* of the form

$$a_1u_1 + a_2u_2 + \cdots + a_nu_n = b \qquad \text{or} \qquad a_1u_1 + a_2u_2 + \cdots + a_nu_n \leq b$$

where $a_1, a_2, \ldots, a_n, b$ are constants, and $u_1, u_2, \ldots, u_n$ are the same variables on which the *objective function*—the *stress* function in this case—depends. When for the variables we take the x- and y-coordinates of the nodes in a diagram, linear constraints permit imposition of spatial conditions on the layout, such as that one node appear above or to the left of another, or that several nodes be horizontally or vertically aligned. Moreover, the ability to use linear constraints with inequalities meant that now it was possible to constrain one node to lie on a certain side of another node, but without having to set the exact distance between them, a crucial advance, necessary for all the techniques developed in this thesis.

Later, Gansner et al. borrowed the technique of *stress majorisation* [GKN04] from the multidimensional scaling community. There it had been introduced by de Leeuw and Stoop [DLS84] under the name SMACOF ("Scaling by MAjorizing a COmplicated Function"). This works by minimising a pure quadratic function formulated as an upper bound on the stress function (which, due to the involvement of Euclidean distance, contains square-root terms). Dwyer and Koren [DK05] built on this by using quadratic programming [NW99] to perform stress majorisation in the presence of a special type of constraint (called "band constraints") that could be used to organise nodes into layers for *layered layout* [STT81].

Subsequently, Dwyer et al. [DKM06] developed a similar system that could now handle arbitrary *separation constraints*. A separation constraint is an equation or inequality of the form

$$u + a = v \qquad \text{or} \qquad u + a \leq v$$

where a is a constant and u, v are variables. Separation constraints are so called because they constrain the difference i.e. "separation" between two variables to be either exactly equal to a certain value, or bounded below by a certain value. The restricted form of separation constraints allowed Dwyer et al. to develop a fast, custom *gradient-descent* algorithm for minimising stress subject to these.

This allowed solving much faster than the system of He and Marriott, and yet, remarkably, nearly every kind of linear constraint that is desirable when arranging the nodes of a node-link diagram can be achieved with separation constraints alone. In particular these allow us to enforce separations, alignments, and distributions amongst nodes in both the x- and y-dimensions.[10] Moreover, by updating the set of separation constraints dynamically between iterations of the gradient-descent algorithm even non-linear constraints like cluster containment and node non-overlap can be achieved [DMS06].

The method is fast enough for interactive use with graphs of up to about two hundred nodes on current processors, as demonstrated in the network layout editor Dunnart [DMW09a, WMMS08]. This provides continuous stress minimisation while the user

[10] A few useful constraints are however not achievable by combining separation constraints. Among these is the "variable distribution," i.e. a constraint stipulating that three or more nodes be spaced evenly but by a variable distance. Others are circular constraints or radial spacing constraints.

edits the layout by moving nodes manually and applying separation constraints. Node positions can be smoothly animated as stress is minimised due to the speed with which the gradient-projection algorithm works.

As is normal for optimisation methods of this kind, the algorithm finds a local stress minimum subject to the given constraints, not necessarily a global one. There are known techniques such as *simulated annealing* [DH96] for working around this, but in practice we have found that it is not necessary. The local minima discovered by the gradient-projection algorithm lead to satisfactory layouts (see Chapter 6), while interaction methods (see Chapter 9) provide ways for the user to move the layout into other (perhaps deeper) stress basins, if desired.

2.2.4 Adaptagrams

The methods of Dwyer et al. for CSML subject to separation constraints are available along with techniques of Wybrow et al. [WMS10] for connector routing, in the Adaptagrams layout library.[11] This library provides the foundational techniques for all layout methods developed in this thesis. In particular, we make use of five operations provided by the Adaptagrams library: PROJECT, DESCEND, OVERLAP-REMOVAL, POLY-ROUTE, and ORTHO-ROUTE.[12]

The first two operations, PROJECT and DESCEND, are understood as mappings $\mathbb{R}^{2n} \to \mathbb{R}^{2n}$ where n is the number of nodes in the network being laid out, and we think of a vector in $\mathbb{R}^{2n}$ as representing the coordinates of all n nodes. We refer to such a vector as a *layout vector*. Both operations are also understood as operating relative to a given set C of separation constraints (see previous section).

The PROJECT operation maps a given layout vector $\lambda \in \mathbb{R}^{2n}$ to the nearest point λ' (under the Euclidean metric) in the *feasible space*, i.e. the subset of $\mathbb{R}^{2n}$ in which all constraints in C are satisfied. It is so named because this is the mathematical operation of *projecting* onto the feasible space.

The DESCEND operation refers to the full gradient-projection algorithm introduced in the last section, which repeatedly alternates between applications of the PROJECT operation and *descent steps* in which the gradient of the stress function is used to guide movement to a new layout vector at which stress is lower. This is repeated until a halting condition is met, namely that either the change in stress falls below some small threshold $\varepsilon > 0$, or that the number of iterations exceeds some limit N, both constants determined experimentally.

Since DESCEND may involve many iterations, each of which includes an application of PROJECT, the former tends to be much slower than the latter. This is an important consideration in our algorithm design in this thesis.

The OVERLAP-REMOVAL operation serves to automatically generate minimal separation constraints necessary to remove node-node overlaps (it does not remove node-edge overlaps). The technique is described by Schreiber et al. [SDMW09].

The remaining two operations, POLY-ROUTE and ORTHO-ROUTE, are responsible for connector routing, i.e. for drawing the plane curves that represent edges in the network. The former creates *polyline connectors*, i.e. piecewise-linear curves, while the latter creates *orthogonal connectors*, a special type of polyline connector in which each line segment is axis-aligned, i.e. horizontal or vertical.

[11] `http://www.adaptagrams.org`

[12] These names are used not in the library itself, but only in this thesis.

2.3 Rule-Based Layout Design

In 1991 Joe Marks completed a doctoral thesis [Mar91b] at Harvard University, in which he developed a system called ANDD, for Automated Network Diagram Designer. Further developed in subsequent papers with colleagues Kosak, Shieber, Dengler, and Friedell [KMS94, DFM93], the system was intended to be able to create node-link diagrams according to the sorts of design rules followed by human graphic designers. In some ways this work presaged the project in the present thesis, while differing in important respects. This section gives a summary of the ANDD system, but a proper comparison with the DiAlEcT layout framework developed in this thesis must wait until Chapter 10.

2.3.1 Perceptual Organisation

Marks and colleagues described the idea of *perceptual organisation*, and added this to the list of basic goals of node-link diagram layout. Previously the main concerns had been *syntactic validity* and *aesthetic optimality*, the former covering basic requirements like non-overlap of nodes, and the latter encompassing all the classic layout goals such as minimisation of edge crossings and bends, maximisation of symmetry, and compaction of the diagram into the smallest area. To this Marks's group added,

> Syntactic validity and layout aesthetics do not, however, account for all the important aspects of network-diagram layout. For example, human graphic designers rely routinely on grouping principles derived from the classical Gestalt Laws of perceptual psychology...to organize diagrams visually. ... We therefore generalize the problem of network-diagram layout by introducing layout considerations that concern *perceptual organization*. [KMS94], p. 440.

Furthermore, they suggested that perceptual organisation be given priority over aesthetic considerations.

The perceptual organisation of a diagram was to be specified by a set of *visual-organisation features* or VOFs. These were said to,

> ...specify perceptual groupings due to various kinds of proximity relations, sequentially ordered layout, alignment, axial and radial symmetry, and special, easily recognised layout patterns, such as the "T-shape" pattern that describes a conventional layout for nodes that are related hierarchically. [KMS94], p. 442.

More recently, the phrase "perceptual organisation" has been used again by van Ham and Rogowitz [vHR08], who ran an experiment to learn about the kinds of organisations people create when working by hand. The experiment presented in Chapter 4 is in the same tradition, and we too will use the distinction between pure aesthetics on the one hand, and perceptual organisation on the other hand, throughout the rest of the thesis.

2.3.2 The Diagram Creation Problem

Marks divided the diagram creation problem into two subproblems: (1) the creation of what he called an *expressive mapping*, and (2) the layout instantiation problem, meaning the computation of node positions to satisfy the expressive mapping as well as basic aesthetic and syntactic criteria. The ANDD system was responsible for the first subproblem of designing the diagram; it was in subsequent papers that various approaches to the second (layout) problem were investigated.

Under the first subproblem, designing the expressive mapping, Marks took a much broader view of node-link diagram design than we take in this thesis, including choices of node shape and colour, stroke width and style, and clustering. For example, given

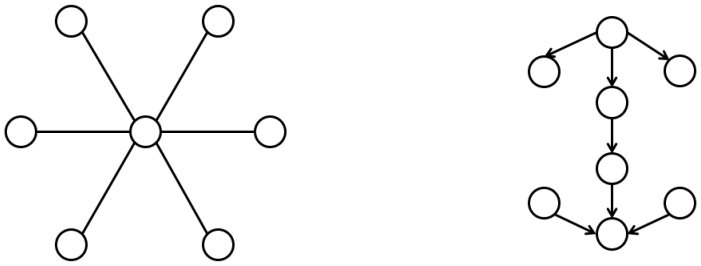


Figure 2.2: In Marks's system, applying the VHUB design directive to a set of nodes indicates that the layout system should attempt to arrange those nodes in a hub pattern (left). Applying the SSP design directive to a path of nodes in a directed graph, starting with a source and ending with a sink, indicates that the layout system should attempt to arrange those nodes in a straight line (right).

the same diagram design task, his system ANDD could in one instance choose to mark all nodes of a given type by drawing them in the same colour, while in another instance it could instead decide to indicate the similarity of these nodes by drawing them in a cluster inside a bounding box. Marks motivated this system with statements like, "Good designers use the entire gamut of graphical properties to communicate information," and, "Network diagrams have a mutable semantics." [Mar91a]. In this respect his emphasis was quite different from our own. For example, when we work with SBGN in Chapter 8, all diagram aspects such as node shape, stroke style, and the question of placing nodes inside of boxes, are completely prescribed by the SBGN specification itself. Our only task here is to choose node positions.

Where the perceptual groupings or "VOFs" mentioned above entered the ANDD system was through what Marks called *pragmatic design directives*. There were five of these: VHUB, SSP, FL, IN, and OUT, standing respectively for "visual hub", "source-sink path", "feedback loop", "inputs", and "outputs". These were descriptions of certain graph-theoretic substructures that might be present in the given network, and, when desired, it was up to the user to tell the system to apply one of these directives to a substructure for which it was suitable. For example VHUB could be applied to a node v and all its neighbours u to indicate that the central node v should be thought of as a "hub" around which the neighbours u were to be spaced at a constant distance and at equal angles, like spokes on a wheel. For another example, SSP could be applied to a path of nodes starting with a source node (one sitting only at the source end of directed edges) and ending with a sink node (one sitting only at the target end of directed edges), to indicate that in the layout these nodes should be aligned. See Figure 2.2.

Once the expressive mapping was settled, the problem of creating a layout remained. Marks and colleagues tried three approaches to this problem: one rule-based approach implemented in Prolog, and two optimisation-based approaches, one using a Newton-type iteration [DFM93], and the other using a parallel-genetic algorithm [KMS94]. For our purposes there is no need to distinguish between the latter two; we need only think about the difference between the rule-based and optimisation-based methods.

These two approaches involve fundamentally different ways of responding to the design directives such as VHUB and SSP. In the rule-based method (described in greater detail in Section 2.3.3), design directives must be satisfied precisely. For example when a VHUB rule is applied nodes will be arranged exactly as on the left of Figure 2.2. In the optimisation-based methods each directive is instead merely represented by a term in a goal function, measuring deviation from the desired arrangement. This means that precise satisfaction of the directives is traded off for the ability to incorporate goal function terms representing global considerations, like overall number of edge crossings. In the final

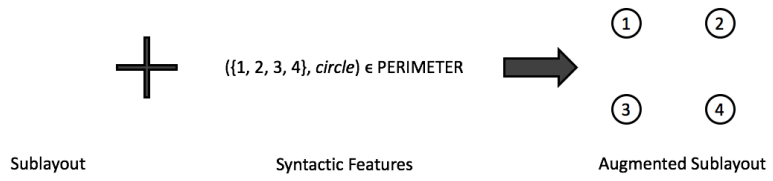


Figure 2.3: Recreation of Marks's Figure D.4

layout the design directives may be only approximately satisfied, such as in Figure 3 of Dengler et al. [DFM93], in which a VHUB directive has resulted in a slanted, imperfect version of the configuration on the left of Figure 2.2.

The task of producing layouts in which desired node arrangement patterns may be satisfied only approximately is an essentially different problem than the one we work on in this thesis. Of much greater interest in comparison with the present work is the rule-based approach of Marks et al., which we now review in more detail.

2.3.3 A Rule-Based Approach

In the rule-based system of Marks's group the "visual organisation features" (VOFs) had to be expressed exactly. Moreover, their characterisation of the rule-based approach shows it to be an attempt to achieve the very same ends I have set in this thesis:

> Our motivation in using heuristic rules for layout is to emulate how human graphic designers appear to lay out diagrams. Similar sublayout patterns tend to recur repeatedly in human-designed network diagrams, suggesting that human designers utilise a small set of patterns when generating diagram layouts. This pattern-generation expertise can be captured to some degree in heuristic rules. [KMS94], p. 442.

Implemented in Prolog, it was a resolution-based search strategy. There were twenty-two basic rules (see Appendix D.2 in Marks's thesis [Mar91b]), of the form:

$$\text{Sublayout} + \text{Syntactic Features} \rightarrow \text{Augmented Sublayout}$$

indicating how the layout could be built up, a few nodes at a time, starting from an empty layout, i.e. with no nodes having any assigned position.

In order to get started, a rule would have to be applied in which the Sublayout term on the left-hand side of the rule was empty. For example, Figure D.4 in Marks's thesis gives a rule in which an empty Sublayout plus an expressive mapping including a Syntactic Feature called PERIMETER could result in the placement of four nodes on the perimeter of a square in the Augmented Sublayout. See Figure 2.3. Later, a rule like that in Marks's Figure D.7 could be applied to an existing Sublayout consisting of two vertically aligned nodes; the rule states that if the right Syntactic Features are present in the expressive mapping, then a new node may be added to the diagram, in vertical alignment with the two that are already present, and placed so that the three are evenly spaced. See Figure 2.4. In this way the layout is to be gradually built up by repeated application of the various rules.

As explained in Kosak et al. [KMS94], p. 445, some "weak" rules are intentionally included, so that some rule will always be applicable. The more complex and less-likely applicable rules are placed first in the list, so that Prolog will attempt to apply them first, but the list of rules given in Marks's thesis ends with a sort of catch-all rule, requiring no Sublayout and no Syntactic Features whatsoever, and allowing an Augmented Sublayout in which a single node may be positioned "arbitrarily." See Figure 2.5. In other words, this catch-all rule means that at any step of the rule-based approach at which no special rule

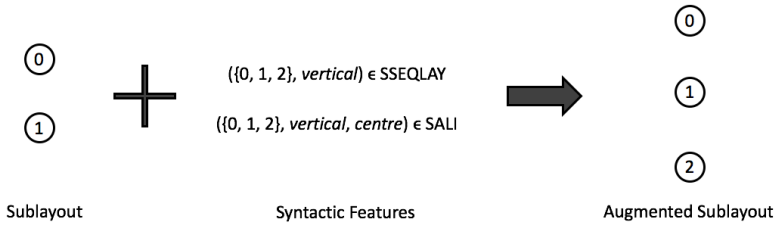


Figure 2.4: Recreation of Marks's Figure D.7

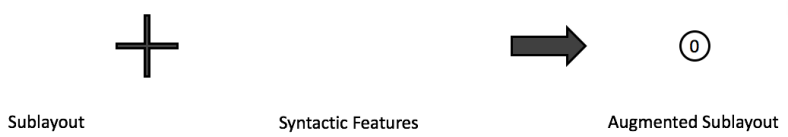


Figure 2.5: Recreation of Marks's Figure D.22

may be applied, a node is to be placed "arbitrarily". As will be seen in Chapter 10, this points toward a fundamental way in which the DiAlEcT system offers an improvement, namely, in node positions being guided by stress-minimisation even as perceptual node arrangements are precisely enforced by constraints.

As for the completeness and suitability of their rule base (i.e. the 22 Prolog rules), Marks and colleagues felt that it covered reasonably well the sorts of design patterns they had observed in existing diagrams, while acknowledging that in the future the rule base could be tailored to special diagram types or styles. They wrote,

> Though we do not believe that our set of VOFs is inherently exhaustive, we have noted over a period of time and through informal taxonomic research that our set of features provides excellent coverage in practice for the actual organisational primitives conventionally used in network diagrams by graphic designers. [KMS94], p. 442.

However, the user study described in Chapter 4 of this thesis represents a major contribution of the present work, going beyond mere "informal taxonomic research" as a way of learning about the kinds of node-link diagram layouts created by human designers.

Kosak et al. added that

> The current rule base used by ANDD is only one of probably many useful rule bases. Different rule bases might reflect different layout styles, and may be better for some kinds of network diagrams than others. The development and comparison of different rule bases is a possible goal for future research. [KMS94], p. 445.

Indeed, that goal is taken up in this thesis in our treatment of two different layout dialects: one for orthogonal diagrams and one for SBGN diagrams.

2.4 Topology-Shape-Metrics

Topology-Shape-Metrics, or TSM, is an approach for creating orthogonal node-link diagrams in three phases. It was developed by researchers at the University of Rome, Italy, and predates the work of Marks et al. by about five years. It took a purely aesthetics-based approach to network layout, the very approach on which Marks later aimed to improve

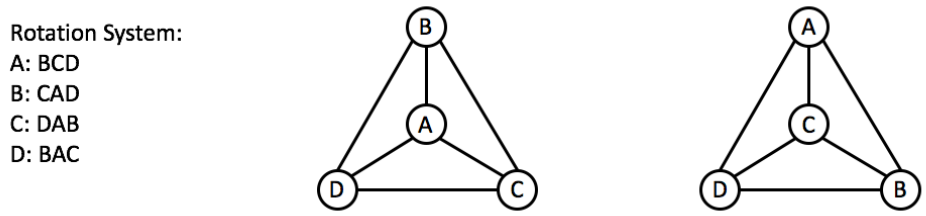


Figure 2.6: For each node, the rotation system lists that node's neighbours in clockwise order around it. Note that the two embeddings shown here of the graph K_4 have the same rotation system, but are not homeomorphic in the plane $\mathbb{R}^2$ (they are however homeomorphic on the 2-sphere). This is why in order to determine the topology of a planar embedding we must give not just the rotation system but also must say which is the external face.

by paying attention to what he called perceptual organisation. In this thesis too I have aimed to improve on TSM by creating an orthogonal layout technique (Chapter 5) that takes perceptual organisation into account, although my approach has been more robustly empirical than that of Marks, owing to my use of formal user studies (Chapters 4 and 6).

Appropriately, TSM layout was born out of not just an improvement but a *solution* to one of the aesthetic layout problems, to efficiently minimise the number of bends in the orthogonal connector routes while preserving a given planar embedding (see next Section). This came in the form of a procedure discovered by Roberto Tamassia whereby a given graph layout problem was translated into a corresponding network flow problem, so that by computing a minimum-cost flow one simultaneously obtained a bend-minimal layout. Tamassia's sole-authored article [Tam87] on the algorithm was published in June 1987 but had been received by the editors nearly three years earlier, in October 1984.

A family of graph layout algorithms have flowed from this original idea, and the very first of these was formulated by Tamassia and his colleagues Batini and Nardelli also in October 1984, right on the heels of the theoretical foundations, though this paper too was delayed in publication until 1986 [BNT86]. This second paper gave an algorithm for orthogonal layout in three phases, called *Planarisation*, *Orthogonalisation*, and *Compaction* (see Figure 2.7). These three phases determined, respectively, the so-called "topology", the "shape", and the "metrics" of the graph drawing being produced.

2.4.1 Planar Embeddings, Rotation Systems, and Topology

Before we can examine how the TSM procedure works, and the motivation for its design, we must pause to understand the connection between *topology*, *planar embeddings*, and *rotation systems*. These notions are also important for understanding the design of the DiAlEcT framework of Chapter 7, and will be next revisited in Section 5.2.2.

Recall the definition of a planar embedding of a graph, from Section 2.1. Given a planar graph G, a planar embedding $\varphi : G \to \wp(\mathbb{R}^2)$ defines a *rotation system*, which consists of the clockwise ordering of the edges incident at each vertex v of G. Equivalently, it is the clockwise ordering of the neighbours of each vertex. For example, Figure 2.6 gives the rotation system for a planar embedding of the graph K_4. For details see Mohar and Thomassen [MT01].

In order to understand the connection between these ideas and the notion of topology, we first note that the topology of a graph *drawing* is something entirely distinct from the topology of the *graph* itself. The latter simply refers to the graph's connectivity, or the question of which nodes are connected to which others. In the practical world this is the topology we are concerned with when considering different ways of connecting computers in a local area network, and in the theoretical world it is the sense of topology that was born in Euler's analysis of the Bridges of Königsberg problem (see Section 2.1), where all

that mattered was which land masses were connected to which others, and by how many bridges.

We mean something completely different when, following Batini et al. (see Section 2.4.2), we speak of the topology of a drawing of a graph. Here we begin by defining two planar embeddings $\varphi : G \to \wp(\mathbb{R}^2)$, $\varphi' : G \to \wp(\mathbb{R}^2)$ of a planar graph G to be *topologically equivalent* if there exists a homeomorphism $\psi : \mathbb{R}^2 \to \mathbb{R}^2$ of the plane carrying one embedding onto the other, i.e. such that $\bar{\psi} \circ \varphi = \varphi'$, where $\bar{\psi}$ is the map on $\wp(\mathbb{R}^2)$ induced by ψ. (See for example Munkres [Mun00] on basic point-set topology and the notion of homeomorphism or bi-continuous bijection.) Intuitively, if the plane were imagined as an infinitely stretchable rubber sheet, the two graph embeddings would be considered topologically equivalent if the sheet could be stretched in such a way as to turn one drawing into the other. Then by "the topology of" a given drawing (i.e. embedding) of a graph we mean the topological equivalence class of that drawing.

The connection between the topology of a drawing and its rotation system is enunciated in the *Heffter-Edmonds-Ringel rotation principle*, a key result in the theory of graphs on surfaces (see Mohar and Thomassen [MT01], page 91). This states that on an *orientable surface* (such as the 2-sphere S_2, i.e. the surface of the three-dimensional sphere) a drawing of a given graph G is determined up to homeomorphism by its rotation system. In other words, on a surface like S_2, two drawings with the same rotation system have the same topology, and if you change the rotation system you change the topology.

The plane $\mathbb{R}^2$ is not an orientable surface, but there is a close connection with the rotation principle since *locally* the 2-sphere looks like the plane. (Close to its surface, the Earth appears flat.) Mathematically, moving from S_2 to $\mathbb{R}^2$ means that we must augment the rotation system with one more bit of information: namely, which face of the embedding is the *external face*. Together, the rotation system and external face of a planar graph embedding completely determine its topology. See Figure 2.6.

2.4.2 A Refinement Approach to Layout

Topology-Shape-Metrics is what we will call a *refinement* approach to layout. For in TSM we imagine that we start with the set Ω of all possible drawings of the given graph G, and then this set is progressively refined until at last it contains only a single member, and that is the final drawing produced by the algorithm. The refinement takes place by repeatedly considering equivalence relations that partition the set of possible drawings into equivalence classes, picking one class, and discarding all the rest.

The first equivalence relation is the topological equivalence defined in the previous section, and accordingly the T- or Topology-phase of the TSM pipeline is concerned with choosing the topology of the drawing. Batini et al. referred to this first phase as *Planarisation*, since it involves planarising the given graph G (adding dummy nodes at edge intersections if the graph G is not planar to begin with), and choosing the rotation system and external face of the embedding.

The second equivalence relation in the TSM process determines what Batini et al. referred to as "shape". Throughout the first phase we were free to imagine graph drawings in which the connectors were drawn as arbitrary curves, but now, in the shape phase, they must be drawn as proper orthogonal connectors, i.e. piecewise-linear arcs in which each line segment is axis-aligned, or parallel to one of the x- and y-axes. Two drawings have the same *shape* when for each edge (u, v) in the graph their respective orthogonal drawings of this edge, traversed in the same direction (say from u toward v) have the same sequence of left and right bends. Formally, it is as though we wrote down for each edge a sequence (possibly empty) of L's and R's indicating the directions of the bends. The job of the *Orthogonalisation* phase is to choose the *shape* equivalence class of the drawing.

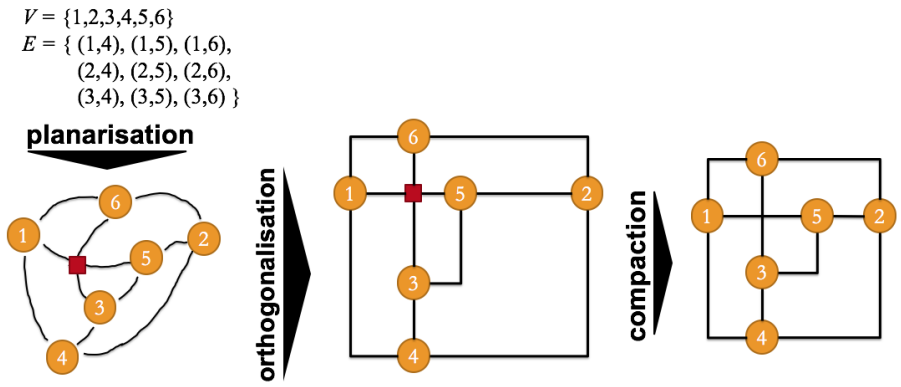


Figure 2.7: The TSM approach first determines the topology of the drawing in the planarisation step, then the shape of the drawing in the orthogonalisation step, and finally the metrics in the compaction step.

Finally the *metrics* of an orthogonal graph drawing in the sense of Batini et al. are determined when we choose the length of each linear segment of each connector. The third phase is called *Compaction* since the goal is to make these lengths as small as possible. However, Batini et al. decided that the drawing should land on a grid, so the segment lengths must be integer multiples of the grid size.

After the third phase the drawing is determined up to rigid transformations, but this was deemed trivial and not deserving of a proper phase of the algorithm. At this point each node, as well as each bend point of each connector, is assigned a position on the grid, and the drawing is complete.

Thus TSM gradually converges toward a drawing of the graph, by first determining the drawing's topology, then its shape, and finally its metrics. See Figure 2.7.

2.4.3 The Hierarchy of Aesthetics

As noted at the beginning of this section, the TSM approach is entirely concerned with aesthetics (and not at all with perceptual organisation), namely, with edge crossings, edge bends, and compactness. But Batini et al. realised that these aesthetics are not always mutually compatible, but may instead present competing goals, and none of them alone is sufficient.

Considering that the hierarchy of equivalence relations from Section 2.4.2 defined a natural way of converging gradually toward a graph drawing, Batini et al. argued that it was natural to handle the competing layout aesthetics in the corresponding order, that is, to deal first with edge crossings in the Topology phase, next with edge bends in the Shape phase, and finally with compactness in the Metrics phase. In this way the hierarchy of equivalence relations on drawings leads to a natural hierarchy of aesthetics.

Therefore in the initial planarisation phase the algorithm attempts to minimise the number of edge crossings.[13] Fixing the topology of the drawing from this point forward means that this number cannot change. The second phase, orthogonalisation, uses Tamassia's algorithm to actually achieve the minimum possible number of connector bends, given the topology. Finally, the compaction phase is concerned with minimising the overall area of the drawing.

Whether or not the hierarchy of aesthetics embodied in the TSM approach is "natural" as Batini et al. stated, it is important to note that this hierarchy enforces an ordering on aesthetic importance that was not justified empirically. In contrast, the experiment of Chapter 4 in this thesis allows us to gather some data on how people make trade-offs

[13]The techniques for this have been surveyed more recently by Gutwenger and Mutzel [GM03].

among the aesthetics. Moreover, since TSM is concerned only with aesthetics, and yet remains the state-of-the-art in orthogonal network layout today, this leaves an unmet need—namely, for an algorithm that produces good orthogonal node-link diagrams while taking perceptual organisation into account. This is the goal of the HOLA algorithm developed in Chapter 5.

2.4.4 Framework versus Algorithms

TSM was intended as a general graph drawing *framework*, whereas any algorithm that works according to this framework, i.e. by first determining the topology of the drawing, then its shape, and finally its metrics, may be called an *algorithm* of the TSM variety, or simply "a TSM algorithm". The original TSM algorithm of Batini et al. was built into a diagram layout system called GIOTTO [TDBB88], and several years later the algorithm itself began to be called GIOTTO [DBGL95].

Due to a limitation in Tamassia's bend minimisation technique, wherein it required a graph of maximum degree four,[14] the GIOTTO algorithm required a step called *expansion*, where each vertex was replaced by a "skeleton", a set of new vertices linked together into the shape of a box, each serving as a connection point to one of the original vertex's neighbours. The need for this complication was removed by Fössmeier and Kaufmann [FK95] in work from 1995 in which they augmented Tamassia's algorithm to handle graphs of arbitrary degree. In subsequent papers this new algorithm came to be known as Kandinsky.[15]

Further refinements and variants have been devised for the Kandinsky system over the years, some of them relevant to our concern with human-quality layout. The sketch-based system of Brandes et al. [BEKW02] works from a user-drawn sketch as a starting point, then proceeds to minimise bends using Kandinsky. This is one way to try to make more natural-looking orthogonal layouts, but it requires that the user provide a sketch to get the system started. Other efforts have for example gone into refining the initial planarisation phase, and exploring different options and configurations for it [GM03].

In Chapter 6 HOLA is compared with TSM, but it must be understood that in that and other chapters when we refer to specific TSM algorithms we mean the Kandinsky system, plus the subsequent techniques and refinements that have been added to it, such as those mentioned above. The whole suite of techniques is available both in OGDF (the Open Graph-Drawing Framework) [16] and in the yFiles layout library [WEK04], which is deployed in the popular yEd network layout editor. Together, these tools make the present state-of-the-art in orthogonal network layout.

2.5 Conclusions

The utility of the stress function in network layout was independently discovered, once from the side of multidimensional scaling, and again from the side of force-directed layout. The latter was in turn independently discovered, both motivated by data visualisation, and also by circuit board layout.

[14] Actually Tamassia's bend minimisation technique is capable of computing a bend-minimal *k-gonal* representation for any $k \geq 2$, where a k-gonal representation is one in which every connector segment has slope equal to an integer multiple of $180/k$ degrees. (Thus the orthogonal case is obtained with $k = 2$, and higher values of k allow for more angles.) However, for a k-gonal representation the technique is limited to $2k$-planar graphs, i.e. planar graphs all of whose nodes have degree bounded above by $2k$. Therefore in the important orthogonal case it is restricted to 4-planar graphs.

[15] The earliest occurrence of the name that I have found in print is in a paper published two years later by the same authors [FK97].

[16] `http://ogdf.net`

A stress-minimal layout is a *faithful* one, in that the geometric separation of nodes faithfully reflects their graph-theoretic separation. This was pointed out by Nguyen et al., and also emerges directly from the multidimensional scaling approach. If this "faithfulness" quality is something that people expect in a network layout (consciously or not), then we may expect stress-minimal layouts to be good starting points for layouts approximating human quality.

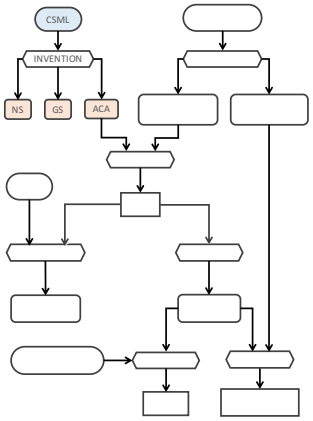
Furthermore, in his original paper on the subject Eades already suggested that a force-directed layout might only provide a "first approximation" to a layout, which could then be "fine tuned." The addition of constraints is a way in which we can perform such fine tuning, and we take that approach in this thesis. Constraints were first introduced into stress-minimised layout by Kamada and Kawai, eventually evolving to the system of Dwyer et al. with the use of separation constraints. It is this latter system that is employed in this thesis specifically, using the ADAPTAGRAMS layout library.

Another approach towards approximating human-quality layout was initiated by Marks et al. with their rule-based system. However their system relied on the user to deliberately select nodes and apply design directives to them. Moreover nodes were added a few at a time according to rules, with nothing to serve as a guide toward a good global layout. When rules for creating syntactic features could not be applied the system had to rely on arbitrary node placement. The DiAlEcT system of Chapter 7 improves on this by allowing node placements to be continually stress-guided, starting from a reasonable stress-based distribution. Marks et al. also did not consider specialised rule sets for different layout "styles" or dialects.

The state-of-the-art approach to the orthogonal layout style is currently the Topology-Shape-Metrics or TSM approach. However, being based entirely on optimisation of layout aesthetics, this approach completely ignores the need for perceptual organisation. It also has a hard-wired ordering on aesthetics, where human designers instead tend to make trade-offs, as we see in Chapter 4. Therefore there is room to improve on this system by creating layouts in the orthogonal style the way that people might create them when working by hand, as we do in Chapter 5 of this thesis, and evaluate in Chapter 6.

Chapter 3

Grid-like Layout with CSML



It was argued in the last chapter that stress minimisation should be a good starting point because of the "faithfulness" quality of this kind of layout. But as we consider how to get from this starting point to a diagram satisfying the conventions of a given layout language or dialect, it is clear that some further technique will be needed. See for example Figure 3.1, which shows the vast difference between stress-minimal and TSM layouts of the same graph. This chapter represents a first investigation into what the required technique might be.

Our goal is to begin developing methods that may help to achieve the layout requirements of various diagram languages and dialects, like orthogonal, SBGN, or even metro-map diagrams, to keep a broad view. We begin by considering orthogonal diagrams, whose requirements are simple and motivate the basic ideas that we build on in this chapter. Along the way we will discover that much of what we can do to make good-looking orthogonal layouts is in fact useful for many other diagram types as well. Thus, while orthogonal style will play the role of primary motivator, this chapter aims to take a broader view of layout style and aesthetics.

Thus we begin by considering the Orthogonal Language Family introduced in Chapter 1. The first fact we must confront is that although this family of diagrams is defined solely in terms of a condition on connector routes (i.e. that they be composed of alternating horizontal and vertical segments), the task of creating a good orthogonal layout nevertheless has everything to do with choosing good positions for the nodes. This fact is vividly demonstrated by what happens if we try to simply apply orthogonal connector routing directly to a pure stress-minimal layout, i.e. one in which stress has been minimised but no constraints have been applied. Using the Adaptagrams library, this means one

application of DESCEND followed immediately by one application of ORTHO-ROUTE (see Section 2.2.4). The typical results, demonstrated in Figure 3.2, are not pretty.

And in this case the prettiness of the layout, i.e. aesthetic quality, is exactly what we should be considering; for it will not be until the experiment of Chapter 4 that we learn something about the perceptual organisation (see Section 2.3.1) that users of orthogonal diagrams expect. The present chapter instead gives us a chance to address purely aesthetic concerns. Accordingly we get started in Section 3.1.2 by defining metrics for the aesthetic qualities that we aim to achieve, and we introduce the name "grid-like layout" for diagrams exhibiting these qualities. We consider that grid-like layout makes a suitable foundation for both orthogonal and SBGN layout.

After thus defining the problem we begin to investigate possible solutions. The basic question is: What new steps can be put in between the initial DESCEND and final ORTHO-ROUTE operations in order to get better results than the kind shown in Figure 3.2? Fundamentally, since CSML is a constrained optimisation system, there are two approaches we can take: we can either modify the goal function (stress) by adding new terms to make it behave differently, or we can add constraints. We may also use a combination of these two approaches.

In Section 3.2 we consider the approach of modifying the stress function, and in Section 3.3 we consider a pure constraint-driven approach that leaves the stress function unmodified but instead automatically chooses and applies constraints. Section 3.4 evaluates, while Section 3.5 concludes that the constraint-driven approach is best. A preview of the various techniques is shown in Figure 3.3. The work described in this chapter was originally published in a different format, in the proceedings of the Graph Drawing conference in 2013 [KDMW13].

3.1 Grid-like Layout

The generally accepted set of graph layout aesthetics [DBETT99] including goals like minimising edge crossings and bends tells us what is so bad about the layout in Figure 3.2. To begin with, it has far too many bends. The crossings do not help either. Moreover, while I am not aware of any published research to substantiate the claim, it seems likely that the dissatisfaction with the layout of Figure 3.2 is amplified by the obvious ease with which the excessive bends could be removed. In other words, one not only finds the layout objectionable, but wonders: Why would anyone leave it that way, when it could so easily be improved?

The tantalisingly easy improvement of the diagram in Figure 3.2 could be achieved by creating *alignments*. Aligning nodes that are nearly but not quite aligned would immediately remove many of the small, squiggly S-bends that clutter up this diagram. The many L-bends at the extremities of the layout, whose presence seems causeless if we forget about the initial stress-minimisation step, could also be removed by creating alignments.

Pursuing this line of thought we are led to revisit the decision of Batini et al. to put orthogonal layouts on a grid (Section 2.4). The wisdom of that choice becomes apparent as we reflect that, on a grid, nodes are never almost-but-not-quite aligned; they are either perfectly aligned, or else far enough out of alignment that their disalignment appears deliberate. In fact grid placement means two things: *alignments* and *commensurate separations*. In other words, many nodes are aligned, while those that are not aligned are separated by a distance that is an integer multiple of a single, fixed grid size.

However, while the aesthetic deficiencies of Figure 3.2 clearly call for alignments, it is not as clear that commensurate separations are really necessary. For this reason we aim to achieve not grid layout, but what we call *grid-like layout*. The full definition is given in Section 3.1.2.

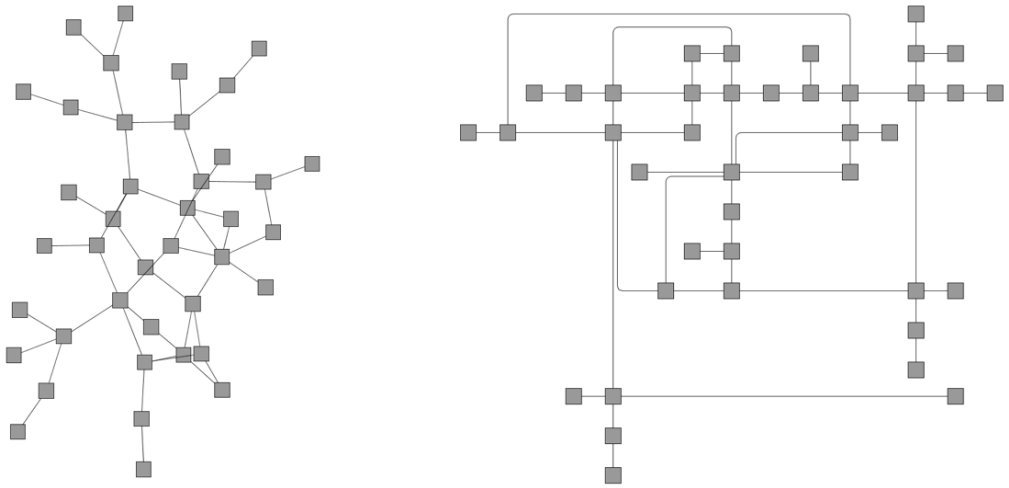


Figure 3.1: Stress-minimal (left) and TSM (right) layouts of the same graph, appear quite different.

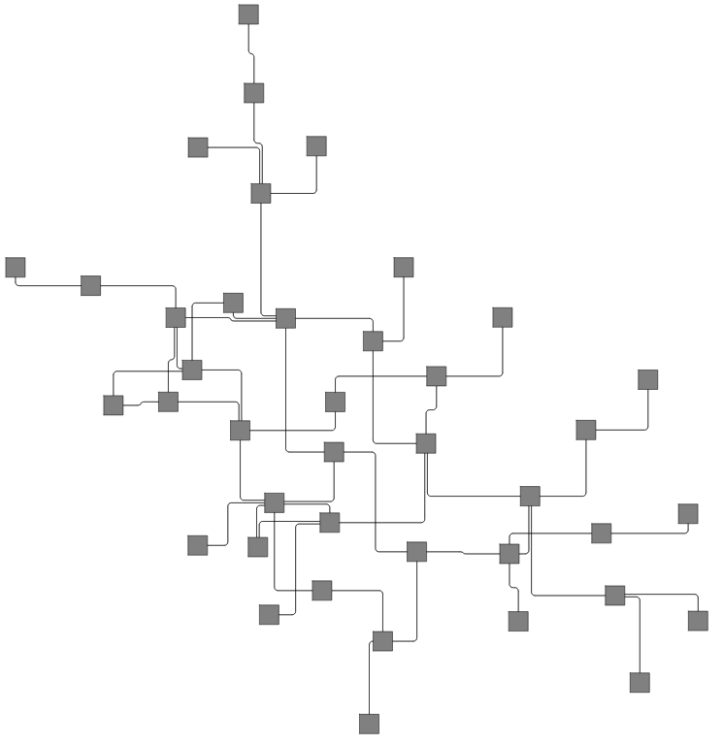
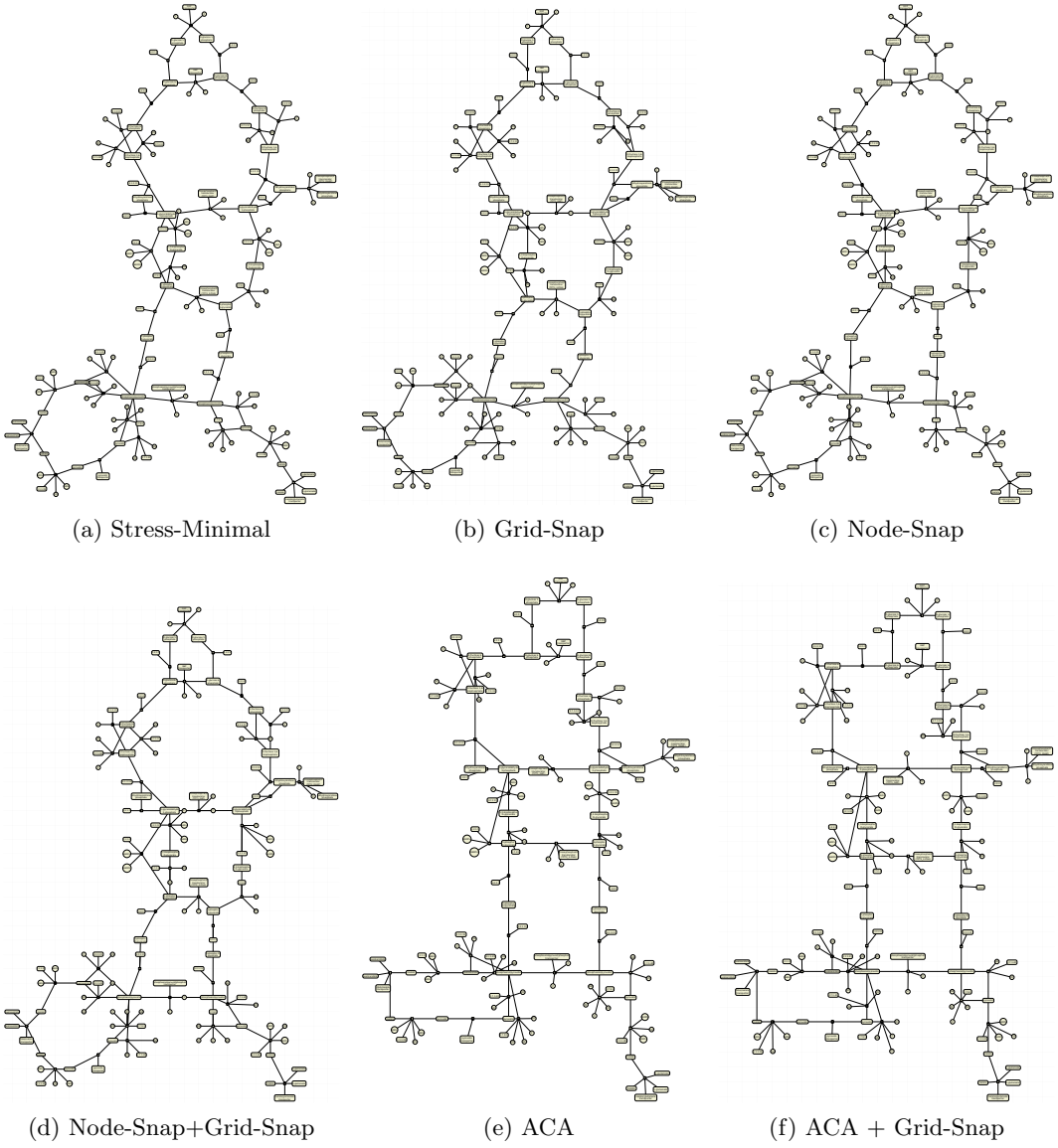


Figure 3.2: Stress-minimised layout with orthogonal routing applied directly tends to produce results with bad aesthetic value.



(a) Stress-Minimal (b) Grid-Snap (c) Node-Snap

(d) Node-Snap+Grid-Snap (e) ACA (f) ACA + Grid-Snap

Figure 3.3: Different combinations of our grid-like layout techniques are shown, compared with pure stress-minimal layout. The layout is for an SBGN diagram of the Glycolysis-Gluconeogenesis pathway obtained from MetaCrop.

In fact, a layout in which many nodes are aligned seems like a reasonable starting point not just for orthogonal layout, but for SBGN layout too, since long chains of alignments are commonly observed in SBGN diagrams in practice. Still, wary of the dangers of overfitting the solution at this early stage, it is important to keep yet other layout styles in mind. In order to broaden the approach we therefore consider *metro-map* (train network) layout too.

Before coming to our definition of grid-like layout in Section 3.1.2 we pause to review related work on grid layout, SBGN layout, and metro map layout in Section 3.1.1.

3.1.1 Related Work

Several proposals have been made for grid-like layout of biological networks [BGHM07, LK05, KNJ+07]. These arrange biological networks with grid coordinates for nodes in addition to various layout constraints. In particular Barsky *et al.* [BGHM07] consider alignment constraints between biologically similar nodes and Kojima *et al.* [KNJ+07] perform layout subject to rectangular containers around functionally significant groups of nodes (e.g. metabolites inside the nucleus of a cell). In general they use fairly straightforward simulated annealing or simple incremental local-search strategies. Such methods work to a degree but are slow and may never reach a particularly aesthetically appealing minimum.

Another application where grid-like layout is an important aesthetic is metro-map layout. Stott *et al.* [SRMOW11] use a simple local-search ("hill-climbing") technique to obtain layout on grid points subject to a number of constraints, such as "octilinear" edge orientation.[1] Wang and Chi [WC11] seek similar layout aesthetics but using continuous non-linear optimisation subject to octilinearity and planarity constraints. This work is based on a quasi-Newton optimisation method (like the one used in Adaptagrams), but it is very specific to metro-map layout. Metro-map layout algorithms such as [NW11a] run for many hours before finding a solution.

There are several existing examples of the application of altered force laws to layout, like we consider in Section 3.2. Sugiyama and Misue [SM95] augment the standard force-model with "magnetic" edge-alignment forces. Ryall *et al.* [RMS97] explored the use of various force-based constraints in the context of an interactive diagramming editor. It is the limitations of such approaches (discussed below) which prompt the development of the constraint-based techniques described in Section 3.3.

3.1.2 Aesthetic Criteria

We assume that we have a graph $G = (V, E, w, h)$ consisting of a set of nodes V, a set of edges $E \subseteq V \times V$ and w_v, h_v are the width and height of node $v \in V$. We wish to find a straight-line 2D drawing for G. This is specified by a pair (x, y) where (x_v, y_v) is the centre point of each $v \in V$.

We quantify grid-like layout quality through the following metrics. In subsequent sections we use these to develop techniques that directly or indirectly aim to optimise them. We use these metrics for evaluation in Section 3.4 as well.

- **Embedding quality:** We measure this using the *P-stress* function [DMW09b], a variant of stress that does not penalise unconnected nodes being more than their desired distance apart. It measures the separation between each pair of nodes $u, v \in V$ in the drawing and their *ideal distance* d_{uv} proportional to the graph theoretic

[1] Octilinear edge routing is the same as 4-gonal representation in Tamassia's terms. See the footnote on page 28.

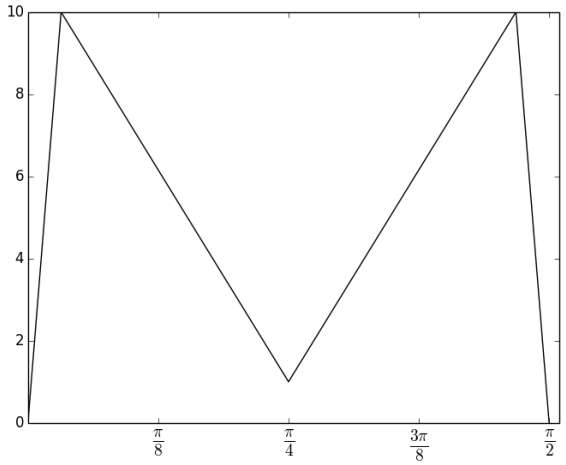


Figure 3.4: M-shaped function. Note that $[0, \pi/2]$ is the range of $|tan^{-1}|$. The "M" function is zero at 0 and $\pi/2$, a small value $p \geq 0$ at $\pi/4$, a large value $P > 0$ at δ and $\pi/2 - \delta$ for some small $\delta > 0$, and linear in-between.

path between them:

$$\sum_{u<v \in V} w_{uv} \left((d_{uv} - d(u,v))^+ \right)^2 + \sum_{(u,v) \in E} wp \left((d(u,v) - d_L)^+ \right)^2$$

where $d(u,v)$ is the Euclidean distance between u and v, $(z)^+ = z$ if $z \geq 0$ otherwise 0, d_L is an ideal edge length, $wp = \frac{1}{d_L}$, and $w_{uv} = \frac{1}{d_{uv}^2}$.

- **Edge crossings:** The number of edge crossings in the drawing.
- **Edge/node overlap:** The number of edges intersecting a node box. With straight-line edges this also penalises coincident edges.[2]
- **Angular resolution:** Edges incident on the same node have a uniform angular separation. Stott *et al.* [SRMOW11] give a useful formulation:

$$\sum_{v \in V} \sum_{\{e_1, e_2\} \in E_v} |2\pi / degree(v) - \theta(e_1, e_2)|$$

- **Edge obliqueness:** We prefer horizontal or vertical edges and then—with weaker preference—edges at a 45° orientation. Our precise metric is $M \left| \tan^{-1} \frac{y_u - y_v}{x_u - x_v} \right|$ where $M(\theta)$ is an "M-shaped function" over $[0, \pi/2]$ that highly penalises edges which are almost but not quite axis-aligned and gives a lower penalty for edges midway between horizontal and vertical (Figure 3.4). Other functions like those of [SRMOW11, KNJ+07] could be used instead.
- **Grid placement:** Average of distances of nodes from their closest grid point.

3.2 Goal Function Methods

The CSML framework presents two obvious ways to customise the layout. For one, we can simply use its capacity to handle constraints and apply these to achieve the kind of layout we want. We explore this approach in Section 3.3. For another, we can recall that

[2]Node/node overlaps are also undesirable. We avoid them completely by using non-overlap constraints [DMS06] in all our tests and examples.

CSML is just a case of constrained optimisation, with stress as the goal function, and we can think about ways to modify that goal function to try and achieve the desired layout.

Such an approach will necessarily only *promote* certain kinds of layout, and will not *guarantee* them, as constraints will. This has the advantage of allowing various desiderata to compete, and is simple to implement. Note that, while we dismissed the goal-function-based approaches of Marks et al. in Section 2.3.2, that was because they led to imprecision in *perceptual organisation*. By contrast, in the present chapter we are still concerned only with *aesthetics*, where we deem imprecision to be much more acceptable.

In this section we describe two new terms that can be combined with the *P-stress* function to achieve more grid-like layout: *NS-stress* for "node-snap stress" and *GS-stress* for "grid-snap stress." An additional term *EN-sep* gives good separation between nodes and edges. Layout is then achieved by minimising

$$\text{P-stress} + k_{ns} \cdot \text{NS-stress} + k_{gs} \cdot \text{GS-stress} + k_{en} \cdot \text{EN-sep}$$

where the constants k_{ns}, k_{gs}, and k_{en} control the "strength" of the various components. These extra terms, as defined below, tend to make nodes lie on top of one another. It is essential to avoid this by solving subject to node-overlap prevention constraints, as described in [DMS06]. To obtain an initial "untangled" layout we run with $k_{ns} = k_{gs} = k_{en} = 0$ and without non-overlap constraints (Fig. 3.3a). We then run again with the extra terms and constraints to perform "grid-like beautification".

Minimisation of the *NS-stress* term favours horizontal or vertical alignment of pairs of connected nodes (Figs. 3.3c and 3.9). Specifically, taking σ as the distance at which nodes should snap into alignment with one another, we define:

$$\text{NS-stress} = \sum_{(u,v)\in E} q_\sigma(x_u - x_v) + q_\sigma(y_u - y_v) \quad \text{where } q_\sigma(z) = \begin{cases} z^2/\sigma^2 & |z| \le \sigma \\ 0 & \text{otherwise.} \end{cases}$$

The discontinuity of $q_\sigma(z)$ at $\pm\sigma$ may seem a strange choice, but it turns out that the function must at least be non-differentiable there (if not necessarily discontinuous). This is because we want the function to attain its globally maximal value (over all of $\mathbb{R}$) at $\pm\sigma$, and so if it is also smooth at those points then it will be concave-down somewhere over the interval $[-\sigma, \sigma]$ and this causes a problem. Specifically, downward concavity causes the standard step-size calculations on which the Adaptagrams gradient-projection algorithm is based to make steps that increase the goal function rather than decreasing it. Therefore more obvious choices like an inverted quartic $(1 + (z^2 - \sigma^2)^2)^{-1}$ or a sum of inverted quadratics $(1 + (z + \sigma)^2)^{-1} + (1 + (z - \sigma)^2)^{-1}$ in place of $q_\sigma(z)$ simply won't work. We examine the step size, gradient, and Hessian formulae for our snap-stress functions in Appendix A.

We designed our *GS-stress* function likewise to make the lines of a virtual grid exert a similar attractive force on nodes once within the snap distance σ:

$$\text{GS-stress} = \sum_{u\in V} q_\sigma(x_u - a_u) + q_\sigma(y_u - b_u)$$

where (a_u, b_u) is the closest grid point to (x_u, y_u) (with ties broken by favouring the point closer to the origin), see Figure 3.3b. The grid is defined to be the set of all points $(n\tau, m\tau)$, where n and m are integers, and τ is the "grid size". With *GS-stress* active it is important to set some other parameters proportional to τ. First, we take $\sigma = \tau/2$. Next, we modify the non-overlap constraints to allow no more than one node centre to be in the vicinity of any one grid point by increasing the minimum separation distance allowed between adjacent nodes to τ. Finally, we found that setting the ideal edge length equal

to τ for initial force-directed layout, before activating *GS-stress*, helped to put nodes in positions compatible with the grid.

Our third term *EN-sep* is also a quadratic function based on $q_\sigma(z)$ that separates nodes and nearby axis-aligned edges to avoid node/edge overlaps and coincident edges:

$$\text{EN-sep} = \sum_{e \in E_V \cup E_H} \sum_{u \in V} q_\sigma \left((\sigma - d(u,e))^+ \right),$$

where E_V and E_H are the sets of vertically and horizontally aligned edges, respectively, and the distance $d(u,e)$ between a node u and an edge e is defined as the length of the normal from u to e if that exists, or $+\infty$ if it does not. Here again we took $\sigma = \tau/2$.

In our experiments we refer to various combinations of these terms and constraints, as follows:

- **Node-Snap:** *NS-stress*, *EN-sep*, non-overlap constraints, $k_{gs} = 0$
- **Grid-Snap:** *GS-stress*, *EN-sep*, ideal edge lengths equal to grid size, non-overlap, constraints with separations tailored to grid size, $k_{ns} = 0$.
- **Node-Snap+Grid-Snap:** achieves extra alignment by adding *NS-stress* to the above **Grid-Snap** recipe (i.e. $k_{ns} \neq 0$).

3.3 Adaptive Constrained Alignment

In this section we describe how to make stress-minimised layouts more grid-like simply by adding alignment and separation constraints (Fig. 3.3e).

The algorithm, which we call *Adaptive Constrained Alignment or ACA*, is a greedy algorithm that repeatedly chooses an edge in G and aligns it horizontally or vertically (see *adapt_const_align* procedure of Figure 3.5). It *adapts* to user-specified constraints by not adding alignments that violate these. The algorithm halts when the heuristic can no longer apply alignments without creating edge overlaps. Since each edge is aligned at most once, there are at most $|E|$ iterations.

We tried the algorithm with three different heuristics for choosing potential alignments, which we discuss below.

Node overlaps and edge/node overlaps can be prevented with non-overlap constraints and the *EN-sep* goal-function-term discussed in Section 3.2, applied either before or after the ACA process. However, coincident edges can be accidentally created and then enforced as we apply alignments if we do not take care to maintain the orthogonal ordering of nodes. If for example two edges (u,v) and (v,w) sharing a common endpoint v are both horizontally aligned, then we must maintain either the ordering $x_u < x_v < x_w$ or the opposite ordering $x_w < x_v < x_u$. [3]

Therefore we define the notion of a *separated alignment*, written $\mathsf{SA}(u,v,D)$ where $u,v \in V$ and $D \in \{\mathbb{N},\mathbb{S},\mathbb{W},\mathbb{E}\}$ is a compass direction. Applying a separated alignment means applying two constraints to the force-directed layout—one alignment and one separation—as follows:

$$\begin{array}{rcl} \mathsf{SA}(u,v,\mathbb{N}) \equiv & x_u = x_v \quad \text{and} \quad y_v + \beta(u,v) \leq y_u, & \qquad \mathsf{SA}(u,v,\mathbb{S}) \equiv \mathsf{SA}(v,u,\mathbb{N}), \\ \mathsf{SA}(u,v,\mathbb{W}) \equiv & y_u = y_v \quad \text{and} \quad x_v + \alpha(u,v) \leq x_u, & \qquad \mathsf{SA}(u,v,\mathbb{E}) \equiv \mathsf{SA}(v,u,\mathbb{W}), \end{array}$$

where $\alpha(u,v) = (w_u + w_v)/2$ and $\beta(u,v) = (h_u + h_v)/2$. (Thus for example $\mathsf{SA}(u,v,\mathbb{N})$ can be read as, "the ray from u through v points north," where we think of v as lying north of u when its y-coordinate is smaller.)

[3] In a diagram system in which *edge bundling* [Hol06] was used this restriction could be relaxed, but we do not pursue that approach here.

```
proc adapt_const_align(G, C, K)
  (x, y) ← DESCEND(G, C)
  SA ← chooseSA(G, C, x, y, K)
  while SA != NULL
    C.append(SA)
    (x, y) ← DESCEND(G, C)
    SA ← chooseSA(G, C, x, y, K)
  return (x, y, C)
```

```
proc chooseSA(G, C, x, y, K)
  S ← NULL
  cost ← ∞
  for each (u, v) ∈ E and dir. D
    if not creates_coincidence(C, x, y, u, v, D)
      if K(u, v, D) < cost
        S ← SA(u, v, D)
        cost ← K(u, v, D)
  return S
```

Figure 3.5: Adaptive constrained alignment algorithm. G is the given graph, C the set of user-defined constraints, and K the cost function.

Alignment Choice Heuristics.

We describe two kinds of alignment choice heuristics: *generic*, which can be applied to any graph, and *convention-based*, which are intended for use with layouts that conform to special conventions, e.g. those of SBGN diagrams. Our heuristics are designed according to two principles:

1. try to retain the overall shape of the initial stress-minimal layout;
2. do not obscure the graph structure by creating undesirable overlaps

and differ only in the choice of a *cost function* K which is plugged into the procedure `chooseSA` in Figure 3.5. This relies on procedure `creates_coincidence` which implements the **Edge Coincidence Test** described below. Among separated alignments which would not lead to an edge coincidence, `chooseSA` selects one of lowest cost. Cost functions may return a special value of ∞ to mark an alignment as never to be chosen.

The `creates_coincidence` procedure works by maintaining a $|V|$-by-$|V|$ array of flags which indicate for each pair of nodes u, v whether they are aligned in either dimension and whether there is an edge between them. The cost of initialising the array is $\mathcal{O}(|V|^2 + |E| + |C|)$, but this is done only once in ACA. Each time a new alignment constraint is added the flags are updated in $\mathcal{O}(|V|)$ time, due to transitivity of the alignment relation. Checking whether a proposed separated alignment would create an edge coincidence also takes $\mathcal{O}(|V|)$ time, and works according to the **ECT**. (Proof is provided in Appendix B.) Note that the validity of the **ECT** relies on the fact that we apply separated alignments $\mathsf{SA}(u, v, D)$ only when (u, v) is an edge in the graph.

Edge Coincidence Test: *Let G be a graph with separated alignments. Let u, v be nodes in G which are not yet constrained to one another. Then the separated alignment $\mathsf{SA}(u, v, \mathbb{E})$ creates an edge coincidence in G if and only if there is a node w which is horizontally aligned with either u or v and satisfies either of the following two conditions: (i) $(u, w) \in E$ while $x_u < x_w$ or $x_v < x_w$; or (ii) $(w, v) \in E$ while $x_w < x_v$ or $x_w < x_u$. The case of vertical alignments is similar.*

We tried various cost functions, which addressed the aesthetic criteria of Section 3.1.2 in different ways. We began with a *basic cost*, which was either an estimate $K_{dS}(u, v, D)$ of the change in the stress function after applying the proposed alignment $\mathsf{SA}(u, v, D)$, or else the negation of the obliqueness of the edge, $K_{ob}(u, v, D) = -obliqueness((u, v))$, as measured by the function of Section 3.1.2. In this way we could choose to address the aesthetic criteria of *embedding quality* or *edge obliqueness*, and we found that the results were similar. Both rules favour placing the first alignments on edges which are almost axis-aligned, and this satisfies our first principle of being guided as much as possible by the shape of the initial force-directed layout. See for example Figure 3.3.

On top of this basic cost we considered *angular resolution* of degree-2 nodes by adding a large but finite cost that would postpone certain alignments until after others had been

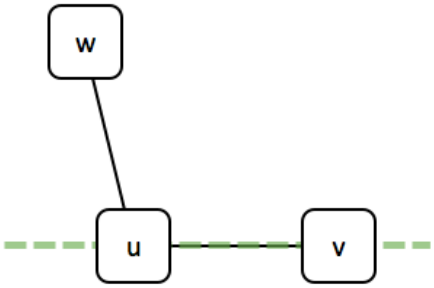


Figure 3.6: Suppose nodes u and v are horizontally aligned. When the ACA process considers the edge (u, w) the basic angular cost will suggest that the vertical alignment of u and w would be best; however, since u has degree 2 we add a special penalty cost to that alignment. The penalty makes the horizontal alignment of u and w more attractive, and this promotes the creation of long, straight chains of aligned nodes.

attempted. This took the form of a fixed cost ten times larger than average values of the cost functions K_{dS} and K_{ob}, added for any alignment that would make a degree-2 node into a "bend node," i.e., would make one of its edges horizontally aligned while the other was vertically aligned. See Figure 3.6. This allows long chains of degree-2 nodes to form straight lines, and cycles of degree-2 nodes to form perfect rectangles.

For SBGN diagrams we get good results by applying the same technique, except that when it comes to locating the "degree-2" nodes we use *non-leaf* degree. This means the degree of a node when we discount all of its neighbours that are leaves. Combining this with another large cost penalty for aligning leaf edges leads to layouts like those in Figures 3.3e and (f). Since leaves have little to do with the overall structure of the graph, their edges can be left unaligned. Instead, the nodes having more to do with the structure wind up getting aligned into long, straight chains.

Respecting User-Defined Constraints.

Layout constraints can easily wind up in conflict with one another if not chosen carefully. In Adaptagrams such conflicts are detected during the PROJECT operation, an *active set* method which iteratively determines the most violated constraint c and satisfies it by minimal disturbance of the node positions. When it is impossible to satisfy c without violating one of the constraints that is already in the active set, c is simply marked *unsatisfiable*, and the operation carries on without it.

For ACA it is important that user-defined constraints are never marked unsatisfiable in deference to an alignment imposed by the process; therefore we term the former *definite* constraints and the latter *tentative* constraints. We employ a modified projection operation which always chooses to mark one or more tentative constraints as unsatisfiable if they are involved in a conflict.

For conflicts involving more than one tentative constraint, we use Lagrange multipliers to choose which one to reject. These are computed as a part of the projection process. Since alignment constraints are equalities (not inequalities) the sign of their Lagrange multiplier does not matter, and a constraint whose Lagrange multiplier is maximal in absolute value is one whose rejection should permit the greatest decrease in the stress function. Therefore we choose this one.

ACA does not snap nodes to grid-points: if desired this can be achieved once ACA has added the alignment constraints by activating Grid-Snap.

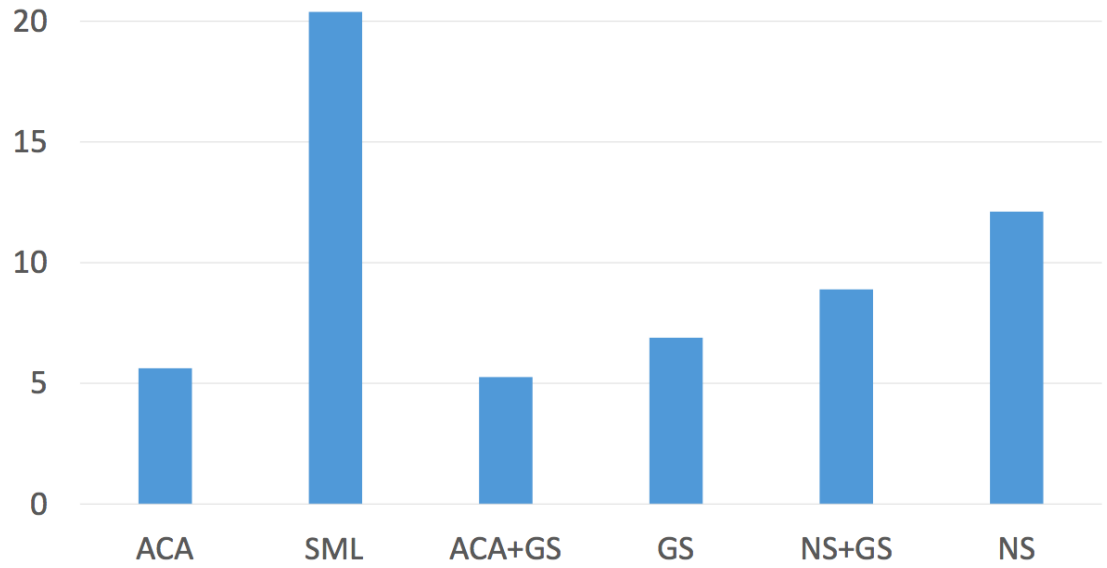


Figure 3.7: Edge obliqueness (see Section 3.1.2) results. The constraint-based approach ACA is better than either of the goal-function-based approaches Grid-Snap (GS) and Node-Snap (NS). The combination of ACA and GS gives the best result.

3.4 Evaluation

To evaluate the various techniques we applied each to 252 graphs from the "AT&T Graphs" corpus (`ftp://ftp.research.att.com/dist/drawdag/ug.gz`) with between 10 and 244 nodes. We excluded graphs with fewer than 10 nodes and two outlier graphs: one with 1103 nodes and one with 0 edges. We recorded running times of each stage in the automated batch process and the various aesthetic metrics described in Section 3.1.2, using a MacBook Pro with a 2.3GHz Intel Core I7 CPU. Examples are visible in Figures 3.12 and 3.13. In these and other figures in this section we refer to: unconstrained (except for non-overlap constraints) stress-minimising layout as SML, Grid-Snap as GS, Node-Snap as NS, and Adaptive Constrained Alignment as ACA.

We found that ACA was the slowest, often taking up to 10 times as long as the other methods, on average around 5 seconds for graphs with around 100 nodes, while the other approaches took around a second. See Figure 3.10. ACA was also sensitive to the density of edges. Of the goal-function approaches, Grid-Snap (being very local) added very little time over the unconstrained force-directed approach.

The Edge Obliqueness results are shown in Figure 3.7. ACA does the best job here, with its performance improved slightly if GS is added. On the other hand, use of GS increases the stress of the layout dramatically (Figure 3.8), whereas both ACA and NS keep stress nearly as low as pure SML. It is natural that stress has to be increased a little as we enforce alignment constraints. Comparing performance on both edge obliqueness and stress shows ACA to be the clear winner. ACA also shows the best performance on angular resolution (Figure 3.11), and does a good job of aligning long paths of nodes, as is visible in Figures 3.3 and 3.9.

3.5 Conclusions

We identified a set of layout aesthetics we call "grid-like" layout, to address the basic aesthetic requirements of many types of node-link diagram, including orthogonal and SBGN.

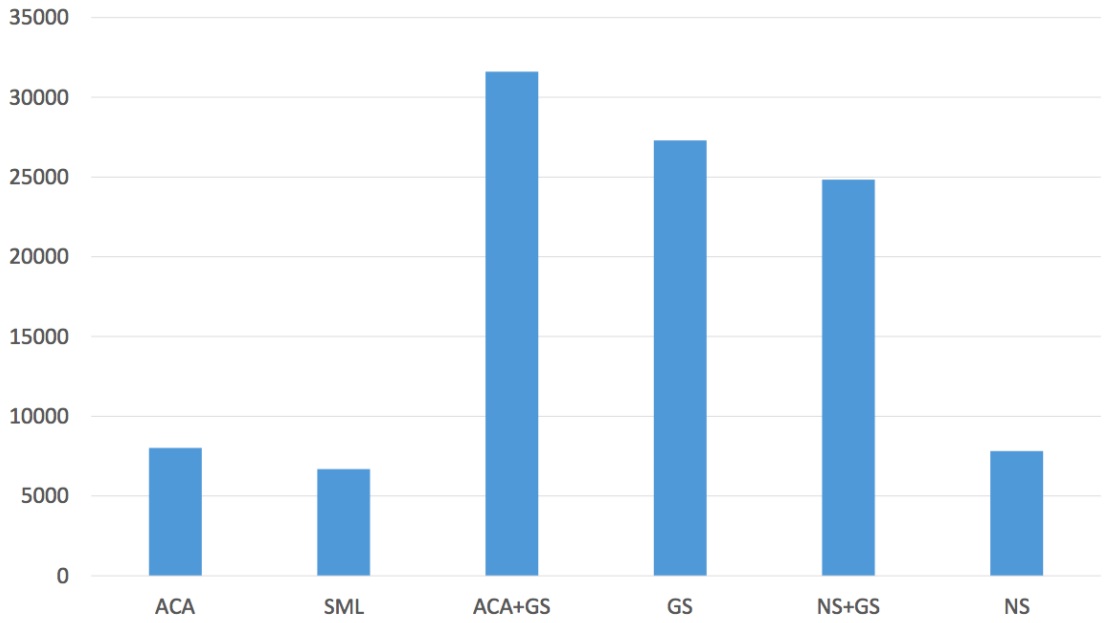


Figure 3.8: P-Stress values normalised by graph size and density. There is not much to note except that GS introduces the most significant stress. Basically, this means that optimisation over *GS-stress* introduces the most distortion of the underlying SML layout.

We explored ways of achieving grid-like layout with CSML, including both goal-function-based approaches (Node-Snap, Grid-Snap) and an adaptive constraint-based approach (ACA) in which alignment constraints are added greedily. ACA is slower but gives more grid-like layout and so is to be preferred.

One of the lessons learned is that a good layout needs a moderate amount of stress. A high-stress layout is not faithful to the structure of the network. On the other hand, if we simply minimise stress (like in Figure 3.2) then we have no hope of satisfying the expectations of a special layout language or dialect. In this chapter we have seen how we can improve the aesthetics by adding constraints and thereby adding a bit of stress. In Chapters 5 and 8 we will see how the same technique also allows us to create good perceptual organisation.

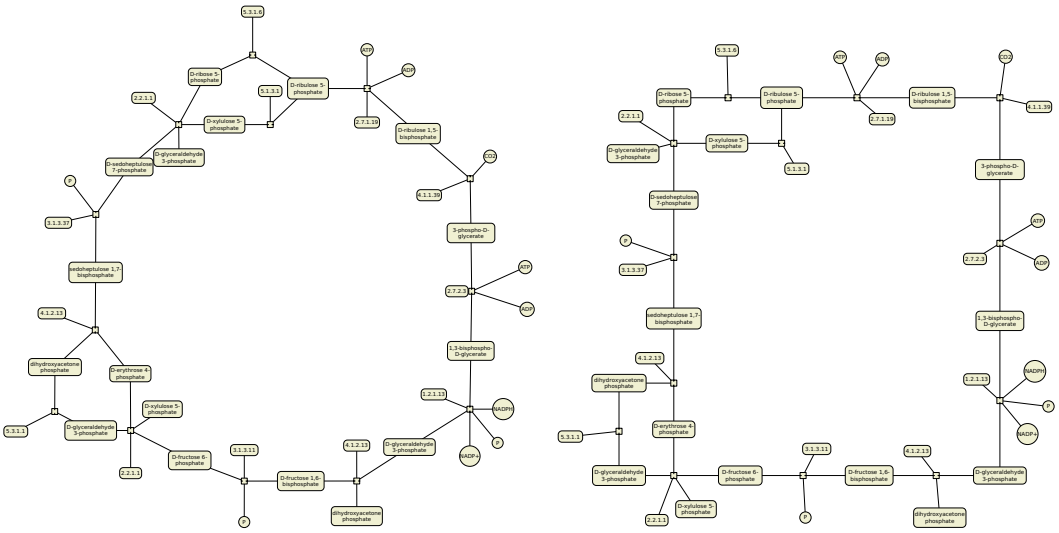


Figure 3.9: Layout of a SBGN diagram of Calvin Cycle pathway shows how ACA (right) gives a more pleasing rectangular layout than Node-Snap (left).

Figure 3.10: Running time in seconds for the six different grid-like layout methods against number of nodes for the 252 graphs in our corpus. Times given do not include the other layout stages. For example, ACA does not include the initial SML layout. ACA+GS, is just the additional grid stage after SML+ACA.

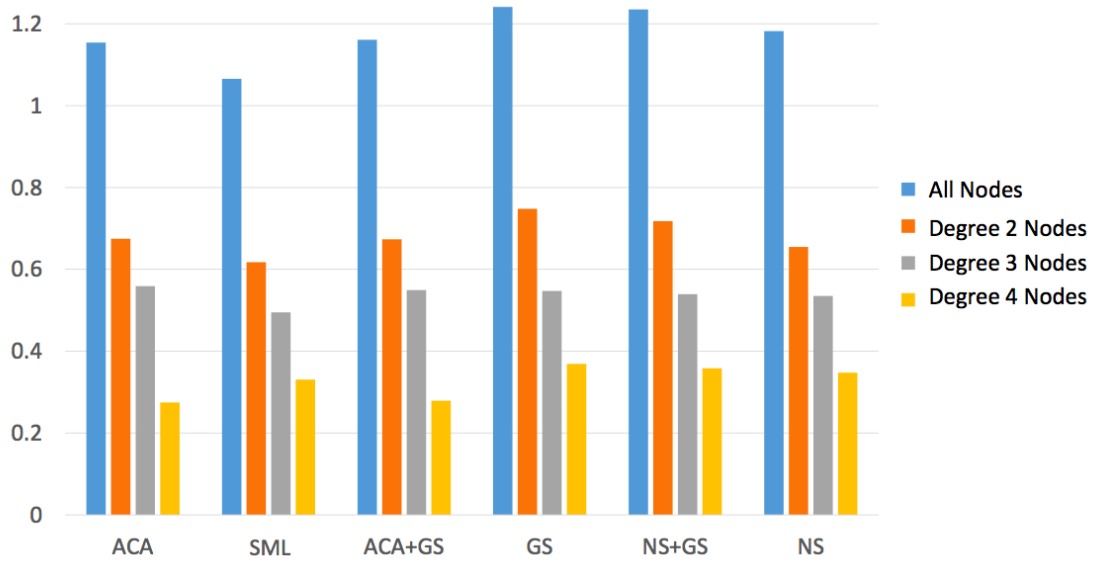
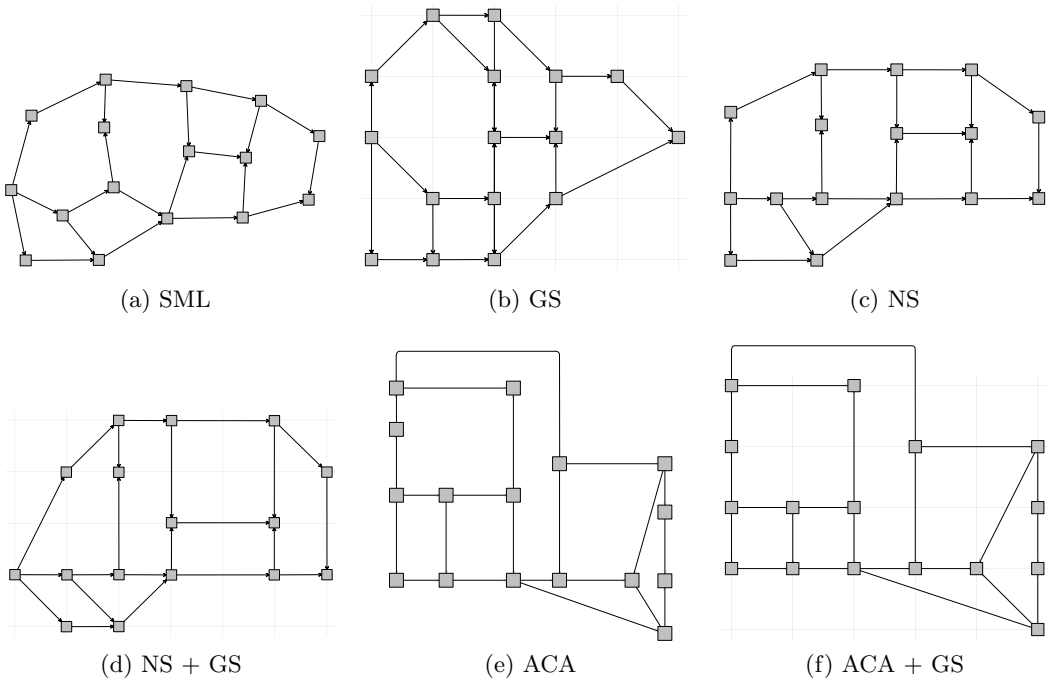


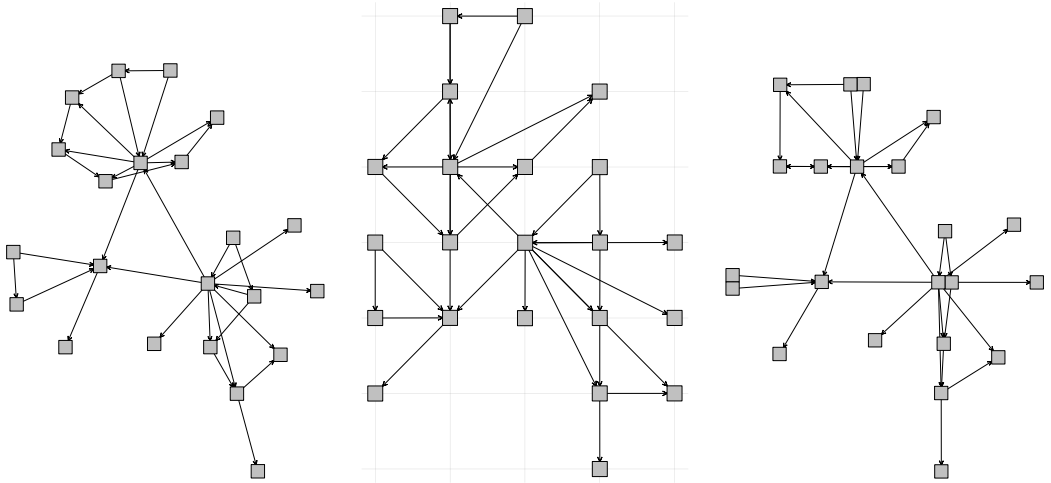
Figure 3.11: Angular resolution for the various techniques for all nodes, but also broken down for lower degree nodes. We see ACA does almost as well as SML on degree-2 nodes, and results in better angular resolution than SML for degree-4.



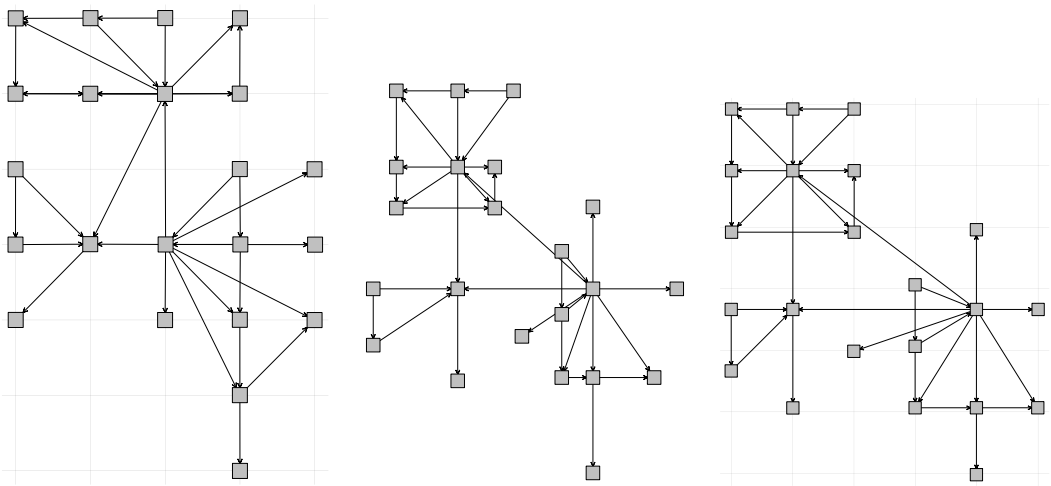
(a) SML (b) GS (c) NS

(d) NS + GS (e) ACA (f) ACA + GS

Figure 3.12: Different combinations of our automatic layout techniques for the graph "ug_213" from the AT&T Graphs corpus, as generated during our evaluation. In 3.12e and 3.12f we use a simple post-process to see if edges involved in crossings can be rerouted to avoid crossings using the orthogonal connector routing scheme described in [WMS10].



(a) SML (b) GS (c) NS

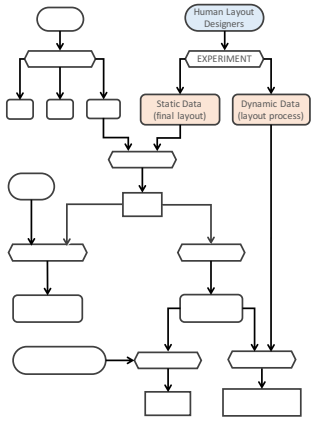


(d) NS + GS (e) ACA (f) ACA + GS

Figure 3.13: Different combinations of our automatic layout techniques for the graph "ug_268" from the AT&T Graphs corpus, as generated during our evaluation.

Chapter 4

Investigating Manual Layout



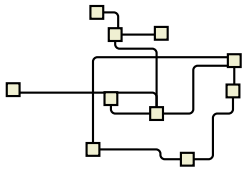
The study of Chapter 3 showed us that we can use CSML to create grid-like layout aesthetics, a promising start toward our goal of teaching the computer to "speak" various layout "dialects". However, in order to reach that goal we must pick a particular type of diagram and conduct a user study to find out how real people actually lay out diagrams of that kind in practice. In terms of the three-part methodology described at the end of Section 1.1, this is the formative user study.

The Orthogonal Language Family is a natural starting point. It is highly general and broadly useful, and its aesthetic demands should be particularly well served by the many alignments created by the ACA procedure of Chapter 3. It will be our focus in this and the next two chapters. Much of the material in these three chapters originally appeared in a different format in [KDMW16], though significant additions have been made.

This chapter describes a user study called *Orthowontist*, a pun referring to the layout designer's *wont* in creating *ortho*(gonal) diagrams. Section 4.1 reviews prior studies investigating manual layout. Our experiment is then described in Section 4.2 and the results in Sections 4.3 and 4.4. The results teach us about the aesthetics that our study participants were concerned with, and how they traded these off against each other (Section 4.3), as well as about the kind of perceptual organisations the participants applied to the nodes (Section 4.4).

4.1 Prior Studies

Historically, formal user studies of the requirements for good layout only occurred *after* the initial development of automatic layout methods. The earliest user studies that we know of were not until the mid 1990's. See Section 4.1.1. These used carefully designed



diagrams to investigate whether the aesthetic criteria intuitively identified by the early algorithm designers did in fact assist with comprehension, e.g. [PCJ97, MBK97]. At the time this was taken very much as an affirmation of the existing algorithms since these studies largely confirmed that the aesthetic criteria identified by the algorithm designers, such as reducing crossings and bend points, did affect readability.

It was not until quite recently that studies such as [vHR08, DLF+09, PPP12] returned to first-principles in understanding the requirements for network layout by investigating the kind of node-link diagrams that humans construct by hand. See Section 4.1.2. Such "human authored layout" studies began to identify aesthetic factors not considered by early algorithm designers or in the previous usability experiments. Yet until now such studies have not influenced the design of automatic layout algorithms. This is not really surprising, as creating new algorithms or re-engineering existing algorithms to capture new aesthetics is difficult and time consuming.

4.1.1 Early Aesthetic Studies

The first network layout user studies investigated whether the intuitive aesthetic criteria used to motivate the design of graph drawing algorithms did in fact affect human understanding of graphs. Early studies by Purchase and her colleagues [PCJ96, Pur97, PCJ97, PCA02] investigated how reducing edge crossings, reducing bends, showing subgraph symmetry, increasing angle of incidence of edges entering/leaving a node, and orthogonality (which was taken to mean node placement on a grid) affected user performance on tasks like finding the shortest path between two nodes in abstract graphs. Another study by Purchase *et al.* [PAC02] investigated the effect of these aesthetics on user preference in UML diagrams while Huang *et al.* [HHE07] investigated their effect on both preference and performance for social networks.

These studies reported strong negative effects on task performance and user preference for edge crossings and to a lesser extent edge bends, and found that symmetry and orthogonality were preferred. Huang *et al.* found a preference for important nodes to be placed at the top of the drawing. Subsequent studies by Ware *et al.* [WPCM02] and an eye-tracking study by Huang [Hua07] have suggested that the negative effect of edge crossings is reduced if the edges cross at large angles and that continuity of edges through nodes is important in path following. A recent study by Marriott *et al.* [MPWG12] examined the effect of these aesthetics on recall. Starting with Himsolt [Him95] some user studies have used preferences and task performance to compare different automatic layout algorithms and layout styles [Pur98, PCA02, HHE07].

4.1.2 Human-authored Layout Studies

We believe that a new kind of user study pioneered by van Ham and Rogowitz [vHR08] can provide particularly important formative input for network layout algorithm design. In such studies participants are asked to manually draw or edit graphs. Their drawings may reveal the existence of aesthetic criteria not previously considered and also provide insight into how people trade off the different competing criteria.

These studies of human-composed network layout have reported a number of findings about human preferences for layout that have not yet been incorporated into algorithms. In particular, van Ham and Rogowitz found that people like to arrange "clusters" in graphs such that the edges form a convex boundary. Dwyer *et al.* [DLF+09] found that layouts with low stress were strongly preferred over layouts with large variance in edge lengths. The same study also found that users strongly preferred force-directed or user-generated layouts that resemble force-directed layout over an orthogonal diagram produced by the GIOTTO-Kandinsky algorithm (as implemented in the yFiles library).

Most recently Purchase *et al.* [PPP12] asked participants to draw a graph specified by an adjacency matrix using a graph drawing sketch tool. They found that edge crossings were avoided and that grid-like layouts were preferred. They also found that clusters were emphasised and that edge lengths were relatively uniform.

However, while the few previous such studies [vHR08, DLF+09, PPP12] are certainly relevant to understanding what humans like in orthogonal network layout, none considered orthogonal connector routing. We therefore conducted a human-composed network layout study specifically designed to identify the factors that humans regard as important for good orthogonal layout.

4.2 Experimental Design

Like [DLF+09, PPP12] our study had two stages. In Stage A participants were asked to manually lay out some small graphs, starting from an initial messy layout. This allowed us to see what kinds of layout people created, and by what process. In Stage B (different) participants were asked to rank the manual layouts. This allowed us to identify the manual layouts that really were regarded as being of high quality. We also included the initial messy layout and a layout automatically generated by yFiles in this ranking.

4.2.1 Stage A

Apparatus & Materials: Participants were asked to manually improve the layout of 8 small graphs. The initial layouts are shown in the leftmost column of Figure 4.1 while other columns show some of the manual improvements. The graph stimuli had between 4 and 13 nodes and the initial layout was quite messy with many bends and crossings. Small graphs were used because it was unrealistic to require participants to spend the large amount of time needed to manually layout graphs with more than this number of nodes.

To edit the layout participants used an easy-to-use web-based interactive layout tool explicitly designed for manually editing orthogonal graph layouts. The tool, also called Orthowontist, was programmed in HTML5/JavaScript. It allowed the user to move nodes, to add, delete or move bend points in the orthogonal connector routes, and to change the position in which a connector enters the node. The tool logged editing actions and their time as well as recording the final layout. See Figure 4.2.

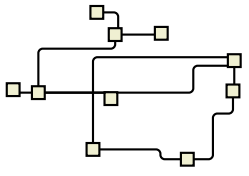
Participants: The study was advertised on *Monash Memo*, a university-wide bulletin. Seventeen participants undertook the study and all completed it. Three $50 gift cards were offered as incentive, and participants were instructed that they would win one of these if their layouts were ranked in the top three in Stage B of the study.

Procedure: Participants completed the study online, with the graph editing tasks taking an average 15 min 6 sec in all. This stage of the study had four components:

1. After reading an explanatory statement and signing a consent form participants completed a short questionnaire to ascertain their prior experience with node-link diagrams. The study was done anonymously, but participants were also asked if they wished to leave contact details in case they won one of the gift cards.

2. The participants completed training in the use of Orthowontist.

3. In the main part of the study the participants were presented in turn with the 8 graphs in random order. Participants were asked to edit each diagram until they felt that it "looked good" and clearly conveyed the connections between the nodes. They were asked to imagine that the diagram would be used to convey information and should be clear and readable. Participants were instructed that the experiment



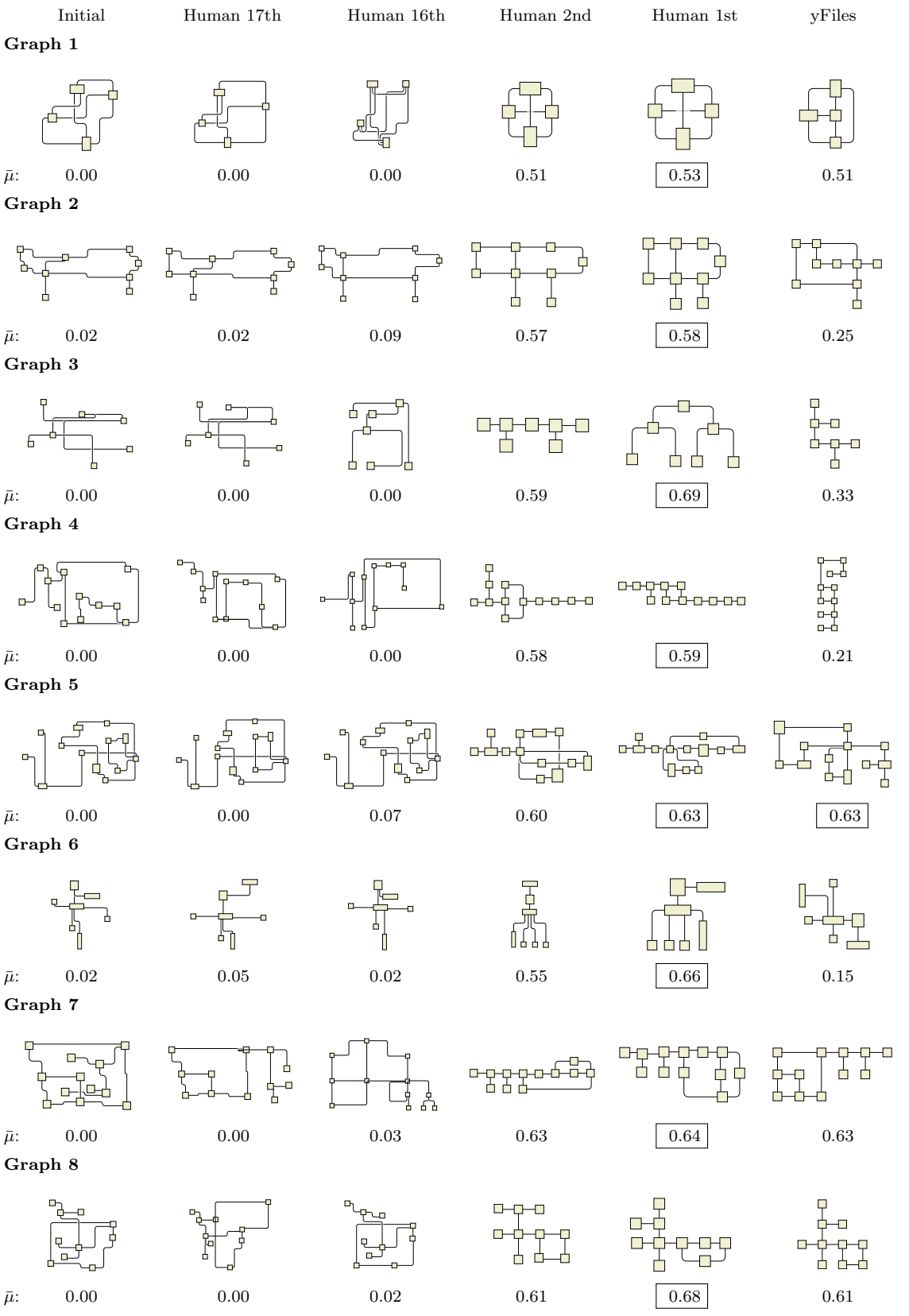
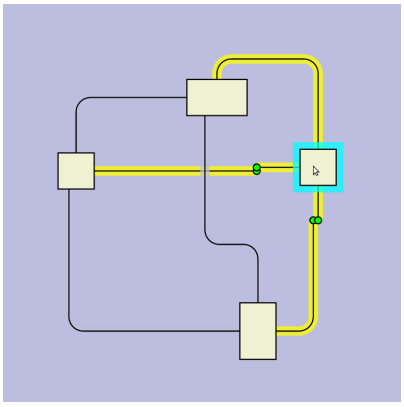
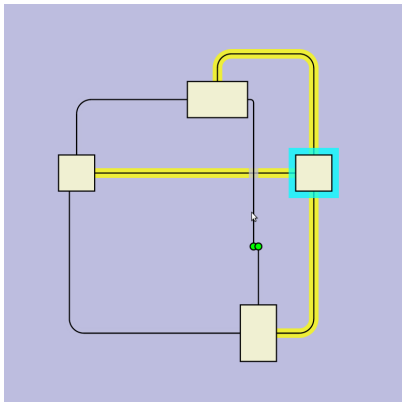


Figure 4.1: The 8 graphs and some of their layouts from the study. At left is the initial layout. The next 4 columns show the two worst and the two best manual layout. The final column shows automatic layout from yFiles. $\bar{\mu}$ = normalised inverted mean rank (see Section 4.3.3). Best possible value is 1, worst possible 0. Means in boxes indicate best actual mean rank.



(a) Bend point "snap-to" when dragging a node



(b) Dragging an edge

Figure 4.2: The Orthowontist online editor. We considered existing editors yEd and MS Visio, but found the controls for editing orthogonal connectors to be overly complex, so devised the simple interface employed in Orthowontist.

was not timed, and they could take as long as they liked. Once they were satisfied with the layout they moved to the next graph.

4. Finally the participants were asked to write down what their goals were when improving the layout of the networks.

4.2.2 Stage B

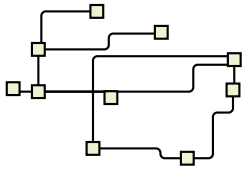
Apparatus & Materials: In the second stage participants were asked to choose the best layout obtained from Stage A for each of the 8 sample graphs. In addition to the seventeen human-made layouts of each graph, we also included the original messy layout, as well as a yFiles layout computed for that graph by the *classic orthogonal layout* command in the yEd diagram editing software (version 3.9.2) with default settings.[1]. There were thus eight graphs $g_1, g_2, \ldots, g_8$ with nineteen layouts each. We denote by $L_j^{(i)}$ layout j of graph g_i, where $i \in \{1, 2, \ldots, 8\}$, and $j \in \{0, 1, \ldots, 18\}$, with $j = 0$ meaning the original messy layout, $j = 18$ meaning the yFiles layout, and $j \in \{1, 2, \ldots, 17\}$ meaning the layouts created by the seventeen human respondents of Stage A.

Because of the large number of graphs to compare we used a tournament structure to identify the best layout. This meant that participants were only required to vote for the best layout out of the three presented to them in each match of the tournament. We wrote a web-based tournament tool so that the study could be conducted online. See Figure 4.3.

Participants: The study was advertised on *Monash Memo*. It was completed by 66 participants. A $50 gift card was offered as incentive, and it was explained that the winner would be the participant whose choices were the closest to the aggregate choices, thus that in order to win your best strategy was to choose the layouts which you thought other participants would also choose.

Procedure: The survey was conducted online. Upon loading, the tournament software organised the human-made and yFiles layouts for each graph g_i into a tournament structure with random seeding; i.e. the eighteen layouts of positive index $L_1^{(i)}, L_2^{(i)}, \ldots, L_{18}^{(i)}$ were

[1]The yFiles orthogonal layout algorithm has many options. After experimentation we came to the conclusion that the developers have already tuned the default settings to give best all-around results. For this reason, and also for easy reproducibility, we chose to use default settings across all graphs used in our studies.



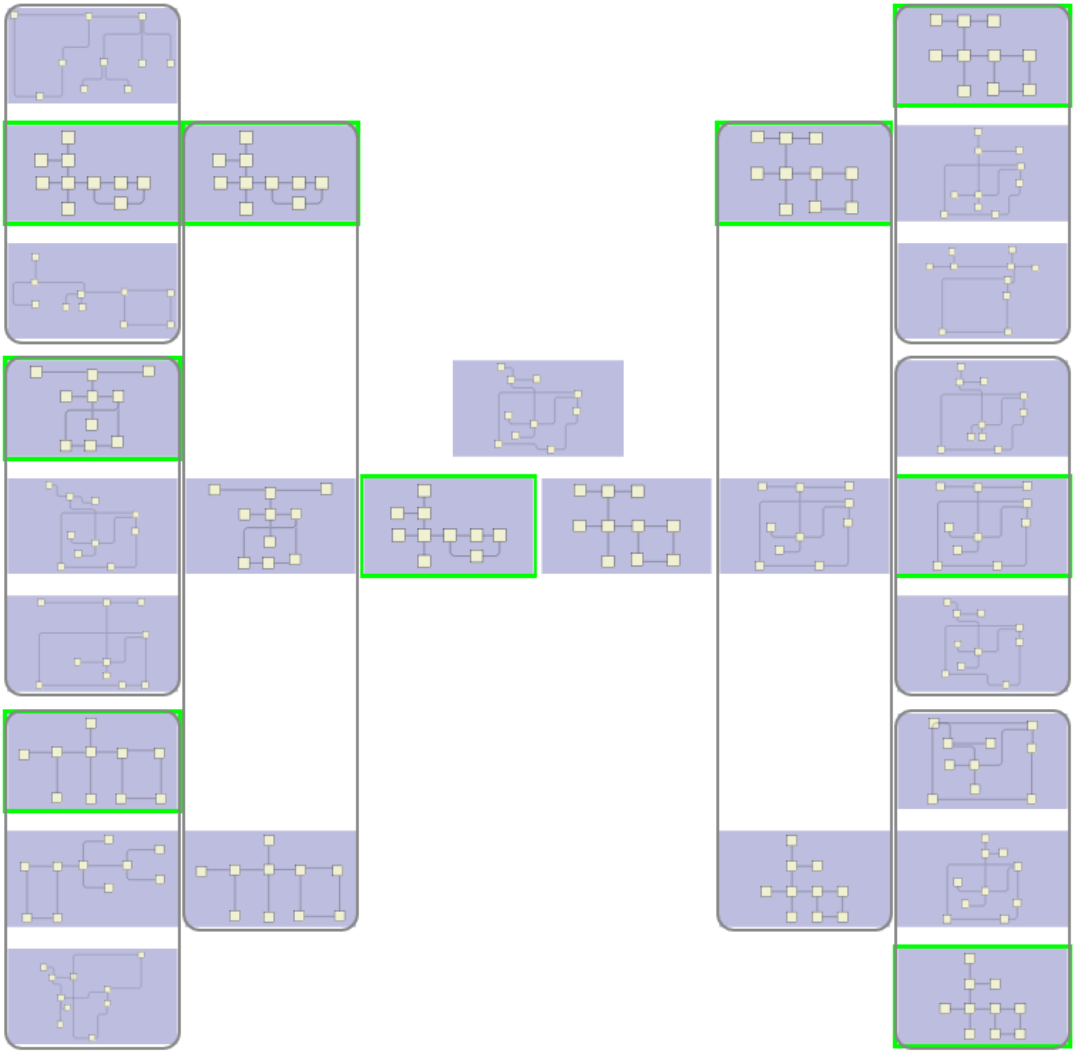


Figure 4.3: An example tournament for one of the eight graphs, and for a single participant of Stage B. Participants were *not* presented with a figure like this, but were instead simply shown three layouts at a time, constituting each match of the tournament. For each participant the "seeding" of the tournament was randomised, which in terms of this figure means that the order of the layouts in the far-left and far-right columns was random. The seventeen human-made layouts and the yFiles layout entered the tournament on equal footing, while the original messy layout received a "bye" to the final round, as a sanity check.

shuffled into a random order to make the seeding of the tournament. For each *match* of the tournament the participant was presented with three layouts and asked to choose the best one. The tournament thus fell into three *rounds*, with six matches in the first, and two matches in the second, at which point the two best layouts among the eighteen of positive index had been chosen. For the final round these two layouts were pitted against the original messy layout $L_0^{(i)}$. Participants spent an average of 6.57s per choice (discounting outliers of one minute or longer), and the entire survey took an average of 7 min 53 sec to complete.

In order to weed out "mindless clicking" from our results we considered whether any participant consistently selected the first, second, or third layout presented to them with excessive frequency. We found that for 65 of the 66 participants, no single choice (first, second, or third) was selected more than half the time. However, for the 66th participant the third layout was chosen 88 percent of the time. It seemed reasonable therefore to exclude this one participant, but keep all of the other 65. This decision was reinforced when we observed that the choices of this one rogue participant also had the lowest correlation with the aggregate choices out of all 66.

4.3 Results: Aesthetics

In this section and the next we analyse what we call the "static data", meaning the final layouts completed in Stage A of the study, and we see how the rankings established in Stage B correlate with various layout metrics such as stress, symmetry, and others. In Chapter 9 we review the "dynamic data", meaning we consider how these same layout metrics varied over time as our participants created their layouts. Thus, the purpose of the present section is to learn something about what a final layout should look like, while in Chapter 9 we instead seek to learn something about how human beings go about creating such layouts. The present section is concerned with aesthetics, while Section 4.4 is concerned with perceptual organisation.

4.3.1 Questionnaire

In response to the familiarity questionnaire of Stage A, 100% of participants said they were familiar with node-link diagrams, 94% said they had used a node-link diagram created by someone else, and 59% said they had created a node-link diagram in order to convey information to others.

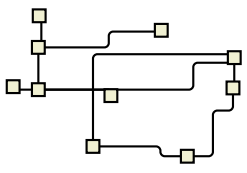
In the answers to the post-task question about their aims when manually laying out the graphs, many participants mentioned untangling the graph and removing crossings; this seemed to be the most basic aim. Some mentioned symmetry, overall shape, balance, and trying to lay it out on a grid. Another mentioned "layout the less connect[ed] node[s] on [the] outside in a diagram." See Appendix C for the complete responses.

4.3.2 Time

Excluding outlying inter-drag pauses of a minute or longer, the average time spent editing each graph ranged from 1 min 4 sec (Graph 1) to 3 min 22 sec (Graph 5), with an overall average of 1 min 53 sec per graph.

4.3.3 Rank

Based on the tournament results in Stage B we computed the *mean rank* of each of the 19 layouts for each of the 8 stimuli/input graphs. Equal ranks were averaged so that the winner of a tournament received rank 1, layouts that lost in the final round rank 2.5 each



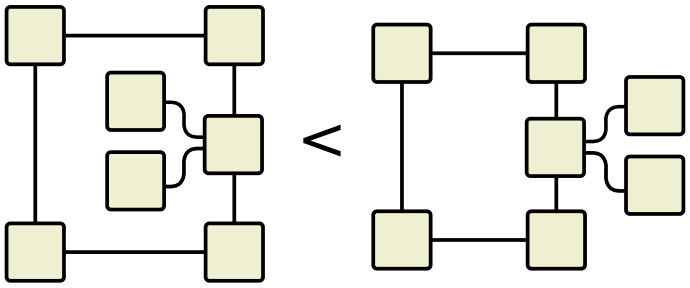


Figure 4.4: Overall, arrangements like that on the right where trees were placed on the outside of the layout were preferred over arrangements like that on the left, where trees were placed in inner faces.

Table 4.1: Pearson's correlation coefficient between normalised inverted mean rank $\bar{\mu}$ and various indicators of the quality of a layout. This table shows mostly positive correlations, indicating features that make a better layout. See Table 4.2 for negative correlations. Single star * means significance at $p = 0.05$ level. Double star ** means significance at $p = 0.01$ level. A '–' indicates a feature not applicable to that graph.

	Compactness	Gridiness	Symmetry	# Outer trees
Graph 1	**0.734****	**0.517***	**0.662****	–
Graph 2	**0.725****	**0.658****	**0.618****	0.148
Graph 3	**0.687****	**0.623****	**0.851****	–
Graph 4	**0.695****	**0.883****	**0.803****	**0.740****
Graph 5	**0.870****	**0.805****	**0.663****	-0.055
Graph 6	**0.698****	**0.698****	**0.625****	–
Graph 7	**0.703****	**0.712****	**0.866****	**0.491***
Graph 8	**0.759****	**0.754****	**0.856****	**0.749****

(the mean of 2 and 3), layouts losing in the second round rank 6.5 each (the mean of 4, 5, 6, 7, 8, 9), and layouts that lost in the first round received rank 14.5 (the mean of $10, 11, \ldots, 19$). See Figure 4.3. These ranks were averaged over all participants to compute the mean rank μ for each layout of each graph. This was then adjusted to a *normalised inverted mean rank* $\bar{\mu} = 1-(\mu-1)/13.5$ ranging from 0 to 1, with 1 being the best possible score, and 0 the worst. Figure 4.1 shows the original layout, the two best and two worst manual layouts for each of the input graphs, as well as the yFiles layout.

4.3.4 Analysis

Our analysis identified nine significant results which we will denote **R1-9** in this and later chapters of the thesis. See Appendix D for the definitions of our metrics. The first two findings are novel:

R1: Users like trees placed outside. To be precise, users largely prefer that the maximal acyclic components be placed outside the graph and not in inner faces. See Figure 4.4. This is in line with users wishing to separate clusters [vHR08, PPP12] and is supported by the positive correlations seen in the "# Outer trees" column in Table 4.1. It is also consistent with users wishing to decrease stress in the graph.

R2: Users create "aesthetic bend points". In contrast to previous research, we observed that while bends overall were correlated with a poor ranking, nevertheless certain bend points serving an obvious (to us) aesthetic purpose were present in most of the top-ranked human layouts. This is true for *all* except Graph 4 in the 'Human 1st' column of Fig. 4.1. In particular unnecessary bend points appear to have been introduced to emphasise symmetry or to ensure that if a node has two edges they are on opposite sides of the node, perhaps to ensure continuity in path following. See Figure 4.5.

Table 4.2: Pearson's correlation coefficient between normalised inverted mean rank $\bar{\mu}$ and various indicators of the quality of a layout. This table shows mostly negative correlations, indicating features that make a worse layout. See Table 4.1 for positive correlations. Single star * means significance at $p = 0.05$ level. Double star ** means significance at $p = 0.01$ level. A '–' indicates a feature not applicable to that graph.

	# Crossings	# Bend points	Seg Length Std Dev	Stress
Graph 1	**-0.427***	**-0.705****	-0.339	-0.038
Graph 2	-0.253	**-0.630****	**-0.411***	**-0.565****
Graph 3	**-0.436***	**-0.508***	**-0.798****	**-0.734****
Graph 4	–	**-0.666****	**-0.813****	**-0.885****
Graph 5	**-0.511***	**-0.569****	**-0.614****	**-0.563****
Graph 6	–	-0.048	**-0.602****	0.212
Graph 7	**-0.586****	**-0.626****	**-0.693****	**-0.708****
Graph 8	**-0.660****	**-0.733****	**-0.863****	**-0.898****

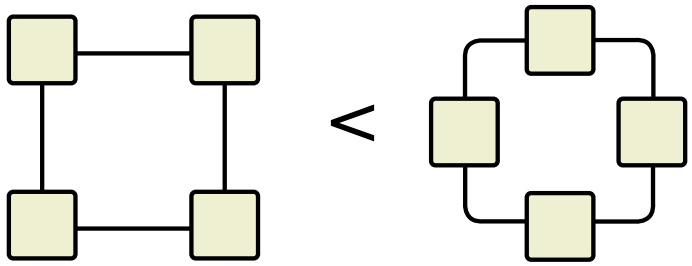


Figure 4.5: Users frequently seem to introduce "aesthetic bend points" i.e. bends in connector routes that allow greater symmetry or allow nodes of degree two to have their edges on opposite sides of the node.

Furthermore we found correlations (Table 4.1) showing that user preference is positively correlated with several metrics, whose precise definitions are given in Appendix D:

R3: compactness

R4: grid-like node placement ("gridiness")

R5: symmetry

and negatively correlated (Table 4.2) with:

R6: edge crossings

R7: edge bends

R8: standard deviation of edge segment length

R9: stress

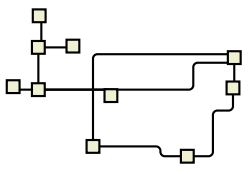
Results **R3** through **R9** accord with the conclusions of earlier studies [vHR08, DLF+09, PPP12].

Our experiment confirmed that manual layout leads to quite a different style of layout than that computed by the most widely used orthogonal layout algorithm. Furthermore humans significantly prefer the best human layout ($\bar{\mu} = 0.621$) to that produced by yFiles ($\bar{\mu} = 0.415$), as confirmed by a Wilcoxon signed rank test with $p = 1.286\text{e-}9$.

Trade-offs: We believe an important reason why manual layout is preferred is its use of trade-offs. Current algorithms for orthogonal layout are designed to first minimise edge crossings, then bend points, and finally area, whereas humans are more flexible and will keep edge crossings and bends if this reveals symmetries, separates clusters, reduces segment length or improves compactness.

4.4 Results: Perceptual Organisation

One of the hallmarks of hand-made layout is the presence of special patterns and arrangements of subgraphs, i.e. the perceptual organisations introduced in Section 2.3.1. These arrangements tend to emphasise something about the structure of the graph, and they



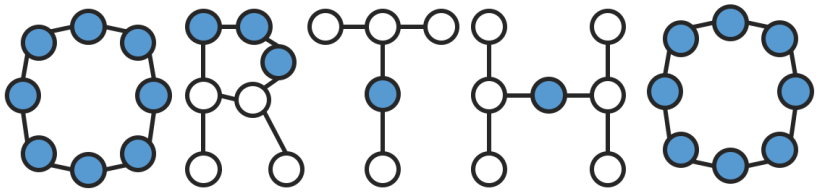


Figure 4.6: Links (blue nodes) can form both open chains (as in the letters 'R', 'T', and 'H' above) and closed chains (as in the letter 'O'). A closed chain must form a connected component unto itself, and is a special case that will not often concern us.

exhibit a clear concept and design. Meeting our expectations, these sorts of arrangements were abundant in the more highly ranked hand-made layouts in our experiment. In particular we note three kinds of organisational patterns: *chains*, *trees*, and *circuits*.

Chains

The simplest perceptual organisation is the alignment—horizontal or vertical—of a substructure we will call a *chain*. To define this we introduce the notion of a *link*, by which we mean simply any node of degree 2. A *chain* is then a maximal connected subgraph consisting entirely of links.

Chains come in two varieties: *closed* (forming a loop) and *open* (connecting to other nodes at each end). To make these notions precise we introduce some definitions. If H is any subgraph of a graph G, then by the *boundary* of H we mean the set of all nodes in $G \setminus H$ that have a neighbour in H, and we denote this[2] by ∂H. By the *closure* of a subgraph H we mean its union with its boundary, $H \cup \partial H$. Finally then, a closed chain is one that equals its own closure, while an open chain is one that does not. The former must be a closed cycle of links and must form a connected component unto itself, while the latter must connect at each end to a node that is not a link. See Figure 4.6.

Aligning the nodes of an open chain horizontally or vertically seems a natural way to emphasise and exhibit this structure. Indeed, we found that this was a common feature of the preferred layouts of Graph 4 for example, which featured the longest open chain of any of the eight graphs (Figure 4.7).

Trees

We have already considered trees—maximal acyclic subgraphs—and observed that users prefer for these to be placed outside the graph, not inside. In some cases, such as Graph 3 in our study, the entire graph may itself be a tree. What we want to now consider is the *shape* trees are given in highly-ranked layouts.

In the case of Graph 3 a layout emphasising the hierarchical structure of the tree was strongly preferred over all others (Figure 4.8a).

On Graphs 7 and 8 a layout employing a similar hierarchical structure for a subtree ranked among the top four hand-made layouts. More highly ranked layouts of these graphs gave the subtree a less regular hierarchical structure. See Figure 4.8.

Circuits

Following Tutte [Tut60], a *path* in a graph G is a finite sequence

$$P = \langle v_0, e_1, v_1, e_2, \ldots, e_k, v_k \rangle$$

[2]We use a conventional notation from point-set topology for denoting boundaries. See [Mun00].

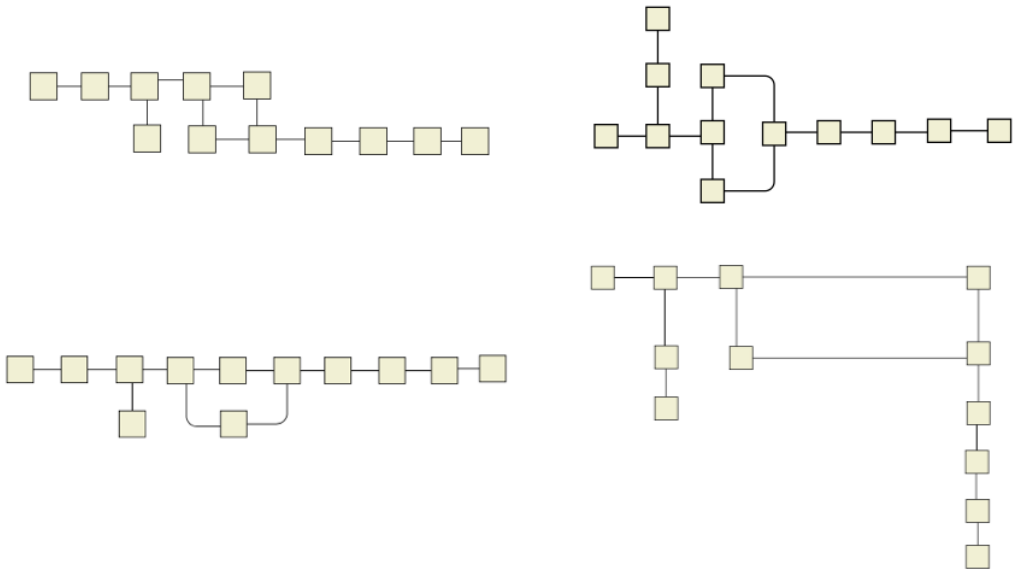
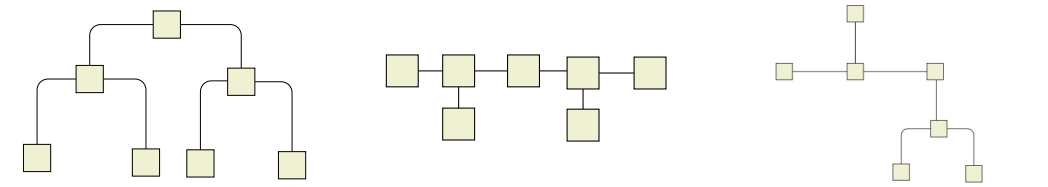
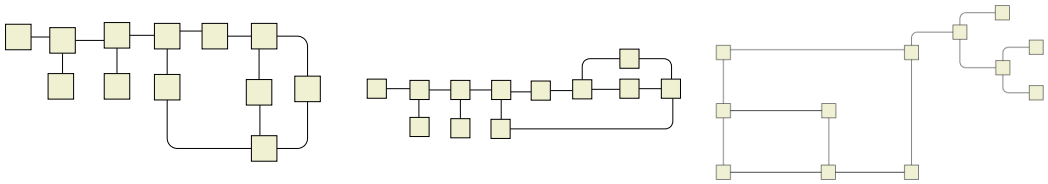


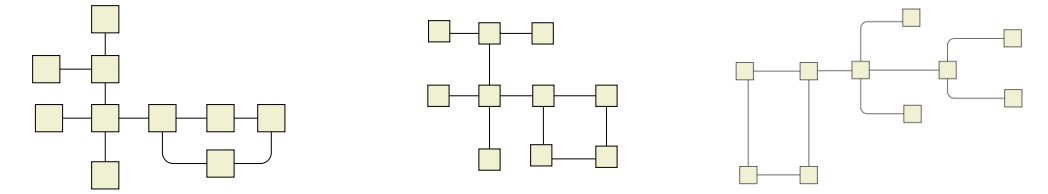
Figure 4.7: While the sixth best (lower right) layout of Graph 4 aligned the longest chain vertically, the three highest ranked layouts of that graph each aligned it horizontally.



(a) Graph 3, best layout (b) Graph 3, 2nd best layout (c) Graph 3, 3rd best layout

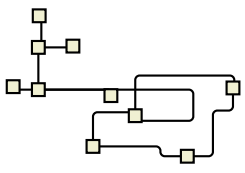


(d) Graph 7, best layout (e) Graph 7, 2nd best layout (f) Graph 7, 4th best layout



(g) Graph 8, best layout (h) Graph 8, 2nd best layout (i) Graph 8, 3rd best layout

Figure 4.8: Hierarchical configuration for a subtree was often found in highly ranked layouts. Figure 4.8a, the single highest ranked layout in the entire study, exhibits a clear schematic expression of the hierarchical structure of the tree. A similar formation is present in half of Figure 4.8c. For Graph 7 a layout featuring this kind of structure (Figure 4.8f) was ranked fourth; it is possible that its lack of compactness hurt its rankings. A similar layout came in third for Graph 8 (Figure 4.8g). For both Graphs 7 and 8 the two top-ranked layouts organised their subtrees neatly, but did not give them such a pronounced hierarchical structure. They were also very compact.



having at least one term, in which the v_i are vertices and the e_j edges of G, and where $e_i = (v_{i-1}, v_i)$ for $1 \leq i \leq k$. The path P is *degenerate* if $k = 0$. It is *circular* if it is non-degenerate and all its terms are distinct except that $v_k = v_0$. The edges and vertices of a path P define a subgraph $G(P)$ of G, and we call $G(P)$ a *circuit* of G if P is a circular path.

In many of the highly ranked layouts in our study, circuits were configured as rectangular faces. Several good examples are present in Figures 4.7 and 4.8, especially if we allow bend points to count among the "vertices" defining such a circuit. This appears to be another way in which human layout designers may organise a subgraph in order to emphasise its structure.

4.5 Conclusions

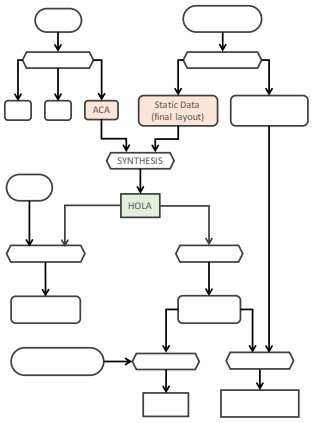
The experiment demonstrated how people may trade off various aesthetic concerns against one another. For example the favourite layout of Graph 1 features a deliberate edge crossing and four deliberate edge bends, all (apparently) in the service of symmetry. This is quite different from what happens in the Topology-Shape-Metrics layout system (Section 2.4), which puts a fixed hierarchy on aesthetics, first minimising crossings, then bends, and finally maximising compactness. In designing a more human-like orthogonal layout algorithm in Chapter 5 we will try to take a flexible approach to aesthetics, in which they can be traded off in a similar way.

Meanwhile the observed perceptual organisations of chains, trees, and circuits give us another layout goal that a more human-like algorithm must attempt to achieve.

Finally, we recall that as our study participants created their layouts the Orthowontist software recorded the editing process. This means that in addition to what we have learned in this chapter about the layout *product*, there also may be something to be learned by observing the *process*. In the flow chart of Figure 1.3 we named the former the *static data* and the latter the *dynamic data* of the Orthowontist experiment. The goal of the next chapter is to design a new orthogonal layout algorithm inspired by the findings from the static data, but we will return to the dynamic data in Chapter 9 when we develop interactive layout editing tools.

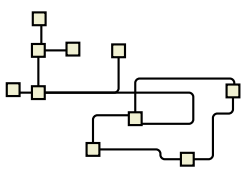
Chapter 5

Designing a human-like layout algorithm



Recall that in Section 2.3.1 we reviewed the complementary layout concerns of aesthetics and perceptual organisation. Based on the findings of the study from Chapter 4 we can now go beyond the preliminary work of Chapter 3 in addressing both these aspects of layout. This means devising a new orthogonal layout algorithm based on the ideas from Chapter 3 which should (a) create the same sorts of perceptual organisations that people create when working by hand, and (b) trade off the various aesthetic concerns against one another the way human beings do. For example it should sometimes deliberately create connector bends in service of other considerations like stress minimisation, symmetry, or even perceptual organisation itself. The new algorithm we develop is called HOLA: Human-like Orthogonal Layout Algorithm.

We start in Section 5.1 by choosing ACA as the foundational technique from Chapter 3 on which to build, not GS or NS. Then in Section 5.2 we consider how the ACA process can and should be generalised in order to reach the kinds of layout goals we now want to achieve. Next we get straight to the definition of the HOLA algorithm in Section 5.3 without further motivational discussion. The algorithm design is based on the findings of Chapter 4, and this constitutes Step 2 of our human-centred methodology: designing an algorithm based on the findings of the formative user study. However, as explained in Section 1.6, it will not be until Chapter 7 that we articulate a way of achieving this second step systematically. This is the natural order of development, since the system of that chapter will be based on an analysis and generalisation of the algorithm developed in this one.



5.1 Configuring with Constraints

If we apply pure stress-minimising layout as well as the three techniques NS, GS, and ACA from Chapter 3 to each of the eight graphs from the Orthowontist study, the results appear as in Figure 5.1. While some good grid-like aesthetics are present in some of these layouts, it is clear that something more will be needed in order to create perceptual organisations.

To begin with, consider chains. As Graph 4 demonstrates, merely snapping nodes to the nearest grid point (GS), or snapping them into alignment with nearby nodes (NS), can easily fail to align the nodes of a chain. Next consider trees. Graphs 3 and 7 for example confirm that snap forces cannot be counted on to create a hierarchical arrangement like in the top-ranked human layout of Graph 3 (see Figure 4.1).

ACA fails to organise trees hierarchically too, but it offers a more promising starting point than NS or GS. It is an iterative process in which we repeatedly pause to analyse the current layout before choosing the next constraint to be added. It is natural to generalise this by making the analysis more involved. The algorithm can look for trees and choose constraints to put them into the desired shape, and do likewise for chains, cycles, or any other substructures that are meant to be given some special perceptual organisation. On the contrary it is in no way clear how snap forces could be devised to achieve something like this. We therefore choose the ACA procedure as the template on which to develop the new algorithm, HOLA.

5.2 Generalising ACA

The ACA algorithm uses very simple criteria for selecting edges for alignment, and for choosing whether to align horizontally or vertically. Setting aside the various refinements we considered in Chapter 3 the basic idea of ACA is: among the edges that have not yet been aligned find one that is closest to aligned, either horizontally or vertically, and apply a constraint to align it in that way if possible.

While we found that this simple greedy application of constraints can significantly improve the grid-like aesthetics of an initially stress-minimal network layout, it is clear that we will need to do something more deliberate in order to create perceptual organisations or to make trade-offs between competing aesthetics. But to merely say that the simple approach of ACA is no longer good enough and that we now need "something more" is too open-ended. We therefore pause in this section to in a sense work out "the rules of the game". In other words we wish to understand what it means to generalise ACA, and what are the limits and boundaries that we want to stay within.

5.2.1 Goals

Since the only goal of ACA is to align edges, we may say that the edges of the graph are its *targeted substructures*. Now that we are interested in creating perceptual organisations we must target larger substructures. We must consider a whole chain, a whole subtree, a whole cycle. As ACA used the current position of a given edge as a guide in choosing *which* alignment it should be given (horizontal or vertical), we may want to use the current positions of the nodes to help us decide on a way of organising a given substructure.

The approach of ACA may be generalised in the pursuit of pure aesthetic goals as well, not just the goal of creating perceptual organisations. In order to achieve the many alignments that make the layout grid-like and well suited for orthogonal connector routing, we may try working node-by-node, rather than edge-by-edge. In other words instead of picking an edge and aligning it, we can pick a node and align up to four of its neighbours with it (one in each compass direction).

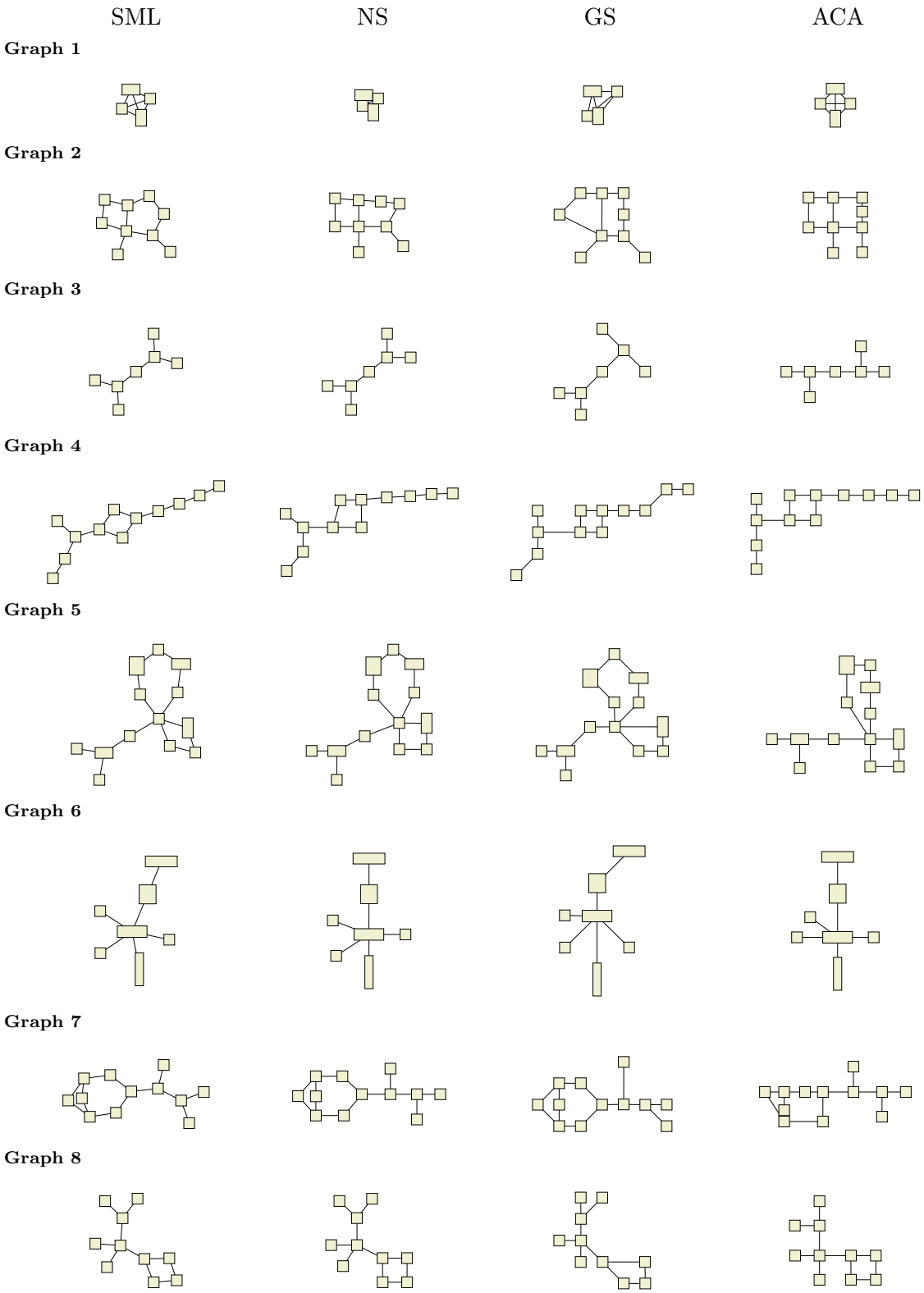
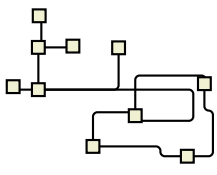


Figure 5.1: The eight graphs from the Orthowontist study laid out by pure stress-minimising layout (SML), NodeSnap (NS), GridSnap (GS), and Adaptive Constrained Alignment (ACA)



Thus, as we generalise the ACA process we see a generalising of the set of goals that we can address at each step of the algorithm. Instead of just asking which is the next edge to be aligned, we might ask which node should be aligned with its neighbours on all four sides, or which tree should be laid out hierarchically, or which chain should be straightened, or which cycle should be arranged as a rectangle.

5.2.2 Ordering

We know from our study (Chapter 4) and that of Dwyer et al. [DLF+09] that people like low-stress layouts. In ACA we tried to preserve the stress-based shape of the layout by applying the right constraints: When aligning an edge we asked with which axis it was already most nearly aligned, and then aligned it with that one. As we consider designing a more powerful algorithm, how much farther from the given position should we be willing to stray? That is, in applying constraints, which of the "choices" made by stress minimisation should we be willing to convert?

A useful guideline may be to refrain from altering the ways in which nodes have been *ordered* by stress minimisation, and there are two orderings to consider: *cyclic ordering*, and *orthogonal ordering*.

Cyclic ordering means the same thing as the *rotation system* we looked at in Section 2.4.1: the clockwise ordering of the neighbours of each node. As we saw, to alter the rotation system is to alter the topological equivalence class of the layout. That suggests that this is a major change, and perhaps one that we should avoid making lightly. In fact, as Figure 5.2 illustrates, changes to cyclic ordering are truly ill-advised from the perspective of stress minimisation. They can create sharp rises in stress that may be difficult or impossible to dissipate, depending on how many constraints are already in force in the layout. We keep this in mind as a guideline when designing HOLA.

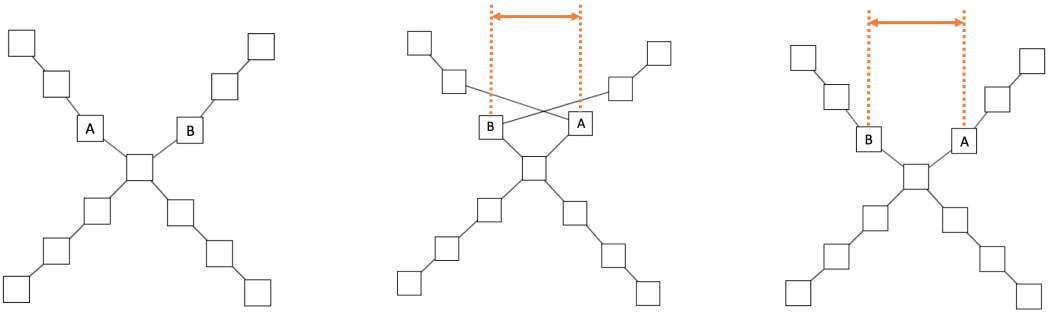
As for orthogonal ordering, this simply means the ordering induced on a set of nodes by considering one coordinate, x or y. In other words it is their left-to-right ordering or top-to-bottom ordering in the layout. For similar reasons to those illustrated in Figure 5.2, we should try to refrain from altering the orthogonal ordering of any two nodes. This is not to suggest an outright ban, but that this sort of operation should be used sparingly.

5.2.3 Recourse

In ACA, when a chosen edge alignment proved impossible to enforce (due to conflicting constraints already in place), we simply abandoned it and moved on to consider another edge. In a new algorithm with more complex goals there might be more room for taking recourse. A complex goal (such as giving a subgraph a good perceptual organisation, or aligning a high-degree node with four of its neighbours, one on each side) could involve a "Plan A" which should be attempted first, but if this fails due to conflicts with existing constraints then there could be recourse to a "Plan B" rather than abandoning the goal altogether and moving on to another.

5.3 The Steps of HOLA

HOLA has 4 main steps as listed in Figure 5.3 and illustrated in Figure 5.4(a–f). Recall our aesthetic goals **R1-9** from Section 4.3.4. Since symmetry (**R5**) is easy in tree layout (Step 3a), and users like trees on the outside (**R1**), we decided to start by decomposing the graph into trees and *core* (see Section 5.3.1), also reasoning that stress-minimisation could better reveal symmetries (**R5**) in the core once the trees had been removed. The flexibility of CSML allows us to then combine a series of new ideas and generalisations of



(a) Over the space of all layouts for this simple X-shaped graph there are many local stress minima, but they are all equivalent up to translation and rotation of the graph, and permutation of its four arms. Stress-minimising layout easily finds one of these minima, and node A happens to wind up west of node B.

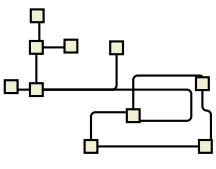
(b) Suppose we now use the PROJECT operation to impose the constraint that node B instead lie west of node A. This changes the cyclic ordering of the neighbours around the central node, and is likely to move the layout into a different stress basin.

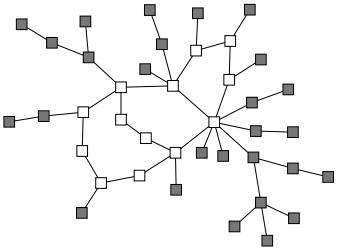
(c) Indeed, in this case an application of DESCEND now causes the entire "A arm" and "B arm" of the X to swap positions fully.

Figure 5.2: Changing the cyclic ordering of the neighbours of a given node via PROJECT (Section 2.2.4) can move the layout from a low point in one stress basin to a high point in a different basin, often necessitating drastic rearrangement of the graph in order to regain low stress. At best, an application of the DESCEND operation is required. At worst, the layout may become hopelessly tangled.

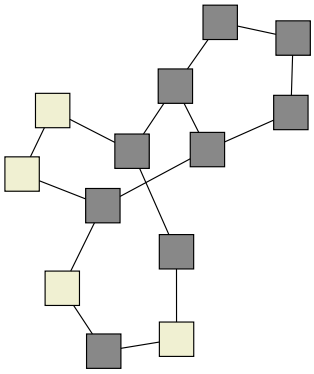
1. Topological decomposition of graph into *trees* and *core* (Figure 5.4a)
2. Layout of the core (Figures 5.4b, 5.4c):
   (a) Stress-minimising layout of core (**P1**:**R3,5,6,8,9**)
   (b) Greedy orthogonalisation of layout (**P2**:**R2,4,9**)
   (c) Orthogonal edge routing (**P2**:**R6,7,8**)
3. Tree layout and placement (Figures 5.4d, 5.4e):
   (a) Symmetric layout of each tree (**P3**:**R5**)
   (b) Planarisation of core
   (c) Insertion of trees into the core (**P3**:**R1**)
   (d) Stress-minimising layout (**P1**:**R9**)
4. Opportunistic improvement (Figures 5.4e, 5.4f):
   (a) Opportunistic alignment (**P2**:**R4**)
   (b) Node distribution by neighbour stress (**P2**:**R3,8**)
   (c) Rotation (**P2**)
   (d) Removal of dummy nodes and final orthogonal edge routing

Figure 5.3: Steps in the HOLA Algorithm, with cross-reference to our design principles **P1-3** (see below) and aesthetic goals **R1-9** (enumerated in Section 4.3.4)

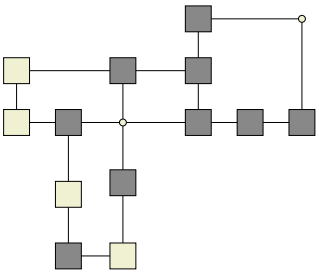




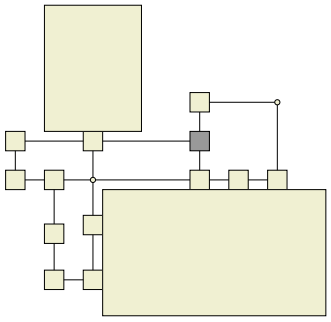
(a) Step 1: Trees (dark grey nodes) will be pruned.



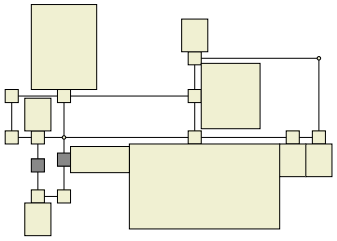
(b) Step 2: Stress-minimal layout of core. Dark grey nodes are the roots of trees.



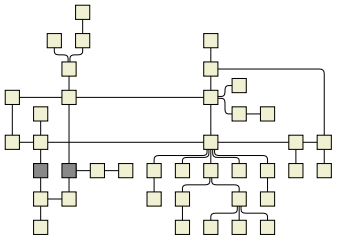
(c) Step 2 (cont.): Core orthogonalised, edges routed, graph planarised.



(d) Step 3: Two trees placed, the tree rooted at the dark grey node is next. It could be placed NW, SW, or E.



(e) Step 3 (cont.) and 4: All trees placed. Note: the two dark grey nodes are almost but not quite aligned.



(f) Final layout. Nodes that were nearly aligned have been exactly aligned. Tree nodes have been reinserted.

Figure 5.4: HOLA applied to a small example graph illustrating the four main steps.

ACA in Steps 2, 3, and 4 to further address the aesthetic goals (**R1-9**), as examined in greater depth below.

The design of the algorithm is guided by the following three Principles, and Figure 5.3 also indicates in which steps these are addressed.

P1: Use stress-minimisation (**R9**) to untangle the graph (**R6**) and reveal underlying symmetries (**R5**) as well as encouraging uniform edge length (**R8**) while keeping the layout compact (**R3**). Also, stress is always considered as one of the optimisation criteria throughout further modifications (**R9**).

P2: Apply incremental extensions/improvements to the existing layout, like an opportunistic human editor. In particular this is used to "tune" bend points (**R2**) and to achieve grid-like alignments where possible (**R4**).

P3: Subgraphs with tree structure are treated specially such that they can be arranged symmetrically (**R5**) and their placement can be controlled precisely (**R1**).

The sections below provide a high-level description of each of the steps of HOLA.

5.3.1 Topological decomposition

Step 1 is a *peeling* process [AMA07, AvHK06, Sei83] where all leaves are removed from the graph G repeatedly until none remain. The leaf nodes are added to a new graph H and reconnected to one another as they are added. When no leaves are left in G the remaining

subgraph C is called the *core*. In Figure 5.4a the dark grey nodes will be pruned by this process. Finally if L is the set of nodes in H, and $\rho : L \to V$ maps each pruned leaf node to the unique node to which it was attached at the time that it was pruned, then we form the set $R = \{\rho(\ell) : \ell \in L\} \setminus L$ of *root nodes*, and add to H a copy r' of each node $r \in R$, connecting it to each ℓ for which $\rho(\ell) = r$. The connected components of H then constitute some t trees $T_1, T_2, \ldots, T_t$, t a non-negative integer equal to the size of the set R, and each tree T_i has root node $r'_i \in H$ which is a copy of a node $r_i \in C$. See Figure 5.5.

If we have $t = 1$ and $C = \emptyset$, i.e. the graph G is in fact a tree, then we simply apply our symmetric tree layout procedure (see Section 5.3.3) and terminate. (See for example the HOLA layouts for Graphs 3 and 6 in Figures 6.12 and 6.13 in the next chapter.) Otherwise we proceed with Step 2, layout of the core.

5.3.2 Layout of the core

(a) Layout of the core graph C computed in Step 1 begins with a simple unconstrained stress-minimising layout (DESCEND), followed by the application of overlap removal constraints (OVERLAP-REMOVAL from Section 2.2.4).The first of these steps gives the nodes a natural and low-stress distribution in the plane (Figure 5.4b). As we begin to orthogonalise the layout we will try to keep the stress low, in service of **P1**.

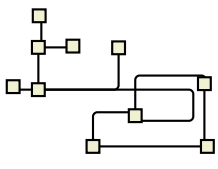
By allowing a simple application of DESCEND to decide the overall distribution of the nodes in the plane, we are making a trade-off between the goals of minimising crossings and minimising stress. This is motivated by our findings on human layout from Section 4.3.4. It is an essentially different approach from that of Topology-Shape-Metrics, which prioritises crossing minimisation over all other aesthetics.

(b) Orthogonalisation proceeds in two parts, which we call *node configuration* and *chain configuration*. The process is like ACA but with generalised goals, as discussed in Section 5.2.1.

Node configuration. During node configuration we sort all nodes v of degree 3 or higher by falling degree, and visit them in order. For each node v we will align at most one of its neighbours in each of the four cardinal compass directions, NORTH, SOUTH, EAST, and WEST relative to v, and we call this a *configuration* of v. We have determined by experiment that configuring the highest degree nodes first tends to result in more favourable configurations, i.e. in a greater total number of alignments. Meanwhile links (or degree-2 nodes, as defined in Section 4.4) are left out of this process entirely. Recall that the maximal subgraphs consisting entirely of links are called *chains*. These are configured in the following step, according to different principles (see below).

As we visit each non-link node v we attempt to assign as many as possible of its neighbours to the four compass directions, favouring an assignment that minimises the total angular displacement of these nodes relative to v. The configuration is achieved by projecting onto separated alignments, as in Section 3.3. Following principle **P1** and the discussion in Section 5.2.2, we prohibit any configuration that would alter the cyclic ordering of the neighbours of v. We also prohibit reversals of the orthogonal ordering of any neighbour with respect to v. E.g. if u was a neighbour of v with $u_x < v_x$ before configuration, then we would require $u_x \leq v_x$ after configuration; in other words u could not be assigned EAST. If in addition we had $u_y > v_y$ then u also could not be assigned NORTH.

As was the case with the ACA process, we must ensure that we do not create and enforce edge overlaps. Therefore as constraints accumulate we will get into situations where the ideal configuration, or "Plan A" for a given node would conflict with existing alignments. In such a situation we compute the next best configuration ("Plan B") that will work, as discussed in Section 5.2.3



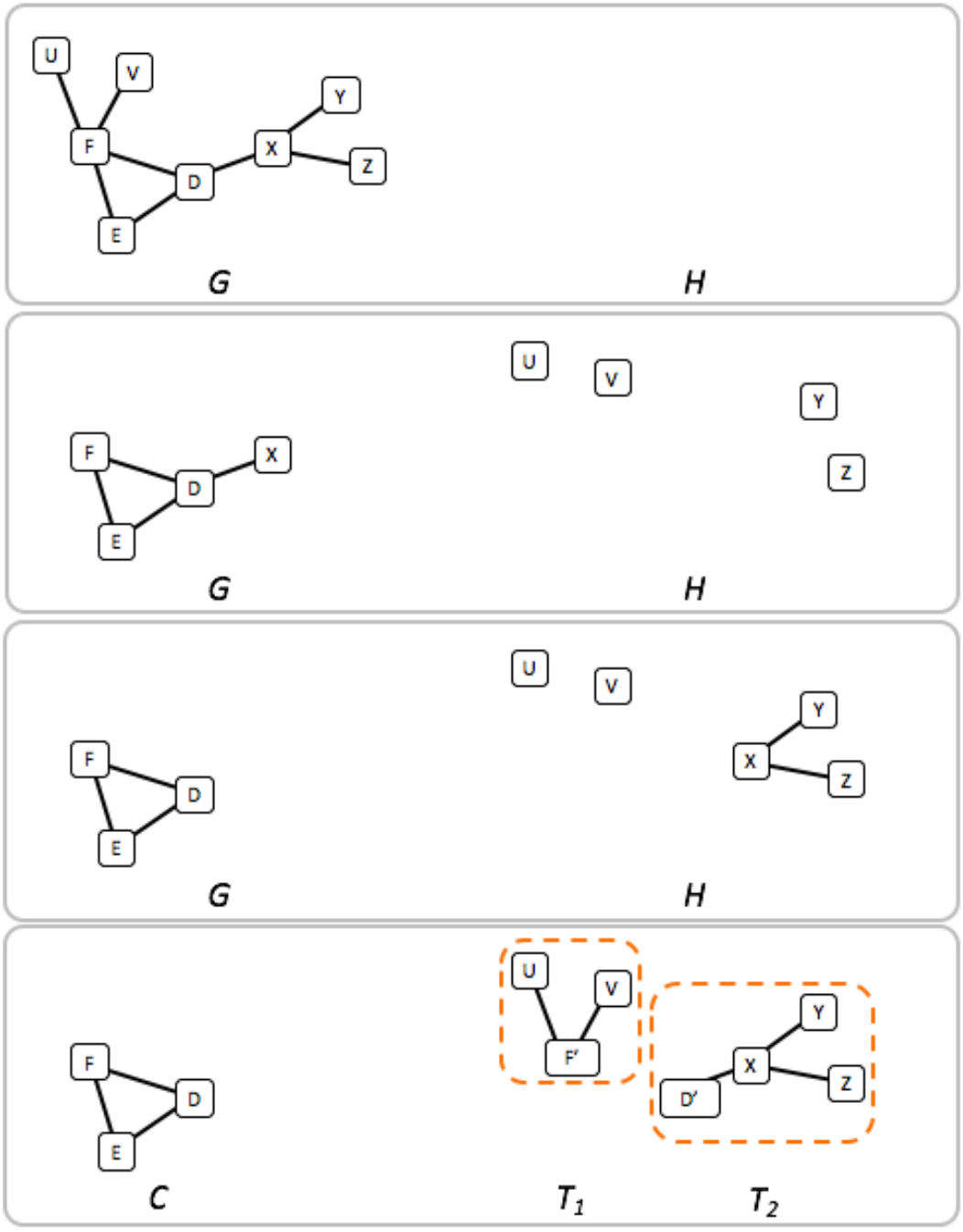


Figure 5.5: The peeling process, and assembly of trees with copies of root nodes. In this case we have $R = \{D, F, X\} \setminus \{U, V, X, Y, Z\} = \{D, F\}$ so the two root copies D', F' are added to H. Then the connected components T_1, T_2 of H are the two trees, while the core C is left over from the original graph G.

See for example Figure 5.6a. The optimal configuration around centre node c, i.e. that which minimises angular displacement, would assign u, v, w to WEST, NORTH, EAST, respectively. But if (c, w) and (u, x) are already horizontally aligned, and (w, x) vertically aligned, then this configuration is impossible. In a case like this HOLA correctly recognises that assigning u, v, w to SOUTH, WEST, EAST (resp.) is the next best configuration. That is, it has the next smallest total angular displacement of the neighbours of c, while still maintaining the existing orthogonal and cyclic ordering.

As we greedily assign the best configuration we can to each non-link node, we use the PROJECT operation to apply each new set of constraints. In general this causes stress to climb, and step by step the geometry departs from a stress-minimal position. Due to our decision not to reverse orthogonal orderings (Section 5.2.2) the current geometry dictates which subsequent node configurations we are willing to attempt. Therefore if for any node v we find that *none* of the eligible configurations is feasible (i.e. consistent with existing constraints) then we apply DESCEND and try again. The minimisation of stress may present us with a different set of eligible configurations to try. We only do this once, and if v again fails to take any configuration then we move on to the next node. We will have more to say about this kind of interplay of the PROJECT and DESCEND operations in Chapters 6 and 7.

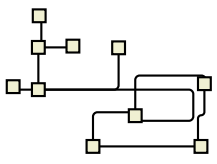
When we have visited all non-link nodes and attempted to configure each one, we apply a final DESCEND to complete the node configuration step. This is because the chains in the core remain unconstrained, and stress-minimisation will now cause them to settle into balanced, well-distributed arrangements, smoothing any jagged corners that may have arisen. This permits us to again honour **P1** as we move on to chain configuration, the second step in the orthogonalisation process.

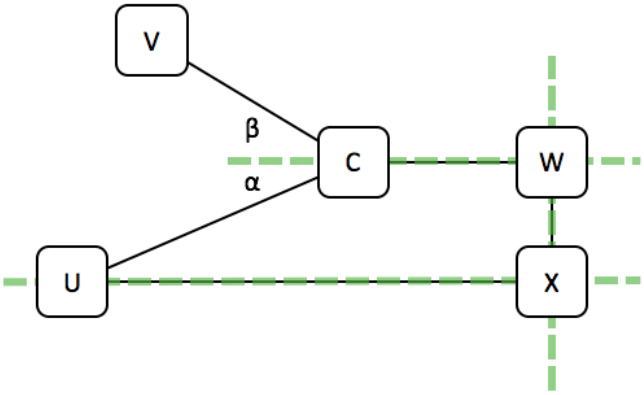
Chain configuration. Let a chain C be given, consisting of the links $v_1, v_2, \ldots, v_k$, and let nodes u and w be the outside neighbours of the first and last links v_1 and v_k, respectively (so $\partial C = \{u, v\}$). Edges $e_0 = (u, v_1)$ and $e_k = (v_k, w)$ may or may not have been aligned already by the node configuration process. For example, v_1 might be aligned EAST of u, and v_k NORTH of w, as in Figure 5.6b. In such a case, choosing where to enforce bends in the chain is similar to the process of routing an orthogonal connector when the connection directions at each endpoint are given. As shown in Figure 2 of Wybrow et. al. [WMS10], the minimal number of bends required, and the direction of each bend (left or right) depends on the relative positions of u and w as well as the connection directions. If the edge e_0 is not yet aligned we simply consider both of the possible compass assignments of v_1 relative to u that preserve their orthogonal ordering, and likewise with e_k.

For a given chain there may be different minimal-length bend sequences, e.g. (R, R, L) or (L, L, L) as in Figure 5.6c. We evaluate each potential bend sequence by a greedy process that chooses locally optimal points at which to create each bend in the chain. *Here we depart from the orthodoxy that bend points are always bad*, as we consider both nodes and edges as potential bend points in the chain. According to **R2**, we believe that a bend point may be deliberately created in an edge in service of other aesthetic principles, namely stress-minimisation and symmetry. This is one example of our attempt to follow **P2**, and work like an opportunistic human editor.

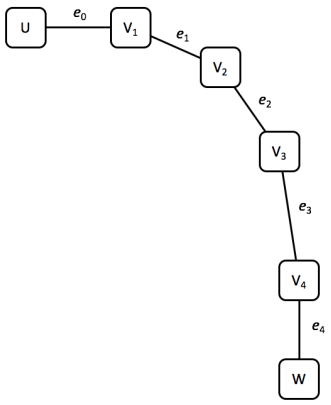
The basic idea of our greedy process is simple: in order to gauge how suitable an edge is for becoming a bend point we measure how close its slope is to ± 1 in the plane. For a node we likewise consider the slope of the base of an isosceles triangle with apex at the node's centre and with one leg parallel to each of the node's edges (Figure 5.6d).

The cost of creating a bend at a given edge or node is measured as a certain increasing function of the difference of the slope at that place from ± 1 (sign chosen according to bend direction), and the cost of a bend sequence is the sum of these costs. We greedily attempt to choose a minimal-cost bend sequence, and enforce it, aligning each edge of the

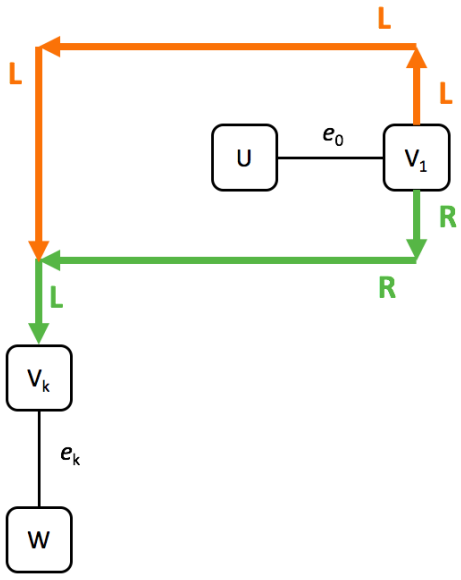




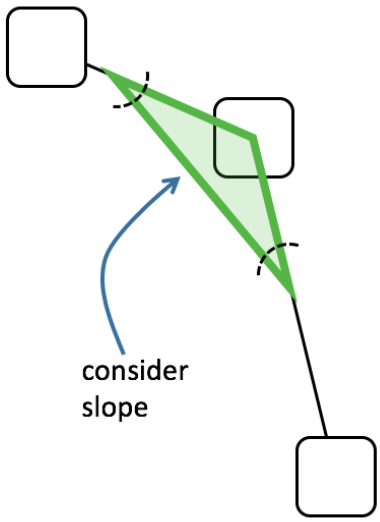
(a) Choosing "Plan B" when locally configuring a node. The pairs (c, w) and (u, x) are horizontally aligned, while (w, x) is vertically aligned. The angles α, β satisfy $0 < \alpha < \beta < \pi/4$.



(b) We must choose the best place(s) to create orthogonal bends in each chain, depending on how it connects to its boundary nodes.



(c) There may be more than one possible minimal-length bend sequence.



(d) Determining suitability of a node to become a bend point

Figure 5.6: Steps in HOLA's configuration process

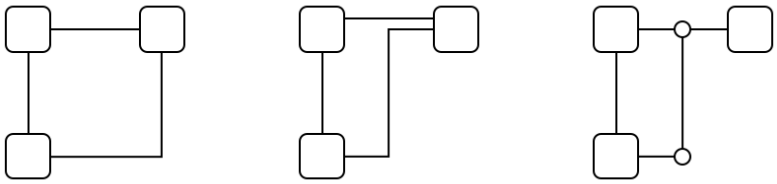


Figure 5.7: An edge routing like that on the left is desirable; a routing like that in the centre cannot be allowed, since it would cause a node to become a leaf when the graph was planarised, as on the right.

chain vertically or horizontally with new alignment constraints onto which we PROJECT. After thus processing each chain, the chain configuration step is complete.

As a configurable option the ACA algorithm may be applied to the chains instead of this process. We have found that this gives better results on large graphs, as will be seen in our evaluation of HOLA in Chapter 6, but not on small graphs as in the first study.

(c) For the low-density graphs at which our algorithm is targeted (Section 1.5) most of the connectors will now be mere straight axis-aligned segments as a result of the orthogonalisation. However some diagonal connectors may remain, and in Step 2c we compute an orthogonal routing for each of these using ORTHO-ROUTE (see Section 2.2.4).

By construction every node in the core C has degree at least two, and we take care to ensure that connectors attach to at least two of the four sides of each node, lest the node become a leaf in the planarisation of Step 3b (see Figure 5.7). This is important because subsequent steps operating on the core graph C depend on the assumption that C contains no leaves.

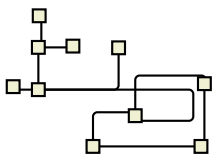
5.3.3 Tree layout and placement

In this step we now incrementally add the trees back into the core in accordance with **P3**.

(a) We first determine a layout for each tree T_i. Motivated by the schematic tree configurations we considered in Section 4.4 and by the desire to create symmetry, we apply the symmetric tree layout algorithm of Manning and Atallah [MA88]. Each tree is provisionally given SOUTH growth direction as in Figure 5.8, meaning that each rank is horizontally aligned and appears below, i.e. with greater y-coordinate than the prior rank, starting with the root node r'_i. If the tree structure is in fact symmetric with respect to the root node then the layout will be symmetric about a vertical axis through the root node; otherwise it will be as close to symmetric as possible about this axis in a certain well-defined sense (namely the "c-trees" [MA88] are paired off about this axis to the extent possible—see Figure 5.8). The edges of each tree are routed orthogonally [WMS10] with all edge connections on the sides facing the opposite rank.

(b) Next we determine how to place each tree in the core. We begin by planarising the core in order to give it a well-defined set of faces into which the trees can be placed. This is achieved in two passes. In the first pass edge overlaps are removed by first introducing a dummy node for each bend point of each connector. As a result some pairs of nodes may now be connected by multiple edges, and in such cases we simply eliminate all but one of those edges. In the second pass, edge crossings are removed by introducing a dummy node at each crossing. Because all edge segments are vertically or horizontally aligned, crossing detection is an easy special case of the $O(n \log n)$ Bentley-Ottmann algorithm [BO79].

(c) We are now left with graph P, which is a planarisation of the core graph C, and whose set of nodes contains that of C as a subset (Figure 5.4c, page 64). In particular each root node r_i for the trees T_i belongs to P. Since P is planar we may compute its set F of



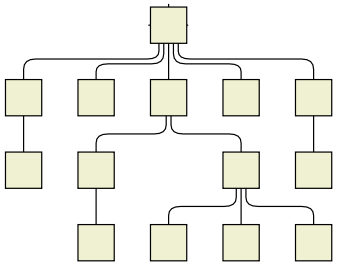


Figure 5.8: This tree has five "c-trees", which are the connected components that remain if the root node is deleted. One of these (the centre one) is unique. The other four pair off into two pairs by isomorphism. The symmetric tree layout algorithm of Manning and Atallah tries to maximise symmetry by putting the unique c-tree in the centre while placing those that pair off in corresponding positions on either side.

faces. Step 3c of the layout process now determines how to reattach the trees laid out in Step 3a to the root nodes r_i in P. This means choosing for each tree T_i a *tree placement* (f, d_p, d_g, b), where $f \in F$ is a face to which the root node r_i belongs, d_p and d_g are the *placement direction* and *growth direction*, and b is the *flip bit*.

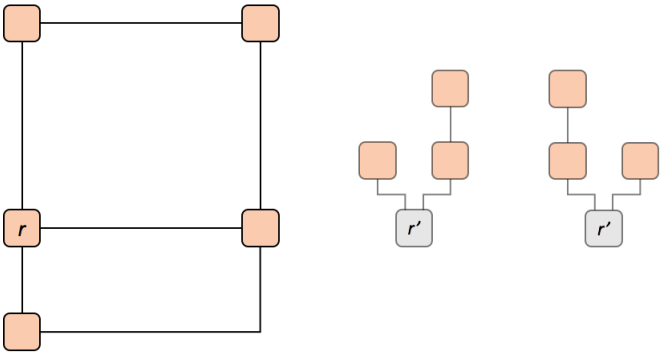
The placement direction d_p is one of the eight compass directions including the four cardinal directions discussed already, as well as the four *ordinal* directions, SE, SW, NW, and NE. The growth direction d_g is one of the four cardinal compass directions. This is because while growth is possible only in the cardinal directions, we include ordinal placement directions d_p in order to allow more attachment choices. If d_p is cardinal then d_g must equal d_p; if d_p is ordinal then d_g must be one of the two cardinal components of d_p. The flip bit b is a Boolean saying whether the tree should be flipped over the axis of its growth direction d_g. See Figure 5.9.

Note that some placements may require that a face be *expanded* in order to make room for the tree, such as any placement in the SE direction in Figure 5.9. Such an expansion can be achieved by the application of suitable separation constraints, which we compute by the process illustrated in Figures 5.11, 5.12, 5.13, and 5.14.

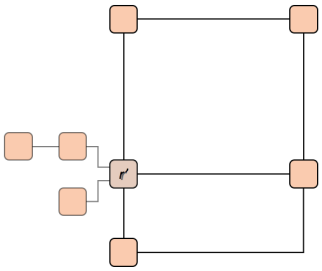
Because of the need to expand, we make tree placement a greedy process in which the trees are considered in descending order of the perimeter of their bounding box. The idea is that this perimeter gives a rough indication of how disruptive any face expansions are likely to be, and that we are better off making the biggest disruptions early in the process, rather than late. If we make them early, the first few expansions are apt to provide sufficient room in neighbouring faces (due to alignments in the core) so that subsequent tree placements won't require any further expansion and can be made more quickly. On the other hand if we make the biggest expansions late in the process, then earlier computation of small expansions may prove to have been a waste of time. See Figure 5.10.

While we recognise that some expansion will probably be necessary, we want to keep it to a minimum. Each expansion means an increase in stress. Therefore as we examine each possible placement (f, d_p, d_g, b) we attempt to make a choice that will minimise the increase in stress. Since the expansion can usually be performed in several different ways we evaluate each option by computing the necessary separation constraints, projecting, measuring the change in the stress of the graph, and backtracking. In this way tree placement is guided by **P1**.

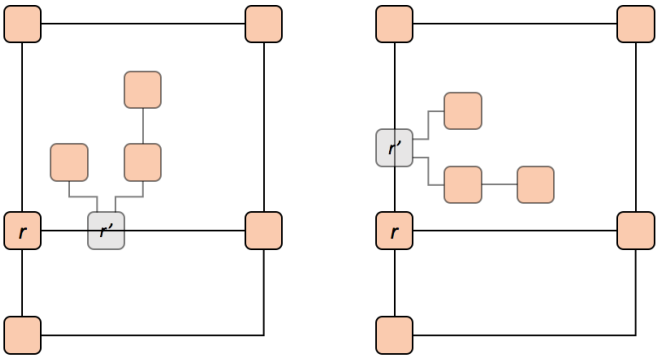
Besides consideration of stress, however, our choice of tree placements is governed by two configurable flags, one saying whether we will favour cardinal placement directions d_p, and one saying whether we will favour placement in the external face f_{ext}, i.e. the unique unbounded face. Under our default configuration both flags are set to true, with cardinal placement taking highest precedence, followed by external placement, and with stress minimisation being considered last.



(a) Consider a simple planarised core graph (left) with a root node r, the tree to be placed at r (centre) and the flipped version of the tree (right). The flip bit b controls whether the tree or its flipped version will be used. Node r belongs to three faces (including the external one), making three placement directions d_p possible: NE, SE, and WEST.

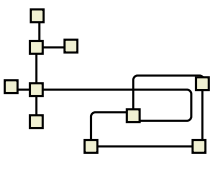


(b) If WEST placement direction is chosen, then the tree is rotated so that its growth direction d_g is also WEST. The tree's copy r' of the root node r is placed directly over the true root node r.



(c) If NE placement direction is chosen, then the tree can be rotated into two possible growth directions: NORTH, and EAST. In both cases, since we have chosen an ordinal placement direction the root copy r' is positioned beside the true root r, not directly on top of it. Accordingly, the connectors between r and its children in the tree will be rerouted if either of these placements is chosen. In all cases the root copy r' is deleted in the final layout.

Figure 5.9: Tree placement and growth directions



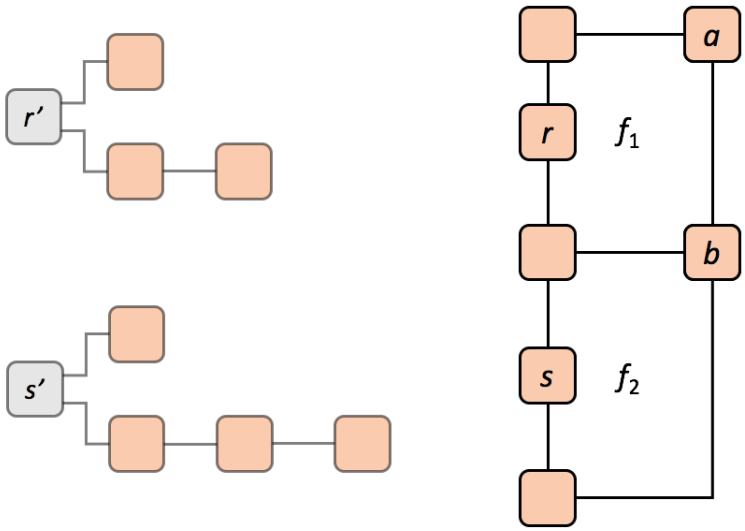


Figure 5.10: Suppose we are going to place the trees rooted at nodes r and s into faces f_1 and f_2 respectively, both with EAST growth direction. In a case like this our strategy to place larger trees first pays off. For if we place the tree at node s first then, due to the alignment constraint on nodes a and b, our expansion of face f_2 results in a simultaneous expansion of face f_1 providing adequate room for the tree rooted at r. If instead we had placed the tree at r first then we would still have to spend time figuring out how to expand face f_2 for the tree at s.

Favouring of cardinal placement is motivated by the desire for symmetry, and to make the routing of connectors from the root node to the tree nodes of the first rank easier (i.e. so that the connectors are shorter overall, and so that there is more room in which to route them; see Figure 5.9c). Favouring of placement in the external face is motivated by findings from our formative study. Thus tree placement is also guided by **P2**.

In Figure 5.4d two trees have been placed already, and their bounding boxes inserted into the graph temporarily as place holders. The tree rooted at the dark grey node is to be placed next, and three placement directions d_p are possible: SW or E into interior faces, or NW into the exterior face. Under our default configuration the E placement will be chosen, since it is the only cardinal direction available.

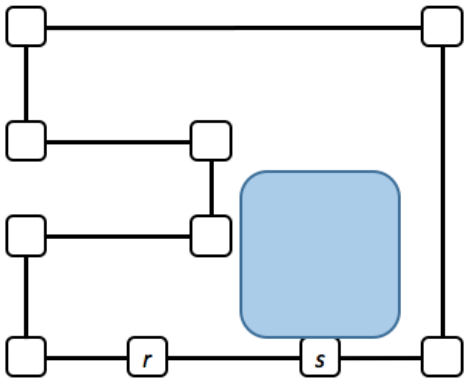
(d) During the tree placement process the stress of the graph will in general climb as we PROJECT onto each new set of face-expansion constraints. Therefore after all trees have been placed (Figure 5.4e, page 64) we perform a DESCEND (Step 3d) to dissipate the accumulated stress as much as possible.

5.3.4 Opportunistic improvement

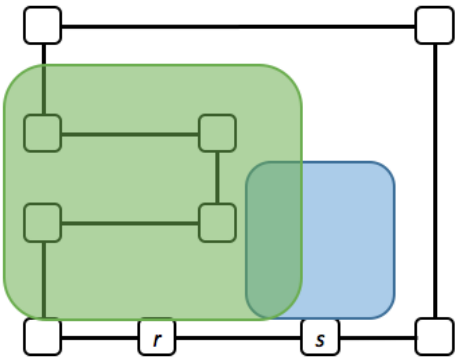
The final stage in the algorithm is designed to tweak the layout, ideally leaving no obvious small changes that would improve it. This consists primarily of alignment and distribution. Afterwards we add a couple of extra refinements.

(a) We begin in Step 4a with a process that searches for nearby pairs of nodes that are almost (but not quite) aligned, and it applies constraints to align them. For example in Figure 5.4e (page 64) the dark grey nodes will be aligned, as in Figure 5.4f.

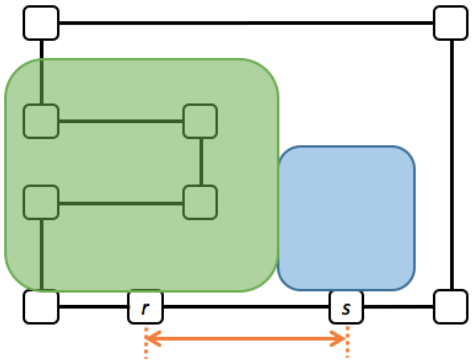
(b) After addressing alignments we consider the even distribution of nodes. This time instead of making local refinements we rely on a global stress minimisation step (Step 4b) in which we modify the stress function to include only those terms corresponding to neighbouring nodes. We refer to this as *neighbour stress*, and this step tends to space sequences of adjacent nodes more evenly by ignoring the effects of long-range forces.



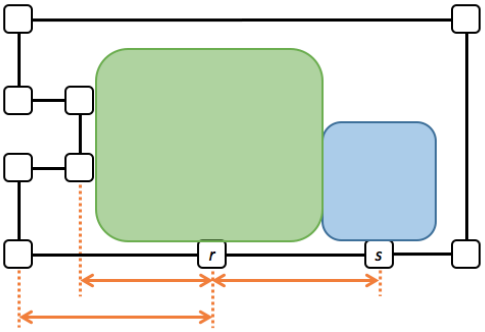
(a) Suppose a tree rooted at r is to be placed into this face. A tree rooted at s has already been placed, and is represented by a box making room for the tree plus padding.



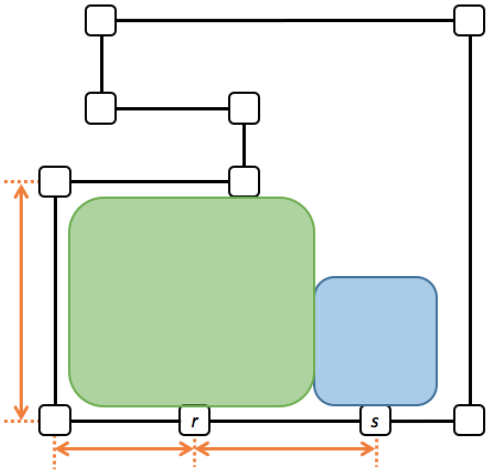
(b) The tree to be placed requires the space represented by the box attached to its root node r.



(c) Using separation constraints between the nodes belonging to the face, there are various ways in which we can make room for the new tree, but all of them require that we push r and s apart.

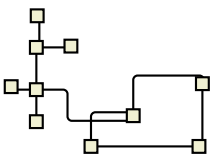


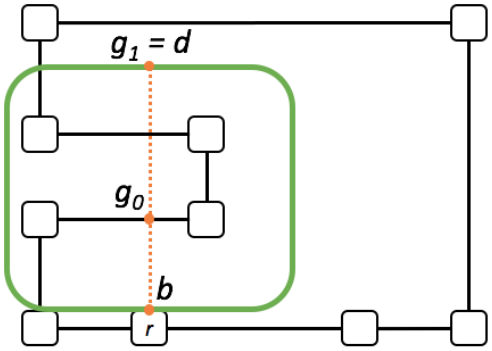
(d) Among all the ways we could proceed to expand the face to make room for the new tree, here is one. In this case we only needed to work in the x dimension.



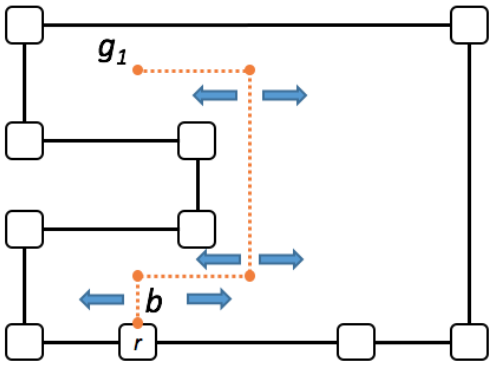
(e) Here is a different way to make room. In this case we needed to work in both the x and y dimensions. There remain several more ways that are not illustrated here. We need some way to choose the best among the available expansion options, where "best" means smallest increase in stress.

Figure 5.11: Expanding a face to make room for a tree placement

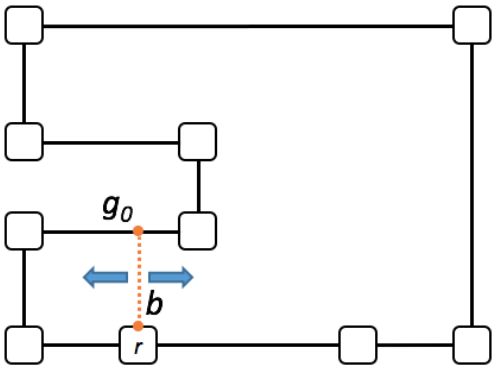




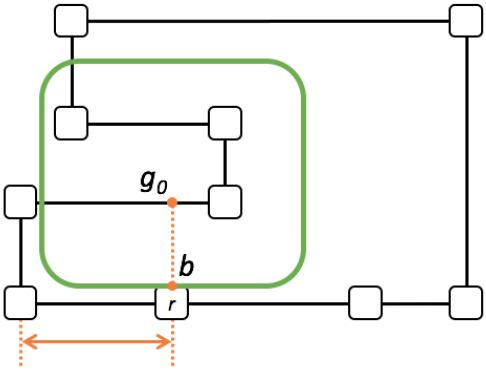
(a) When deciding how to expand a face we may or may not need to operate in both dimensions x and y; our first choice is which dimension we will operate in *first*. Suppose we choose to work in the x dimension, meaning that we're going to push nodes left and right. In that case we begin by drawing a vertical line from the *base point* b where the tree box meets its root node, to the *distal point* d on the opposite side of the tree box. On this line we mark all the points at which going any further would cause us to leave either the tree box or the face. We call these our *goal points* g_0, g_1.



(b) Each goal point provides us with one option for expanding the face in the chosen dimension. Let's begin with g_1. We use the orthogonal connector routing algorithm of Wybrow et al. [WMS10] with each connector segment set as an obstacle. This routes a path from b to g_1 without going outside the face. The vertical segments of this path tell us how to expand: anything lying to the left of a vertical segment gets pushed to the left, while anything to the right gets pushed to the right. Of course, only objects overlapping the desired tree box get pushed at all. The result is shown in Figure 5.11d.

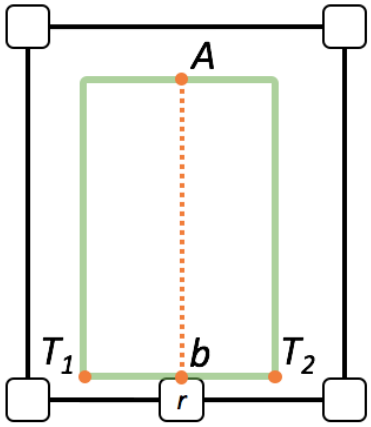


(c) Alternatively, we can try the same thing with goal point g_0. This time the orthogonal routing algorithm connects b and g_0 with a simple straight line, but the rule is the same: anything to the left goes to the left, while anything to the right goes to the right.

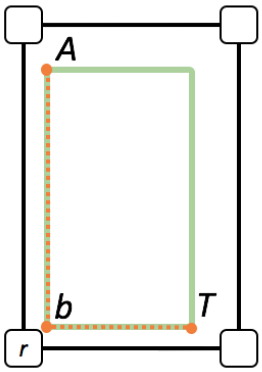


(d) This time working in the x-dimension is not enough, and we have to now expand in the y-dimension as well. The result is shown in Figure 5.11e.

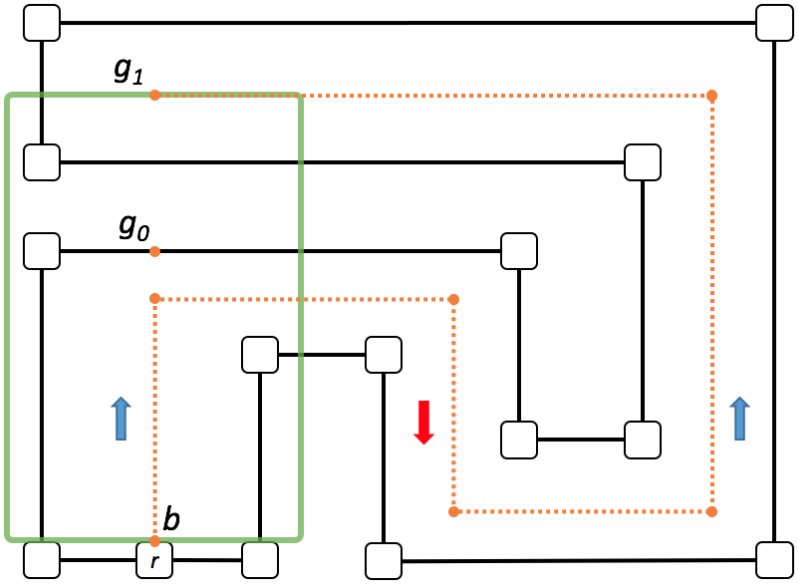
Figure 5.12: Method for determining the expansion options



(a) The vertical line segment we drew in Figure 5.12a is what we call a *goal segment*. It runs from the base point b to *distal point* d. In total, counting both the x- and y-dimensions, a tree with cardinal placement direction d_p has three goal segments: one *axial* (in the placement dimension) and two *transverse* (in the other dimension). We find the axial distal point A by moving in the growth direction d_g. We find the transverse distal points T_1, T_2 by moving in the two perpendicular directions from b, and again stopping at the boundary of the tree box.

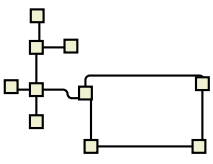


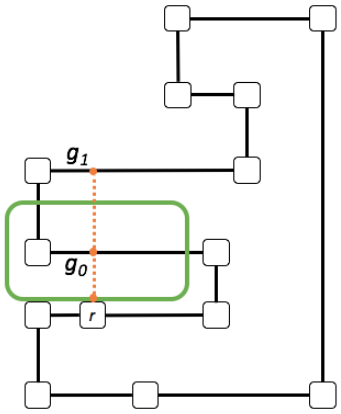
(b) A tree with ordinal placement direction d_p has just two goal segments: one *axial* and one *transverse*, since this time the base point lies at a corner of the tree box.



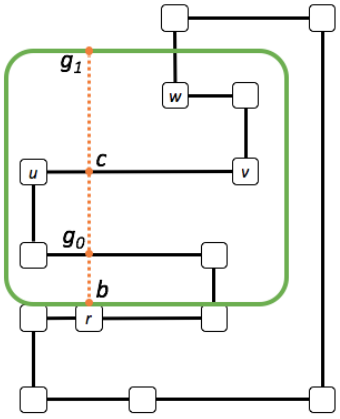
(c) We refer to a case as "pathological" if the route from b to the goal point doubles back, i.e. if it ever goes in the direction opposite the direction from b to the distal point d. In such a case the desired operation of pushing nodes left and right (or up and down) according to the routing won't work, so we simply abandon such cases and opt for a different way of expanding the face. The nearest goal point g_0 is always an option, since by definition we have not yet crossed outside the face, and therefore the route is always a straight line. It is sometimes possible to correct pathological cases by working in alternating dimensions (e.g. in this figure we could start by pushing obstacles up and down according to the horizontal segments in the route from b to g_1) but we decided to abandon this approach in the interest of speed and minimal disruption of the existing position.

Figure 5.13: Goal segments and pathological cases

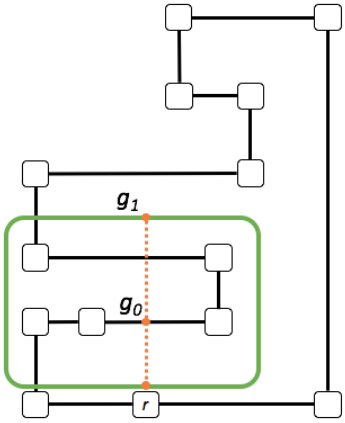




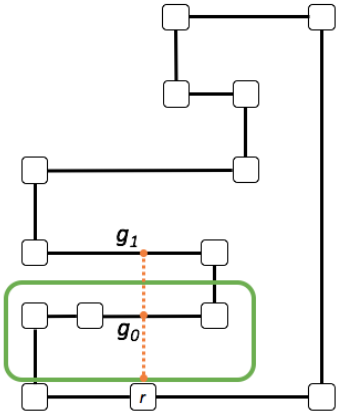
(a) External face, odd number of crossings. Why must g_1 lie beyond the distal point of the goal segment? Because the routing from the base point to g_1 must be contained in the external face.



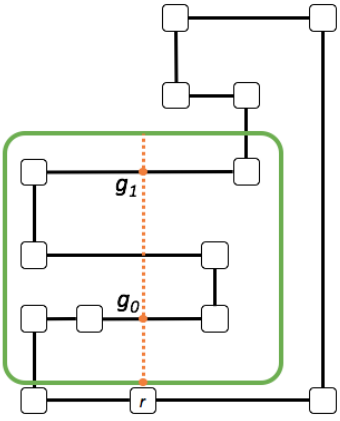
(b) External face, even number of crossings. Why must g_1 lie beyond point c, when the face has already ended? The problem is that if we merely routed from b to c, that would tell us to push node u to the right, and due to non-overlap that would also push node v to the right, all of which is good, but we would fail to push node w to the right.



(c) Internal face, even number of crossings



(d) Internal face, odd number of crossings, final crossing $e = g_1$



(e) Internal face, odd number of crossings, no final crossing point e

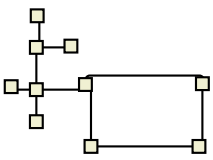
Figure 5.14: Locating the goal points for a given goal segment is actually not quite as simple as was suggested in Figure 5.12a. Here is the procedure: Compute the list $L = \langle c_1, c_2, \ldots, c_k \rangle$ of all points where the goal segment crosses the boundary of the face. (We can have $k = 0$, in which case L is the empty list.) If d is the distal end point of the goal segment (i.e. the end opposite the base point b), add d to the list L. Finally, the ray extending from b through d and onward to infinity may or may not cross the boundary of the face again, beyond d; if it does, append the first such crossing point e to the end of the list L. The list L now looks either like $\langle c_1, c_2, \ldots, c_k, d \rangle$ or like $\langle c_1, c_2, \ldots, c_k, d, e \rangle$. In either case, rewrite the list as $L = \langle p_0, p_1, p_2, \ldots, p_n \rangle$. Then the goal points are those with even index in this listing, i.e. we have the goal points $g_i = p_{2i}$ for $0 \leq i \leq \lfloor n/2 \rfloor$. To illustrate why this works, there are five cases to consider, determined by the following questions: (1) is the face an internal face or the external face; (2) is k even or odd; (3) if internal and odd, then is there a final crossing point e (if external and odd then there *must* be a final crossing point, since all interior faces are bounded).

(c) Next, with a view to the aspect ratio of most existing desktop display devices, we rotate the layout by ninety degrees if its width is less than its height. To make the rotation direction deterministic we prefer that a majority of trees wind up with SOUTH growth direction after the rotation, rather than NORTH (motivated by the most popular layout of Graph 3 from the study). Since we must not rotate the nodes themselves (they may have labels), rotation means that the node positions (but not dimensions) as well as the constraints holding amongst them, are rotated, and after this we apply another DESCEND with neighbour stress.

(d) Finally we remove dummy nodes in passing from the planarised graph back to the original, and perform a final orthogonal routing of the edges. In this step we ensure that edge routes pass through any bend points deliberately created in the chain configuration step, but otherwise take this as a final chance for edge routes to be optimised relative to node positions which in general may have changed since the previous routing.

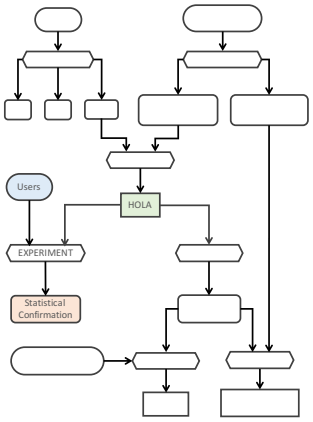
5.4 Conclusions

In this chapter we designed a new orthogonal layout algorithm, HOLA, informed by the user study of Chapter 4. It is designed both to create the kinds of perceptual organisation that people create when working by hand, and to trade off aesthetic concerns as people do. While based on the idea of the ACA algorithm, HOLA generalises that approach in several ways, including decomposing and reassembling the graph, and making more complex constraint choices. Examples of layouts created by HOLA will be presented in the next chapter, where we compare the new algorithm with the current state of the art orthogonal layout procedure (Topology-Shape-Metrics) and with hand-drawn layouts.



Chapter 6

Evaluation of Automatic Layout



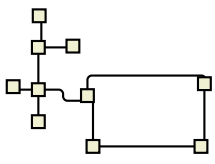
In order to evaluate the human-like orthogonal layout algorithm HOLA developed in the last chapter we performed a second user study, the normative study in our human-centred methodology. This extends the formative study of Chapter 4 in three ways.

First, from the previous study we have eight user-generated layouts that were most highly ranked in the tournament. We can now compare these with two styles of automatic layout: the new HOLA algorithm and the state-of-the-art yFiles implementation of GIOTTO-Kandinsky. Since the best manual layouts have already been shown to be (on average) preferred to yFiles, we expect them to again be preferred by participants in a direct three-way comparison; however, since HOLA is designed to be "human-like" in its approach to layout, we hypothesise

- (**H1**) that HOLA's output will be preferred on average over yFiles, and
- (**H2**) that HOLA will perform comparably to human layout on the corpus of small graphs.

Second, the correlations between layout attributes and preference revealed by our first study (results **R1-9** from Section 4.3.4) were observed in small graphs, limited by the size of graph that could reasonably be manually arranged by our participants. Since both HOLA and yFiles are completely automated we can now directly compare the "human-like" layout approach with the existing layout algorithm on much larger graphs. We hypothesise

- (**H3**) that HOLA layouts will be preferred over yFiles layouts by most participants on larger graphs.



Third, these larger and more complex graphs make readability much more challenging, and so for these graphs we can conduct non-trivial readability tests. We examine user performance on two standard tasks: finding a shortest path between two nodes and finding the neighbours of a node. We hypothesise

- (**H4**) that participants will have greater speed and accuracy in completing these tasks with HOLA layouts compared to yFiles layouts.

The design of the experiment is described in Section 6.1, the four Hypotheses **H1-4** are confirmed in Section 6.2, we consider the efficiency of HOLA in Section 6.3, and finally we examine possibilities for improvement in Section 6.4.

6.1 Design

Apparatus & Materials: The second user study used two graph corpora. The first contained three layouts of each of the eight graphs from the original study: layouts computed with HOLA and with yFiles (default settings, as discussed in Chapter 4) and the highest-ranked human layout of each graph. The second corpus contained HOLA and yFiles layouts for six larger graphs. We chose one SBGN graph (the Glycolysis-Gluconeogenesis pathway), one metro map graph (Sydney), and we generated four random graphs, including one small (60 nodes, 65 edges), one large (120 nodes, 126 edges), and two medium-sized graphs of different densities, (90 nodes, 100 edges) and (90 nodes, 110 edges). See Figures 6.5, 6.6, 6.7, 6.8, 6.9, and 6.10. We could not include human layouts for these larger graphs because manual layout would have taken prohibitively long. It may be noted that none of our large graphs has very high density, the largest being 1.22 edges per node. This fits with our focus on sparse networks as noted in Section 1.5.

Participants: The study was advertised on University newsletter *Monash Memo* as well as within our Faculty. In total 89 participants started the survey and completed Part 1, while 84 completed Part 2, and 83 continued through Parts 3 and 4. Due to a display error in the first part, we had to reject the first 13 participants' answers on three of the eight graphs. A $50 gift card was offered as incentive, and it was explained that the winner would be the participant whose accuracy and speed were the best in Parts 2 and 3, and whose choices were the closest to the aggregate choices in Parts 1 and 4 and thus, in order to win, your best strategy was to choose the layouts which you thought would be preferred by most other people.

Procedure: The survey was conducted online. It had four parts.

Part 1: (Figure 6.1) Participants ranked the three layouts in the first corpus. The order of the graphs was randomised, as was the order of the three layouts on each page. Participants were asked to drag gold, silver, and bronze medal icons onto the three layouts. (The icons also said '1st', '2nd', and '3rd'.)

Part 2: (Figure 6.2) Participants were asked to find a shortest path between two nodes in the six larger graphs. The two nodes were chosen systematically: one was a highest-degree node and the other was a node either four or five links away. Nodes meeting these criteria were chosen randomly during the design of the experiment, but once chosen were held fixed throughout the experiment. Users were shown each graph in two layouts, HOLA and yFiles, with the two chosen nodes coloured red, and they were asked to click on the nodes of a shortest path between them, turning the intermediate nodes green. They were first given a training task of the same kind, in which they could not advance until they had correctly identified a shortest path.

Part 3: (Figure 6.3) This was similar to Part 2 but with a single red node and the task being to click on all neighbours of that node. There was again a training task. The

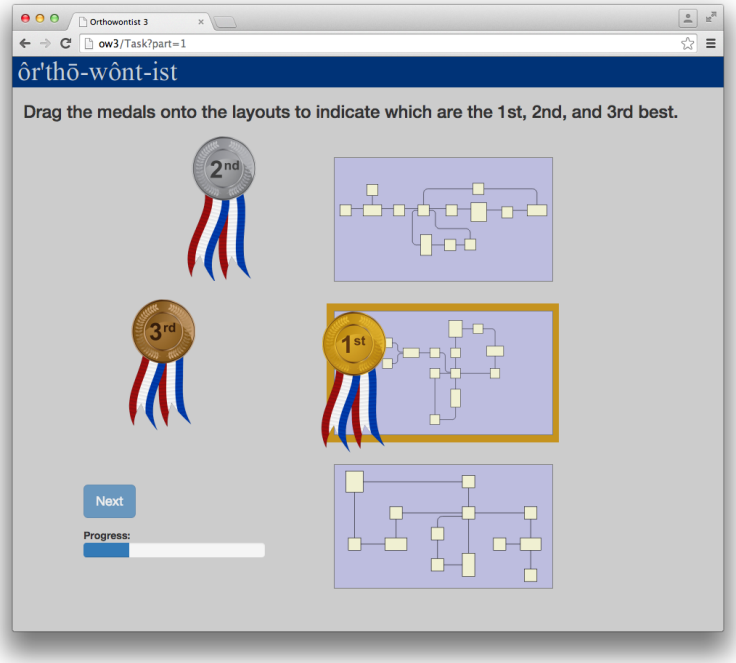


Figure 6.1: In Part 1 of the study, participants ranked the HOLA, yFiles, and best human layout of each of the eight small graphs from the first study.

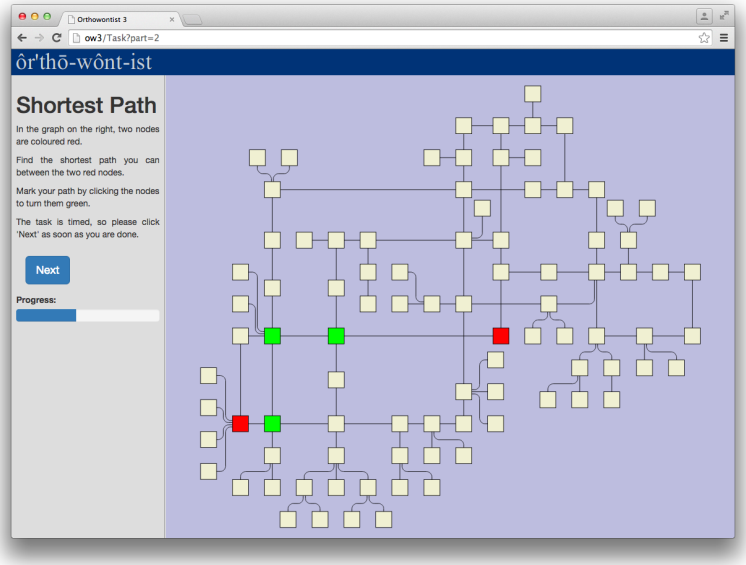
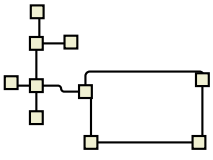


Figure 6.2: Part 2 of the study, finding shortest paths in the large graphs



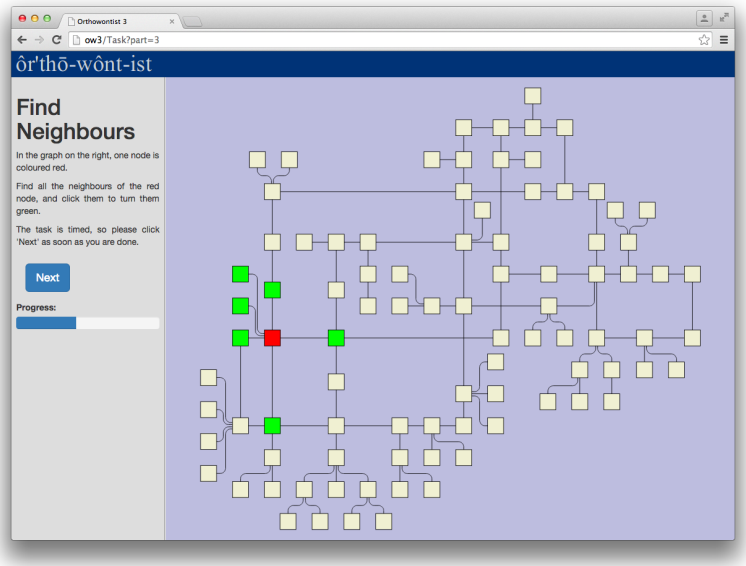


Figure 6.3: Part 3 of the study, finding neighbours in the large graphs

red node was chosen to be a node of maximal degree but different from both of the red nodes chosen for Part 2, to avoid familiarity effects.

Part 4: (Figure 6.4) The large graphs were shown in the two layouts HOLA and yFiles side by side, and participants were asked to say which layout was better, and to briefly explain why in a text box. Again the order of the graphs was randomised, as was the order of the two layouts on each page.

6.2 Results & Discussion

Part 1: Rankings assigned with the gold, silver, and bronze medal icons in the online study were mapped onto the numbers 1, 2, 3 respectively, so that 1 was the best possible rank and 3 the worst. The rankings were analysed using Friedman's test. The aggregate mean rank across all graphs was 1.75 for Human, 1.80 for HOLA, and 2.46 for yFiles, and the null hypothesis was rejected ($p <$ 1.0e-14).

Post-hoc pairwise Nemenyi tests showed that both HOLA and Human layout were preferred over yFiles and hence are an improvement over the best existing orthogonal layout algorithm ($p <$ 1.0e-11 and $p <$ 1.0e-10 resp.). Thus, we can accept Hypothesis **H1** and conclude that HOLA outperforms yFiles in terms of user preference on the small graphs from our study.

On average over all the small graphs, neither HOLA nor Human layout was clearly preferred one over the other, as hoped in our intent to match human layout quality. The significant pairwise preferences for the eight individual graphs amongst the three layout methods are shown in Fig. 6.11. Thus we can accept Hypothesis **H2** and conclude that HOLA performs comparably to hand-made layout on the eight small networks considered.

User preferences on the eight graphs both reconfirmed our findings from the formative study, and provided new insights. See Figures 6.12 and 6.13.

Graph 2 provided new insight. HOLA produced almost exactly the same layout as the top-ranked human, only slightly more compact. The less-compact human layout was preferred significantly. This suggests that it may be possible to overdo compaction. Or perhaps it could simply be that the human layout was preferred because the separations between neighbouring nodes are more nearly uniform. We might also wonder whether this

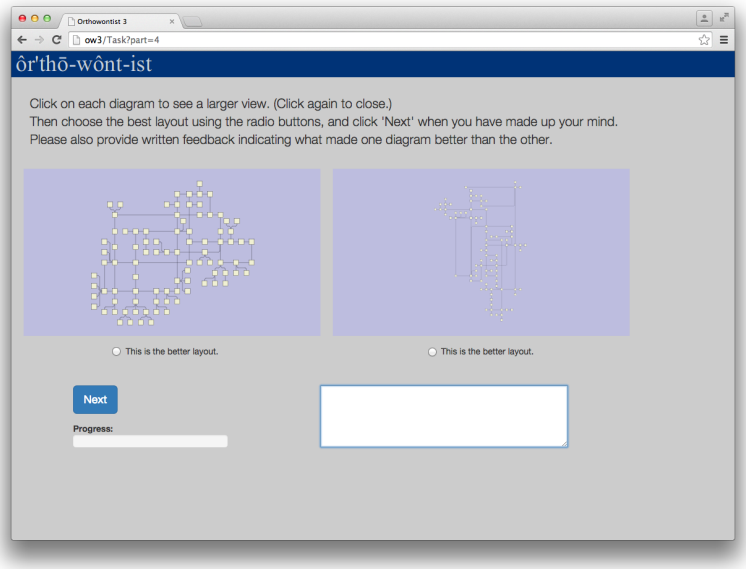


Figure 6.4: Part 4 of the study, user preference

result had something to do with "proportion" but, if so, it does *not* seem to have been anything related to the *golden ratio* ϕ. See Figure 6.14. Perhaps this can be added to the results of Markowsky [Mar92], who claims that the supposed aesthetic optimality of rectangles with aspect ratio ϕ is merely a popular misconception.

On Graph 3 HOLA again mimicked the best human layout, but it was preferred significantly. This is likely attributable to the more precise and even spacing in HOLA's version. This tends to confirm the importance of even distribution.

Preference for HOLA's layout on Graph 4 tends to confirm that users like the arrangement of subtrees in hierarchical format.

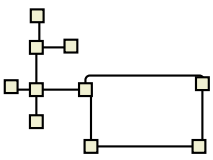
Part 2: This compared performance when finding the shortest path between two nodes on the larger graphs laid out using yFiles and HOLA. Both error rates and timings showed a high degree of leptokurtosis, so Wilcoxon's signed rank test was used. The mean error rates 0.162 for HOLA and 0.548 for yFiles differed significantly ($p <$ 1.0e-13). The mean times 12.27s for HOLA and 29.15s for yFiles also differed significantly ($p <$ 1.0e-14).

Part 3: This compared performance when counting a node's neighbours on the larger graphs laid out using yFiles and HOLA. Again both error rates and timings showed a high degree of leptokurtosis, so Wilcoxon's signed rank test was used. The mean error rates 0.159 for HOLA and 0.349 for yFiles differed significantly ($p <$ 1.0e-08). The mean times 10.10s for HOLA and 12.98s for yFiles also differed significantly ($p <$ 1.0e-11).

The significant results for Parts 2 and 3 together allow us to reject the null-hypothesis for **H4** and accept that overall, participants were significantly faster and more accurate with HOLA layout than yFiles.

Part 4: Finally we compared user preferences on the larger graphs. User rankings were analysed using Wilcoxon's signed rank test. The aggregate mean rank across all graphs was 1.20 for HOLA and 1.80 for yFiles, which differed significantly ($p <$ 1.0e-11).

Among individual graphs only the higher density graph on 90 nodes (Fig. 6.7) showed no significant preference. For all others HOLA was preferred ($p <$ 1.0e-5). This reinforces our decision to target lower density graphs, and shows that HOLA produces better layouts in those cases (**H3**).



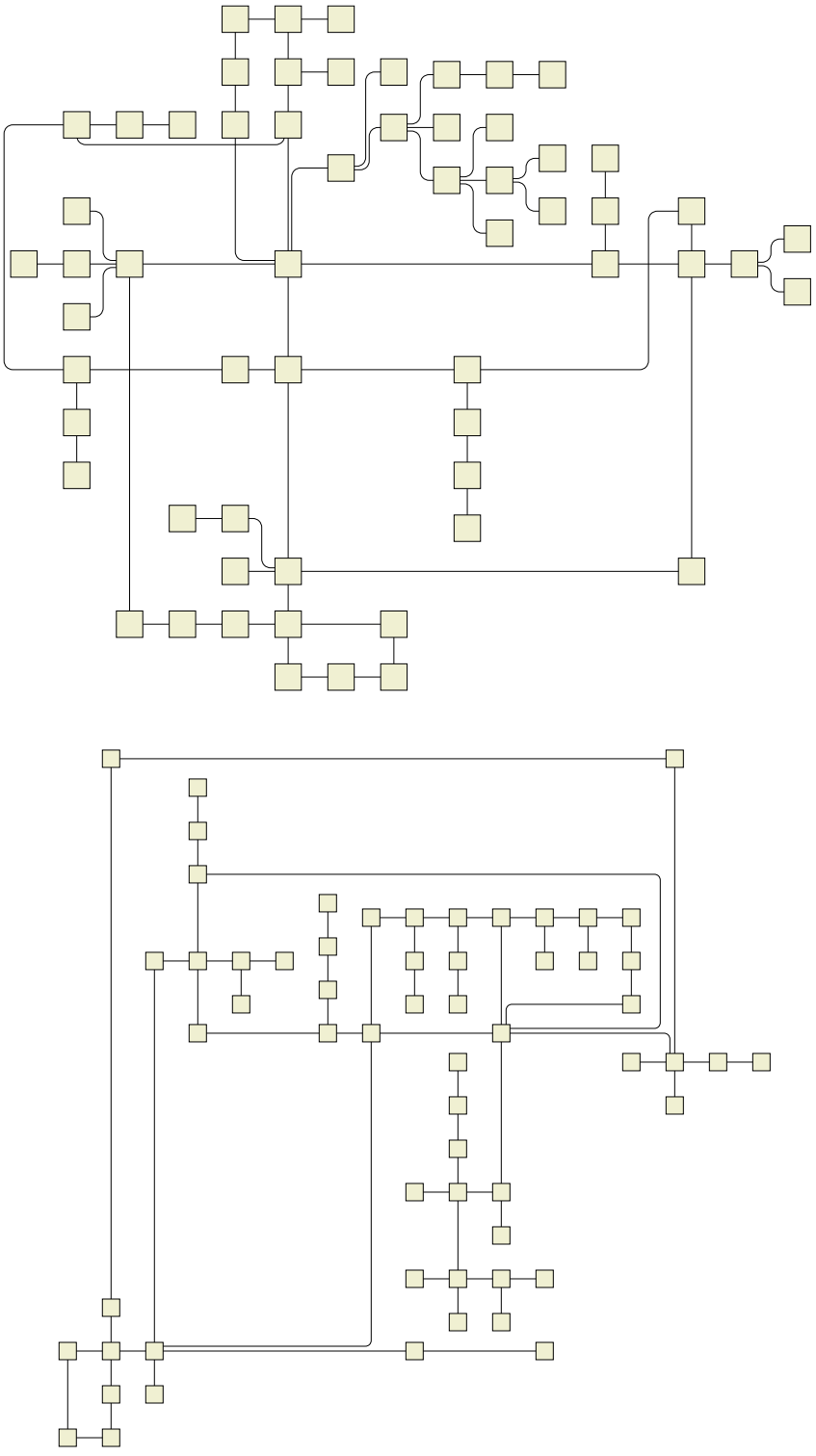


Figure 6.5: HOLA (above) and yFiles (below) layouts of our small graph, with 60 nodes, 65 edges

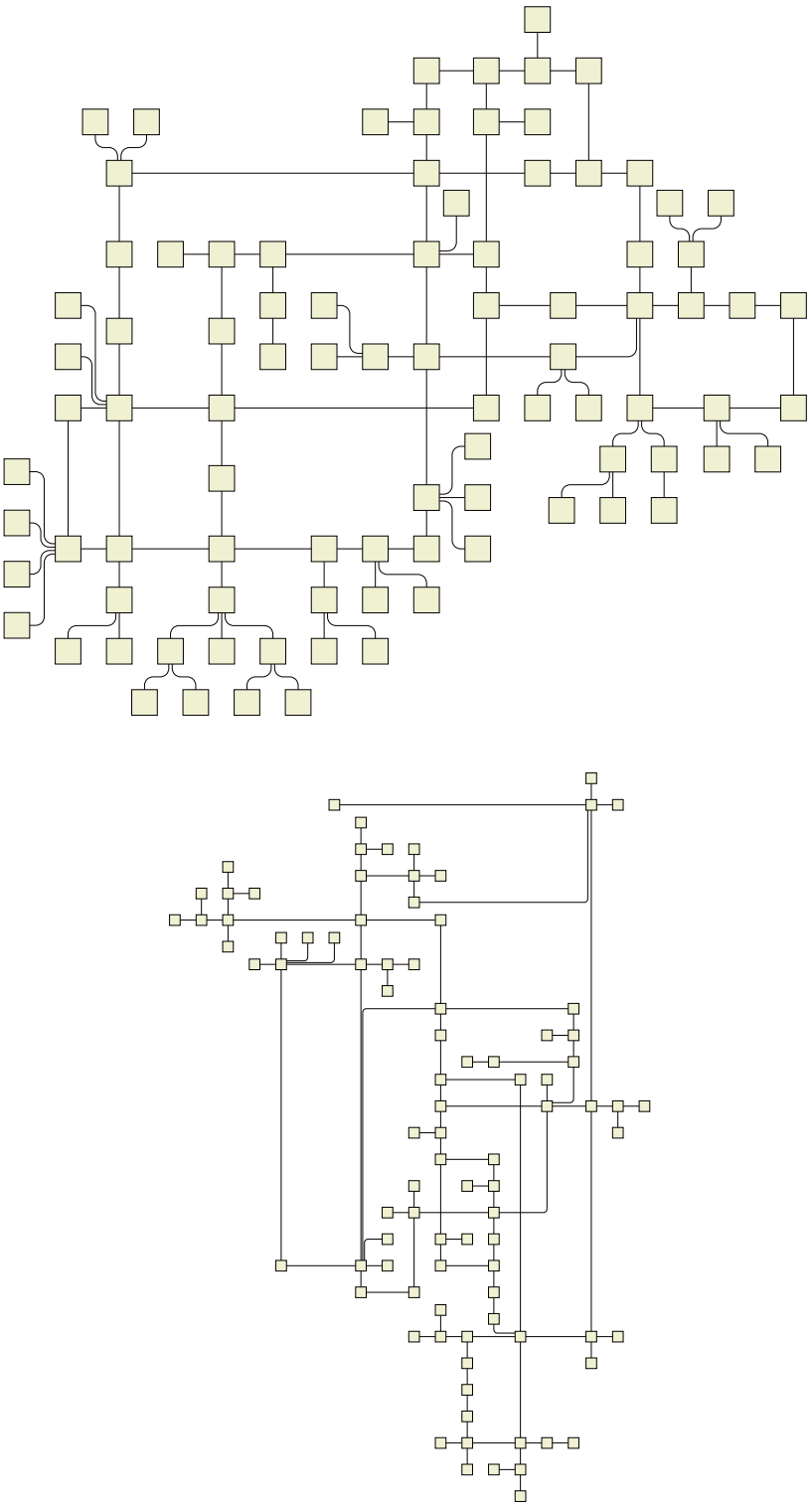
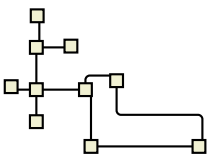


Figure 6.6: HOLA (above) and yFiles (below) layouts of our lower density medium sized graph, with 90 nodes, 100 edges



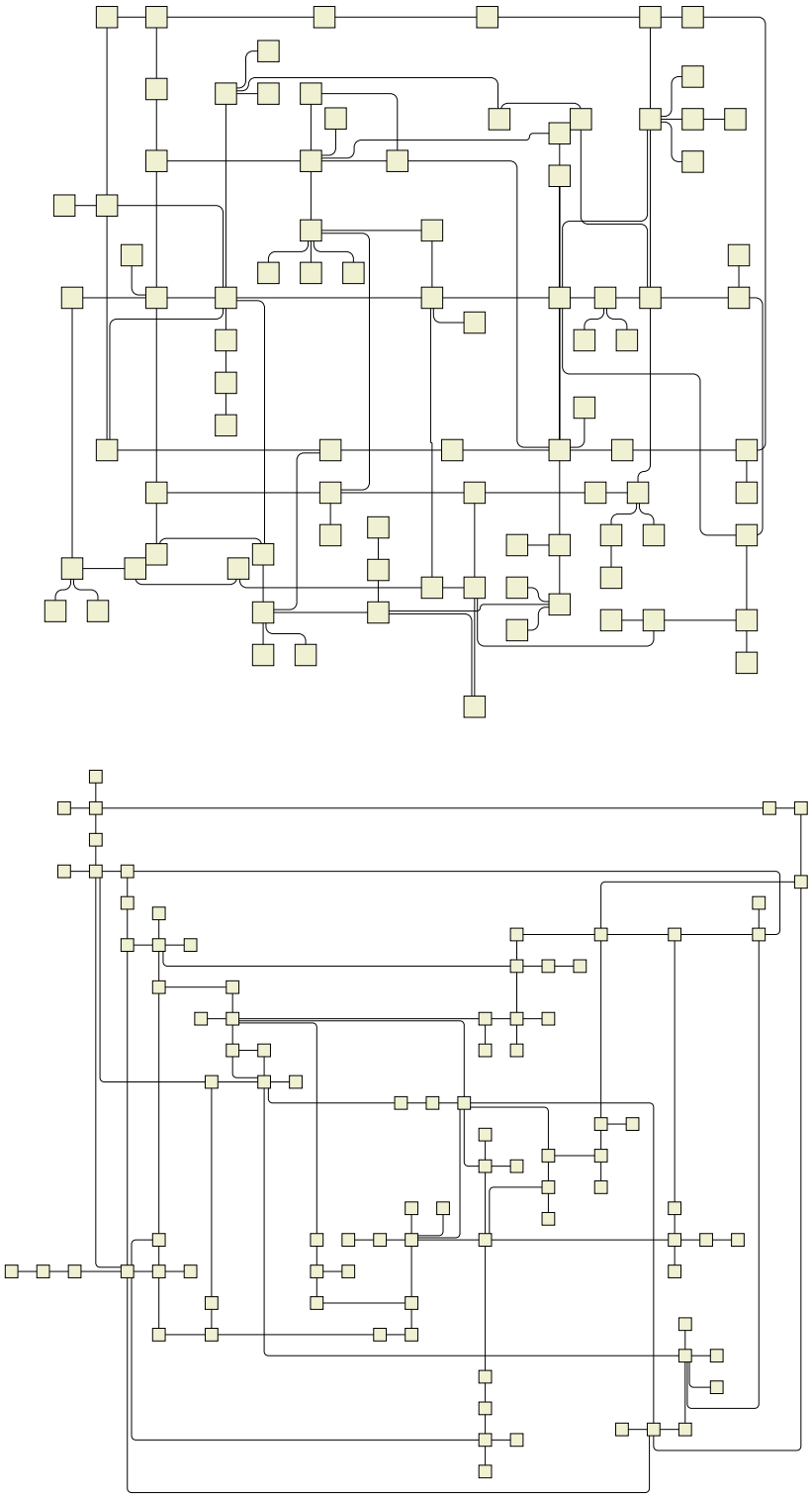


Figure 6.7: HOLA (above) and yFiles (below) layouts of our higher density medium sized graph, with 90 nodes, 110 edges

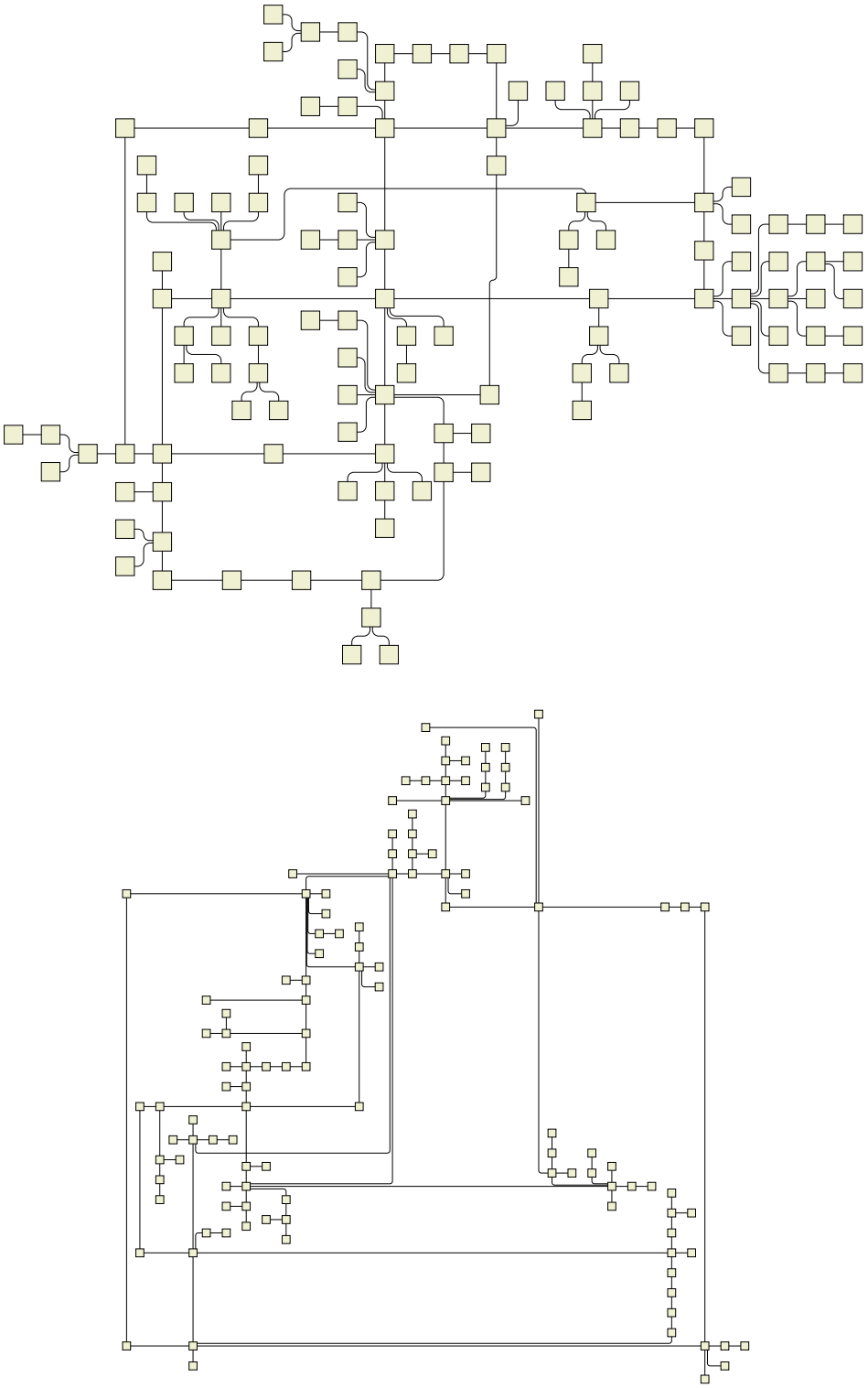
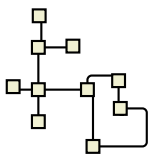


Figure 6.8: HOLA (above) and yFiles (below) layouts of our large graph, with 120 nodes, 126 edges



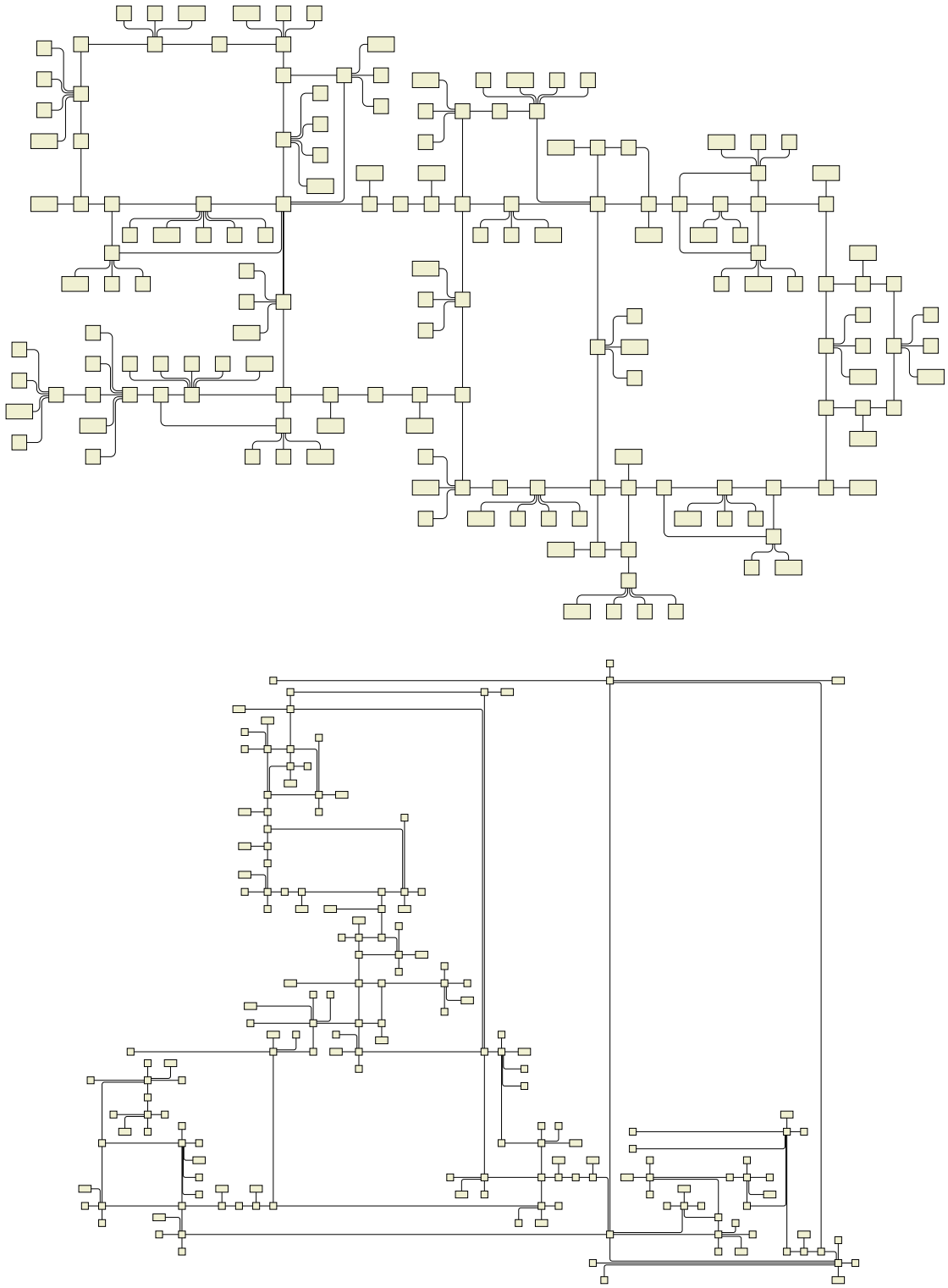


Figure 6.9: HOLA (above) and yFiles (below) layouts of SBGN Glycolysis-Gluconeogenesis pathway network

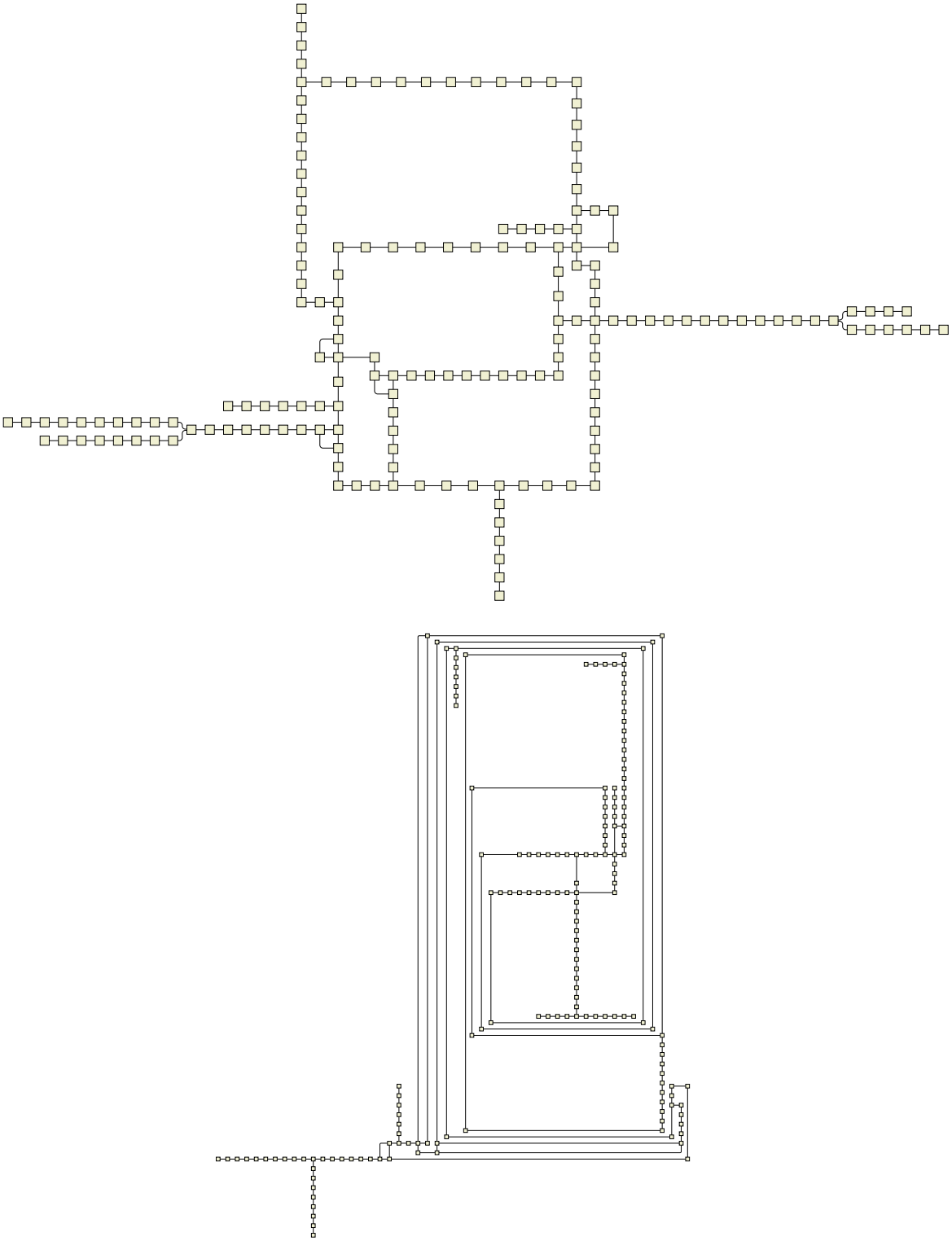


Figure 6.10: HOLA (above) and yFiles (below) layouts of Sydney metro map network

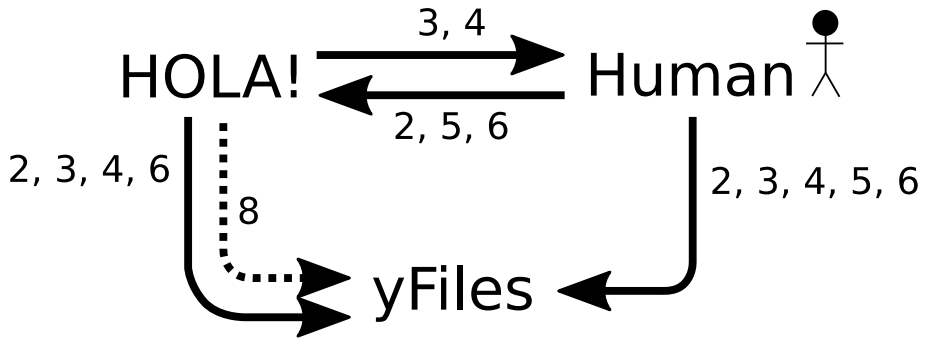


Figure 6.11: Participants' preferences for drawings by human, yFiles or HOLA of graphs from Fig. 4.1. A solid arrow $a \rightarrow b$ indicates a significant preference for condition a over condition b at the $p < 0.01$ confidence level. The dotted arrow indicates a preference for layout of Graph 8 by HOLA over yFiles at the $p < 0.05$ level.

Feedback: After giving their preference for the large graphs, participants left comments in a free-text field. From 83 participants looking at six graphs, 359 comments were collected. There were 65 comments concerning proximity of connected nodes, e.g., *"Connected nodes are generally closer together, making their association more obvioius [sic]."* There were 12 comments mentioning *"structure,"* 8 of them favouring the "structure" of HOLA layout, e.g., *"Clearer to see, more structured layout."* Sixteen comments gave favourable mention to *"trees"* in HOLA, e.g., *"More familiar - like a family tree."* Briefly, other significant terms and the number of times they were mentioned were: *"Compactness"* (13 times), *"Symmetry"* (3 times), *"grid"* (4 times), and *"cross[ings]"* (7 times).

6.3 Algorithm Efficiency

We evaluate the runtime of HOLA on a corpus of random graphs. In choosing the parameters of these graphs we again follow the standard set by Di Battista et al. [DBGL95]. The graphs in their study had between 10 and 100 nodes, and densities of about 1.2 to 1.3. We decided to push these ranges a bit, generating random graphs with between 10 and 170 nodes, and densities from 1.1 to 1.5.

In runtime yFiles does much better than HOLA. On all the graphs in this corpus yFiles layout takes less than 2 seconds, whereas the runtimes for HOLA can be as high as 25 seconds. However, HOLA is fast on smaller graphs. On the eight small graphs from our first study HOLA runtimes ranged from 9 to 97 milliseconds (Figure 6.15).

6.4 Issues

HOLA did well in the evaluation study, but it leaves room for improvement. The main issues are its long runtime compared to yFiles, and its performance on denser networks.

6.4.1 Runtime

What do users want and need?

Users always want a faster algorithm, but there are numerous applications and workflows in which waiting a few seconds to half a minute for a node-link diagram layout is not a problem.

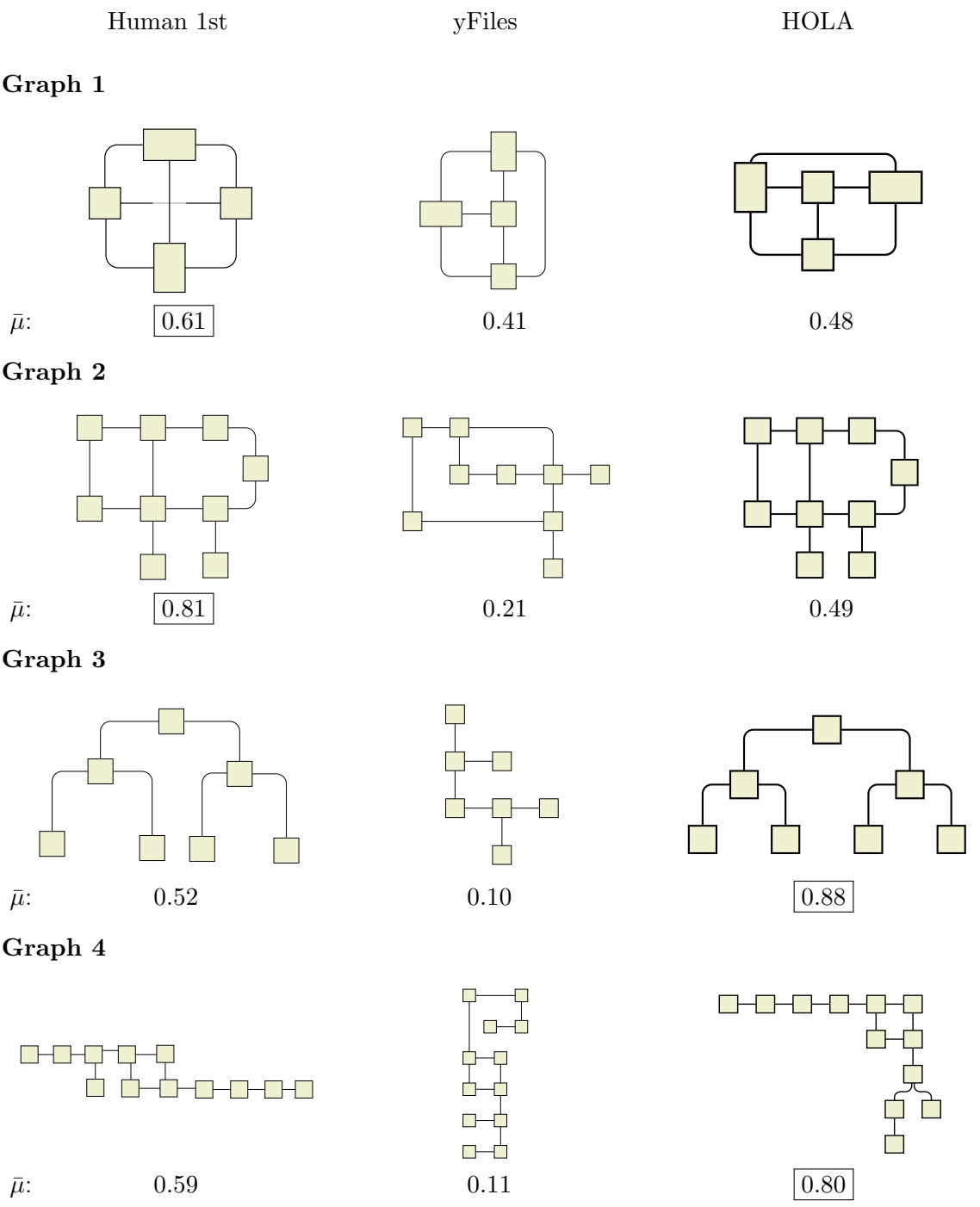
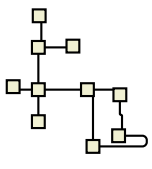


Figure 6.12: HOLA, yFiles, and top-ranked human layout of the first four graphs from the formative study. (See also Figure 6.13.) $\bar{\mu}$ = normalised inverted mean rank. Best possible value is 1, worst possible 0. Means in boxes indicate best actual mean rank in the normative study.



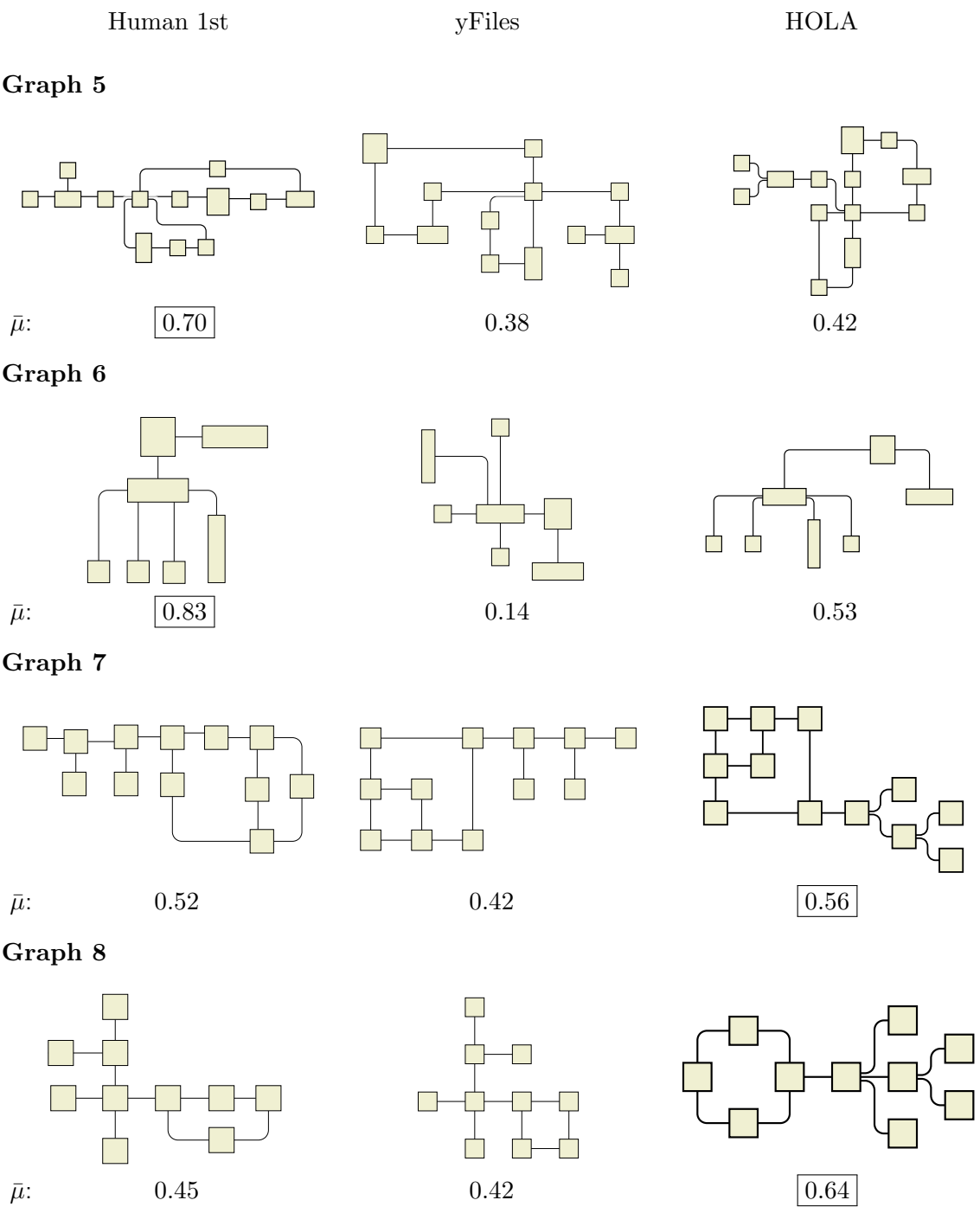


Figure 6.13: HOLA, yFiles, and top-ranked human layout of the last four graphs from the formative study. (See also Figure 6.12.) $\bar{\mu}$ = normalised inverted mean rank. Best possible value is 1, worst possible 0. Means in boxes indicate best actual mean rank in the normative study.

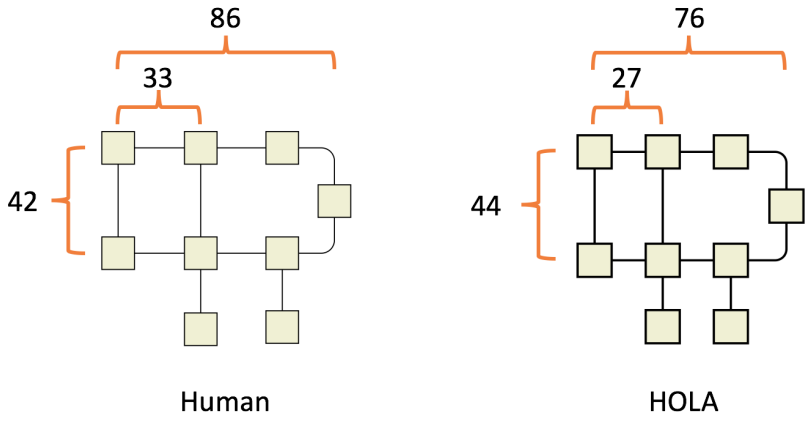


Figure 6.14: Despite the Human layout having been preferred significantly over the HOLA layout on Graph 2, it is the HOLA layout that seems to better exhibit the golden ratio $\phi \approx 1.618$. The aspect ratios of the main rectangles were $86/42 \approx 2.048$ for human and $76/44 \approx 1.727$ for HOLA. The aspect ratios of the left-hand nested rectangles were $42/33 \approx 1.273$ for human and $44/27 \approx 1.630$ for HOLA. This result suggests that preference has more to do with adequate spacing of nodes and avoiding over-compaction, than anything to do with proportions cleaving to the golden ratio. (Note: there is nothing in the design of HOLA that deliberately tries to achieve the golden ratio, and these results were merely by chance.)

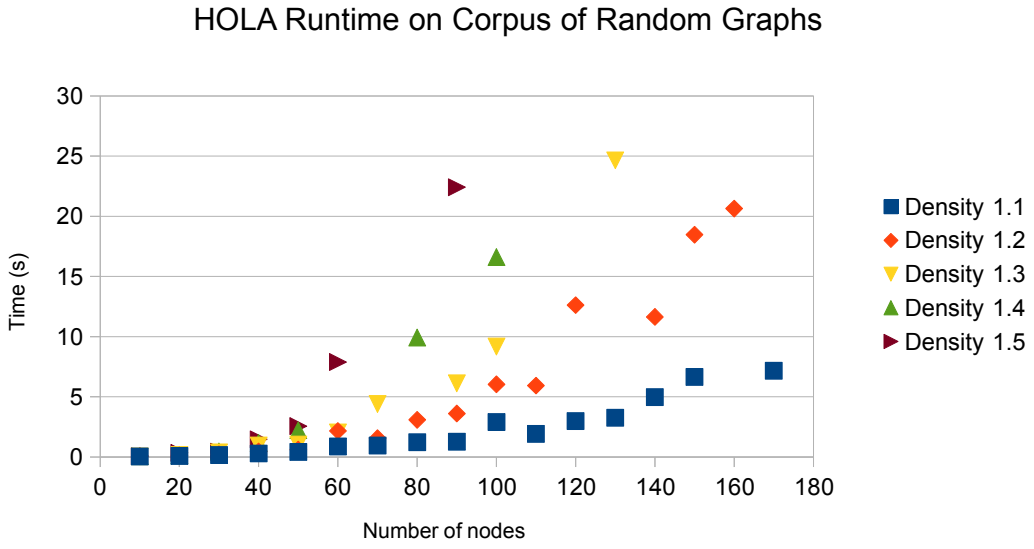
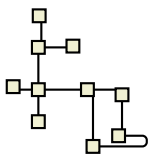


Figure 6.15: HOLA running time on a collection of random graphs with between 10 and 170 nodes, and with densities between 1.1 and 1.5 edges per node. yFiles layout takes less than 2 seconds on each of these graphs. On the small graphs from our first study HOLA runtimes range from 9 to 97 milliseconds.



How might we improve?

We could reasonably hope to speed up HOLA in a few different ways. To begin with, the prototype has been implemented in PYTHON, and might be as much as twice as fast if ported to C++.

Secondly, we could experiment with different approaches to balancing the uses of PROJECT and DESCEND. As was explained in Section 5.3.2, the DESCEND operation is used at key moments during the process. This is so that subsequent constraint choices can be more closely based on the network's natural, stress-minimal position given the constraints already in force. Deciding when to use DESCEND is thus a matter of balancing speed versus quality. It may be possible to tune the procedure more toward speed, if desired.

Another way to trade quality for speed would be to restrict the face expansion operation so that instead of trying all the goal points g_i as defined in Figure 5.14 it would consider only the first point g_0.

6.4.2 Density

What do users want and need?

As noted in Section 1.5, sparse networks are the intended application of HOLA and the SBGN layout algorithm MCGL to be developed in Chapter 8. Still, even the network in Figure 6.7 has a density of 1.222 which is well within the range considered appropriate by Di Battista et al. [DBGL95], and HOLA performed poorly on this one. It is worth considering what could be done to improve.

We can refine the problem by considering the *degree histograms* of the networks on which we have tested HOLA (Figure 6.16), which show how many nodes of each degree were present in each graph. The graph on which HOLA performed most poorly is that whose degree histogram is skewed the farthest toward higher-degree nodes, as is visible in Figure 6.16e. This is no real surprise. Conversely, in our own subjective analysis HOLA's performance on the SBGN graph and the metro map seemed the most impressive, and the histograms for these graphs show extreme preference for degree-1 nodes, and degree-2 nodes, respectively. This suggests that, even more than mere low density, it is the presence of many low-degree nodes that is most important.

Fortunately, the preponderance of degree-1 nodes in the SBGN diagram is not an accident, but is found in SBGN diagrams generally because of manual *cloning* (see below). We take this structural feature into account in Chapter 8 when we develop the MCGL algorithm.

How might we improve?

It is not clear whether HOLA can be made to perform better on higher density graphs, but, insofar as the algorithm is designed to create the same kinds of layouts human beings do when working by hand, perhaps we should not be surprised by this. Recall from Chapter 1 the motivating model of the computer as the ideal assistant. There was never any expectation that the assistant would be able to do things the master could not; only that it would make the job easier.

When the density of the network gets too high human users tend to take steps to lower it, and this may be the most fruitful line of research when it comes to automated techniques as well. Density could be lowered by *cloning* or *vertex splitting*, as a pre-processing step to be applied before HOLA is asked to create a layout.

Cloning is a technique in network layout in which one or more copies or "clones" $u', u'', \ldots$ of a node u are added to the graph, and the edges incident at u are divided

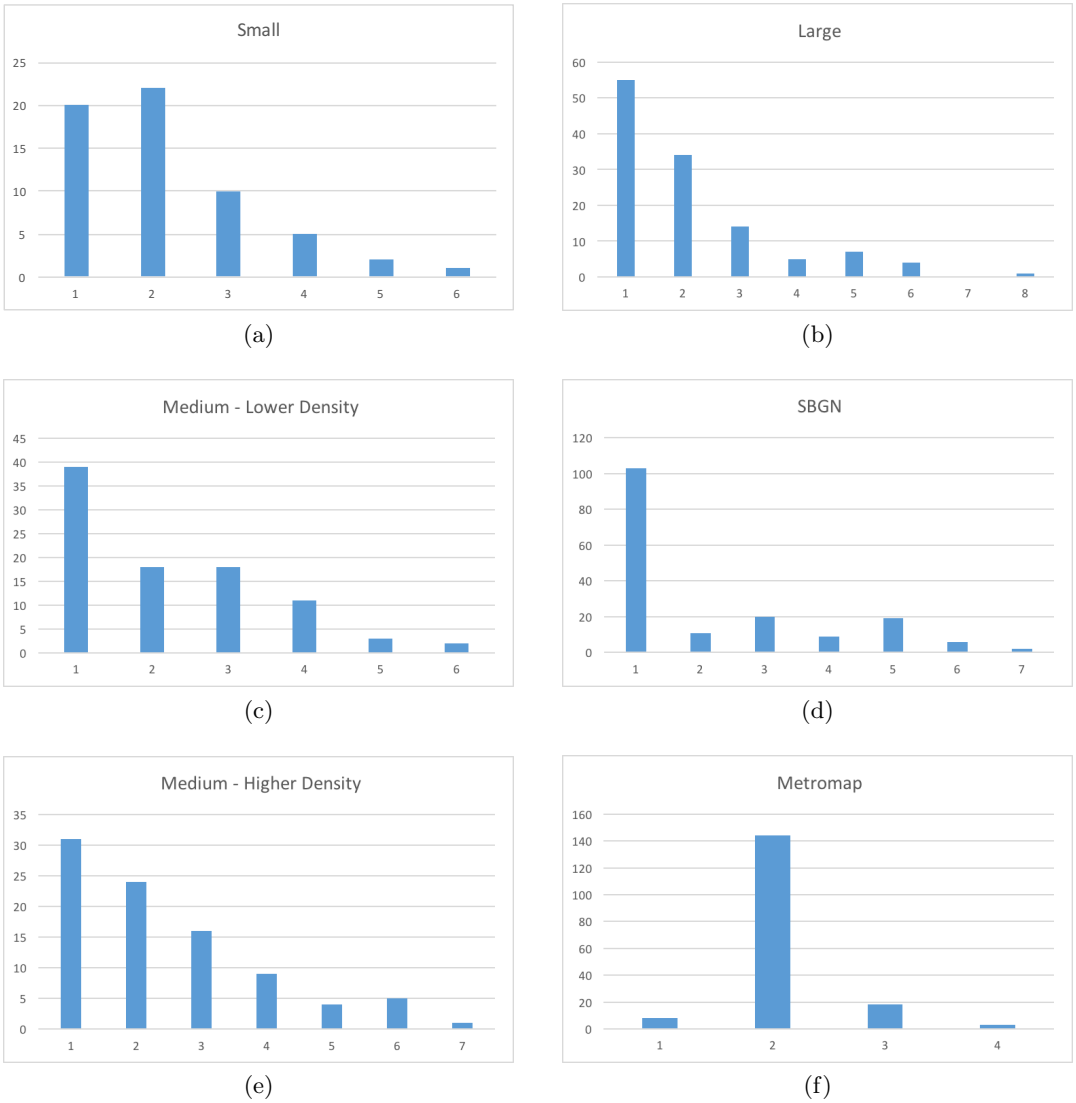


Figure 6.16: Degree histograms show how many nodes of each degree were present in each of the six graphs in our second corpus

amongst u and its clones $u', u'', \ldots$ in some way. In the extreme case enough clones are added so that each of the nodes $u, u', u'', \ldots$ becomes a leaf. Automatic stress-based techniques for cloning have been attempted [EdMN95] but are quite slow, with exponential run time. Some alternative approaches have also been explored [HBF08].

6.5 Conclusions

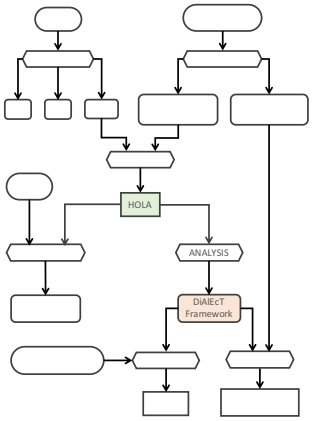
Our four hypotheses **H1-4** were confirmed. HOLA performed comparably to humans, and outperformed yFiles, on the small networks from the formative study. It also beat yFiles both for preference and performance on the large networks except for the denser medium-sized case. These results may be taken as an affirmation of the design of HOLA.

In Chapters 4, 5, and 6 we have (1) conducted a "formative" study to learn about the way people lay out a certain type of network diagram (orthogonal), (2) designed a layout algorithm to achieve similar results automatically, and (3) conducted a "normative" study to check how well our algorithm performed. This is the basic human-centred approach to network layout algorithm design that was promised in Chapter 1 and in the title of the thesis.

If one major question remains about this methodology, however, it is how to make the second step—the algorithm design—more systematic. How do we go from the observations made in the formative study to an algorithm? This is the project of the next chapter.

Chapter 7

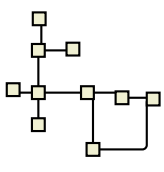
The DiAlEcT Layout Framework



The HOLA algorithm creates orthogonal layouts that mimic the style exhibited in the Orthowontist study; in other words, it can convincingly "speak" the Orthowontist dialect. This raises two questions, the first being: How we can distil out of HOLA a *general framework* that can be applied to create new layout algorithms for different dialects besides the one from the study? To put this differently, we want a systematic way to achieve the second step in our human-centred methodology.

The second question is whether such algorithms are enough. While an algorithm developed according to our methodology can be expected to make the same kinds of design decisions a human "speaker" of the given layout dialect is likely to make, it is bound to eventually make a decision that a particular user will disapprove of. Like the good assistant we imagined in Chapter 1, the system should then understand high-level descriptions of the desired changes.

This chapter answers these questions by defining a four-phase layout framework in which the first three phases are inspired by HOLA, while the fourth is informed by our expectations of an ideal layout assistant. The whole system is an application of CSML, constrained stress-minimising layout. The first three phases yield a layout automatically—the *initial layout*—while the fourth provides interactive tools allowing users to transform the initial layout into one that may be more to their liking. The four phases are called **D***ecompose*/**D***istribute*, **A***rrange*, **E***xpand*/**E***mend*, and **T***ransform*, and the framework is called the DiAlEcT Layout Framework. We may refer to the phases individually by their initials, thus "Phase-D," "Phase-A," etc., or to the first three phases collectively as the "automatic phases" and to the last as the "interactive phase". An overview is given in Figure 7.1.



By a "framework" we mean a high-level algorithm. DiAlEcT identifies the key ideas that were employed in HOLA and ACA and restates them in generic terms. It defines an overall structure, and leaves specific details to be determined in each new application. Each new layout dialect to which the framework may be applied is likely to present new and unique challenges—we will see an example of this in Chapter 8—and the developer will need ingenuity.

In the first three sections of this chapter we cover the three automatic phases, Phase-D, Phase-A, and Phase-E. In these sections the respective phases are described following a consistent format: first the general idea of the phase is described in a section headed "In General," then we examine how this phase was implemented in HOLA and/or ACA in a section headed "In HOLA and ACA," and finally guidelines are given for the future implementation of the phase for new dialects, in a section headed "Application Guidelines." Thus, the idea is explained, examples are given, and it is shown how to apply the idea in the future.

In Section 7.4 we introduce the idea of the *relative constraint matrix*, a data structure recording all the constraints generated during the automatic phases. This, along with a layout, constitutes the output of the first three phases, and the input to the fourth phase. When designing tools according to the DiAlEcT framework constraints should always be described in terms of the relative constraint matrix, as will be seen in Chapters 8 and 9. Therefore it is important to set this foundation now.

Finally we briefly discuss Phase-T of the DiAlEcT framework in Section 7.5, although we do not begin to develop particular interaction methods until Chapter 9.

The framework described in this chapter is by nature abstract, but it is sandwiched between two concrete examples: coming before it is the example of HOLA from which it was induced; coming after it is the example of MCGL, its first direct application. It is in the nature of a chapter like this one to introduce a great number of technical terms. For in a process of induction we must invent terms to refer to the essential aspects of the given concrete example(s) as we identify them. This is necessary so that we may describe the way in which similar things should happen in future applications. In order to help manage the technical terms introduced in this chapter a glossary is provided in Table 7.1.

7.1 Phase-D: Decompose/Distribute

In General

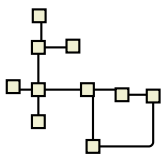
In Phase-D the network may be decomposed into pieces that will be laid out separately before being rejoined in Phase-E. Apart from optionally decomposing, the purpose of Phase-D is to spread out the nodes (of each piece) in the plane, giving them a reasonable distribution that reflects the graph-theoretic structure of the network more or less *faithfully*. As discussed in Section 2.2, a stress-minimal layout is a faithful one. The most straightforward way to achieve Phase-D then, is by a simple DESCEND operation, using Adaptagrams.

In HOLA and ACA

ACA achieves an initial distribution via the DESCEND operation. HOLA does something a bit more involved than this, first decomposing the network into trees and core, then applying DESCEND to the core alone. This effectively enforces a clustering on the nodes of each tree, since they are laid out independently and later reinserted into the drawing with well-defined bounding boxes. Stress-minimisation on the entire network, trees included, would tend to cluster the tree nodes as well, but HOLA takes measures to ensure the

- Phase-D: Decompose/Distribute
  - Use unconstrained stress minimisation to distribute the nodes in the plane.
  - Optionally, decompose the network into parts before distributing.
- Phase-A: Arrange
  - Identify target substructures, i.e. subgraphs for which it is desirable to give some special arrangement.
  - Enter the *constraint generation loop*:
    - Choose next target substructure, or exit loop if none remain
    - Try to arrange the substructure by applying constraints
- Phase-E: Expand/Emend
  - Optionally, planarise and expand faces in order to reinsert parts of network if decomposed in Phase-D.
  - Remove defects by aligning and distributing.
- Phase-T: Transform
  - User can interactively request different arrangements for the substructures arranged in Phase-A, using high-level commands.
  - User can also do low-level editing of the constraints.

Figure 7.1: The DiAlEcT framework describes automatic and interactive layout tools. This figure gives an overview of their functioning. Details are given in this chapter.



Term	Definition	Page
arrangement principles	there are five of these, and they guide the design of Phase-A	105
axis-aligned pair	a GSA (see below) for pairs	103
cardinal-directed hub	a GSA for hubs	104
chain	a maximal connected subgraph consisting of nodes having degree 2 in the larger graph	104
constraint generation loop	the main control structure in Phase-A, during which constraints in the relative constraint matrix are generated in order to enforce GSAs	104
geometric variant	a variation in a GSA that relates to specific geometric questions such as orientation in the plane	102
graph substructure	any special graph-theoretic substructure such as a tree, chain, hub, etc. to which special arrangements are to be applied (see GSA, graph substructure arrangement)	101
GSA: graph substructure arrangement	a scheme for creating geometric arrangements of certain graph substructures, using constraints	102
hierarchical symmetric tree	a GSA for subtrees	104
hub	a subgraph induced by a node and all its neighbours	104
orthogonally-routed chain	a GSA for chains	104
pair	a subgraph induced by a pair of neighbouring nodes	103
relative constraint matrix	data structure representing all the constraints generated in the automatic phases of a DiAlEcT layout process. It is used as input to the interactive phase.	107
structural variant	a variation in a GSA that relates to qualitative design choices, rather than to specific geometric questions such as orientation in the plane	102

Table 7.1: Glossary of technical terms introduced in this chapter

clustering is perfect. Removing the trees also leaves a simpler stress system, allowing the core to more easily find a faithful layout.

Application Guidelines

In future applications of the DiAlEcT framework the lessons from HOLA should be kept in mind. It should be considered whether a decomposition will aid in an overall stress-based distribution process.

Meanwhile there are other reasonable ideas for the distribution phase that were not explored in HOLA, but that may be worth trying. In HOLA the nodes begin at whatever initial positions the user provides (and in our experiments random initial positions were used). Since the local stress minimum reached by a DESCEND operation depends on these starting positions a DiAlEcT layout might fruitfully begin by trying to choose a good initial position based on other considerations, such as minimising crossings. The planarisation approach [GM03] might be applied, for example, or we could try multiple starting points in a greedy randomised adaptive search procedure (GRASP) [FR95].

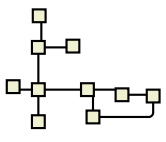
After choosing a starting position in a more principled way, stress should be minimised; however, performing this by a simple DESCEND operation may undo some of the choices that were made. It must be decided whether to simply allow this to happen as it may, or to prevent it, and this represents a trade-off between the faithfulness of the layout and other considerations like crossing minimisation. These concerns tend to be at odds since techniques like the planarisation approach mentioned above, which are good at removing crossings, can lead to very non-faithful (high-stress) layouts. Nevertheless, if it is desired that the topology of the initial position (meaning its rotation system and external face—see Section 2.4.1) not change then the topology-preserving layout of Dwyer et al. [DMW09b] may be applied to now minimise the stress subject to this constraint. This technique is also available in the Adaptagrams library.

7.2 Phase-A: Arrange

It is in Phase-A that we arrange nodes in those special configurations, or *perceptual organisations*, that people often create when working by hand. It is largely to the extent that such arrangements become conventional among a given community of diagram authors that we attribute to that community a layout dialect of its own.

In all such node arrangements there are choices to be made, and here we make a division of labour between Phase-A and Phase-T of the DiAlEcT framework. The automatic layout algorithm makes a "standard choice" in Phase-A, whereas tools are provided for users to alter these choices interactively in Phase-T. Accordingly, Phase-A cannot be discussed without at least mentioning Phase-T as well, although the main discussion of Phase-T is reserved for Section 7.5. Furthermore it will become clear that Phases -A and -T are the ones most directly influenced by the initial, formative user study that makes Step 1 of our human-centred methodology.

This section is broken into two subsections, in each of which we follow the expository pattern of "In General"/"In HOLA and ACA"/"Application Guidelines". Section 7.2.1 defines *graph substructures* and their *arrangements*; Section 7.2.2 introduces the idea of the *constraint generation loop*.



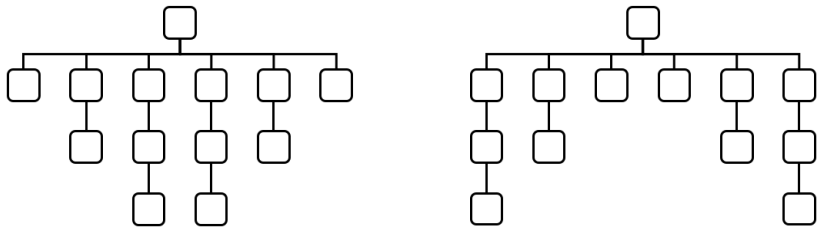


Figure 7.2: Like most GSAs, the symmetric tree layout admits both geometric and structural variants. Both layouts depicted here employ the same geometric variant; namely, they both grow in the SOUTH direction. However they employ different structural variants. On the left we choose to put the deepest trees nearest their parent node; on the right we put the shallowest trees nearest the parent node.

7.2.1 Graph Substructure Arrangements

In General

By a *graph substructure* we mean any identifiable substructure of a graph, defined purely in graph-theoretic terms. The trees and chains identified in Chapter 5 are examples. By a *graph substructure arrangement* or GSA we mean a way of laying out the nodes of a particular graph substructure using constraints. Consider for example the symmetric tree layout applied to the subtrees in HOLA. The trees are a graph substructure, and the symmetric tree layout is a GSA.

Often the basic idea of a GSA admits variations, and we categorise these as being of two types, which we call *geometric variants* and *structural variants*. While there is no strict definition there are the following guidelines.

Generally speaking geometric variants are chosen based on some computation involving the current geometry (hence the name), with the goal of minimising the concomitant stress increase when constraints are applied to enforce the arrangement. They are often concerned with the way in which a substructure is rotated, relative to the rest of the network.

On the other hand structural variants have more to do with the internal arrangement of nodes within the GSA, and less to do with how these nodes relate to the remainder of the network. The choice of structural variant tends not to be influenced by the current geometry, but is instead based on a principle, such as maximising symmetry or convexity of the GSA.

To be clear, when arranging a GSA we must in general choose *both* a geometric and a structural variant; however, some GSAs are so simple that they have no internal structure and only a geometric variant need be chosen. Examples will be seen below.

To return to the example of the symmetric tree layout, the geometric variants are simple: these are the four cardinal growth directions introduced in Section 5.3.3. As for the structural variants, recall it was briefly indicated in Section 5.3.3 that the symmetric tree layout algorithm is recursive, and at each stage tries to pair off isomorphic subtrees relative to their common parent node. But as Figure 7.2 illustrates, we are free to choose how we order those isomorphic pairs, and this leads to the structural variants of the symmetric tree layout. In one variant we put the deepest trees nearest to the parent node. In another we put the shallowest trees nearest to the parent node. Others would be possible.

When designing HOLA we had to choose among the many possible structural variants, and we chose to always put the deepest trees nearest the parent node as on the left of Figure 7.2. This choice accords with the finding of van Ham and Rogowitz [vHR08] that convexity is a desirable property. If we had had good reason to do so we could have

made the choice conditional on some structural properties of a given tree; however, what distinguishes structural variants from geometric ones is that the former are based on internal structure of the GSA so can be planned before execution of the algorithm begins, while the latter are based on the current geometry during the layout process, so must be chosen at runtime. What this suggests is a fundamental division of labour both between Phases -A and -T, and between design time and execution time:

Division of Labour:

When *designing* Phase-A we must set down rules for each GSA saying how structural variants are to be chosen. We call these *standard structural variants*, with the idea that there are usually other reasonable structures that could have been chosen instead. We call the latter the *alternative structural variants*, and we must design Phase-T to make these configurations easily reachable interactively from the standard ones.

When *executing* Phase-A a *geometric* variant will be chosen for each GSA based on the current position. Phase-T should also provide easy ways to switch to other geometric variants interactively.

In summary, when designing Phase-A based on the formative user study of the human-centred methodology we must:

- Decide on the set of graph substructures (trees, chains, faces, etc.) for which special arrangements may be created.
- Encode the observed, dialect-specific layout behaviours as a set of possible GSAs.
- Define for each GSA:
  - a set of structural variants, and among these the *standard structural variants*;
  - a system for choosing geometric variants based on given node positions during the layout process;
  - a system for enforcing a chosen variant using separation and alignment constraints.

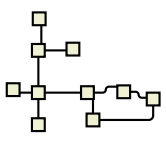
These ideas will become more clear as we consider the examples in HOLA and ACA, and as we apply them to SBGN in Chapter 8.

In HOLA and ACA

In HOLA and ACA we identified several graph substructures of interest. Recall that in Section 5.2.1 we referred to these as *target substructures*. So far we have designed one GSA for each of these, and these are listed in Table 7.2. We may briefly elaborate on these structures and arrangements:

- Pairs: By a *pair* we mean a subgraph consisting of two neighbouring nodes and the edge that connects them. The pair was the only substructure on which ACA operated. It tried to align each pair either vertically or horizontally, and we refer to this as the *axis-aligned pair* arrangement.[1] This is an example of a GSA so simple that it has no internal structure, and hence no structural variants.

[1] This is an alternative way of understanding what we were doing in ACA. At the time we thought of it as aligning edges, but if we instead say that we were operating on node pairs then we get a unified theory that explains what was happening in both ACA and HOLA in terms of GSAs.



Graph Substructure	Graph Substructure Arrangement
pair	axis-aligned pair
tree	hierarchical symmetric tree
hub	cardinal-directed hub
chain	orthogonally-routed chain

Table 7.2: Graph substructures and their arrangements in HOLA and ACA

- Trees: The graph substructure called a *tree* is as defined in Chapter 4. The *hierarchical symmetric tree* or HST arrangement is achieved by the symmetric tree layout of Manning and Atallah, as employed in the HOLA algorithm.
- Hubs: By a *hub* we mean a subgraph consisting of any node u together with all its neighbouring nodes and the edges that connect them. The node u is referred to as the *centre* of the hub. The "node configuration" step in HOLA (Section 5.3.2) served to arrange those hubs whose centres were of degree three or higher. For each hub of centre u and neighbours v_i, the arrangement tried to place up to four of the v_i in the cardinal compass directions from u. We therefore refer to this as the *cardinal-directed hub* arrangement.
- Chains: The graph substructure called a *chain* is as defined in Section 4.4. In HOLA we developed a way to arrange chains that is analogous to the way an orthogonal connector may be routed. We therefore refer to this as the *orthogonally-routed chain* arrangement.

Application Guidelines

In future applications of the DiAlEcT framework we expect the GSAs explored in HOLA and ACA to once again be useful. All known GSAs should be regarded as forming a steadily growing catalogue, from which we can draw and to which we expect to add each time we apply the DiAlEcT framework anew. We may view this catalogue as having begun with Marks [Mar91a], who considered the tree and hub structures. Identifying and designing new GSAs when studying new layout dialects is one of the important steps of the human-centred approach to network layout algorithm design.

7.2.2 The Constraint-Generation Loop

In General

The *constraint generation loop* (CGL) constitutes the main activity of Phase-A. On each iteration of the loop we choose one or more constraints and add them to the layout. Typically this means choosing a particular graph substructure instance and attempting to create a GSA for it. The loop ends when every desired arrangement has either been created, or else attempted and failed due to conflicting constraints.

We may view the CGL as a *greedy* process since we make *locally optimal choices*, and *do not backtrack* based on their outcome. However, the "locally optimal" choices tend to have a forward-looking aspect: As will be seen in the examples from HOLA and ACA examined below and in the guidelines for future application, each iteration of the CGL tries (or should try) both to create a good GSA and to plan for the creation of good GSAs in later iterations.

Another important aspect of designing the CGL is deciding when to use the PROJECT and DESCEND operations. As noted in Sections 5.3.2 and 6.4, this means a trade-off between speed and quality. The PROJECT operation is much faster than the DESCEND

operation, but repeated PROJECT's lead to accumulating stress, and the occasional[2] DESCEND is required to dissipate this so that the geometric variants of subsequent GSAs can be chosen in a faithful way, i.e. in the way that best represents the structure of the network.

In HOLA and ACA

Both HOLA and ACA feature a constraint-generation loop, over which constraints are gradually applied in order to arrange the nodes of the graph. Actually HOLA could be said to involve a series of CGLs – one for its "node configuration" stage, and a subsequent one for its "chain configuration" stage; however, whether these are regarded as two CGLs or merely a single one with two phases is only a question of semantics and does not change the basic idea.

There are several principles involved in managing the CGL, examples of which were demonstrated in HOLA and ACA:

1. In both ACA and HOLA we saw constraint choices guided by the desire to keep stress low.

2. In HOLA we saw choice of standard arrangements, such as symmetric layouts for trees.

3. In both ACA and HOLA we opted for flexible separation constraints of the form $u + g \leq v$, $g > 0$, rather than rigid ones of the form $u + g = v$ (i.e. we used inequalities, not equations), with the idea that greater flexibility would permit better arrangements in subsequent iterations of the CGL.

4. In HOLA there was an approach for allowing neighbouring GSAs to fit together, balancing the demands of each. For example, the *orthogonally routed chain* arrangement of each chain was designed to accommodate any *cardinal-directed hub* arrangement its terminal nodes may already have been given (see Table 7.2).

5. In HOLA we saw a careful balance between speed and quality maintained by using the fast PROJECT operation for most constraint applications, while the costlier DESCEND operation was reserved for key moments when it was deemed necessary. For example it was used to obtain new configuration options when all others had failed during the node configuration step, and as preparation for chain configuration (see page 67).

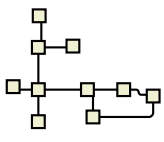
Application Guidelines: The Arrangement Principles

The five practices of HOLA and ACA noted above demonstrate good principles that should always be applied in the constraint generation loop of Phase-A. We refer to these as the *arrangement principles*, and will denote them in the remainder of the thesis by their name in boldface with a capital letter.

1. **Faithfulness**: Again, we describe a stress-minimal layout as a faithful one. Therefore an arrangement process will be said to follow the arrangement principle of **Faithfulness** when it tries to keep stress low while creating arrangements.

2. **Standardisation**: An arrangement process will be said to follow the arrangement principle of **Standardisation** when it chooses good *standard structural variants* for each GSA, reserving *alternative variants* to be created during Phase-T if desired.

[2]This is not to say that the occasions for application of DESCEND should be determined randomly or by manual intervention; rather it is during the design of Phase-A that such occasions must be determined.



3. **Minimality**: An arrangement process will be said to follow the arrangement principle of **Minimality** when it applies the minimum constraints necessary to enforce a given GSA, with the aim of permitting more GSAs to succeed later in the CGL. This means favouring inequalities over equations, and fewer constraints overall.

4. **Cooperation**: An arrangement process will be said to follow the arrangement principle of **Cooperation** when it combines neighbouring GSAs in a fair and even-handed way, giving fair weight to their competing demands.

5. **Balance**: An arrangement process will be said to follow the arrangement principle of **Balance** when it balances use of the two operations PROJECT and DESCEND, finding a mean point between speed and quality.

If we can find little demonstration of **Standardisation** in HOLA (let alone ACA), that is because we contemplated no interactive Phase-T at that time, whereas **Standardisation** essentially involves the separation of arrangement choices between Phase-A and Phase-T. All five principles will in any case be employed fully in Chapters 8 and 9.

7.2.3 Summary

In summary, and now in terms of the five arrangement principles, the task of designing Phase-A for a new layout dialect involves the following steps:

1. Identify any special GSAs for the layout dialect in question, and define for each of these:

   (a) the structural and geometric variants;

   (b) a system for choosing from among these variants during the constraint generation loop, conforming to **Faithfulness** and **Standardisation**;

   (c) a system for enforcing the chosen variant using separation and alignment constraints, conforming to **Minimality**.

2. Describe the constraint generation loop, including:

   (a) the order in which GSAs are considered, and how to make neighbouring GSAs combine, conforming to **Cooperation**;

   (b) how and when to use the PROJECT and DESCEND operations, conforming to **Balance**.

7.3 Phase-E: Expand/Emend

In General

If any decomposition was performed in Phase-D, then it is in Phase-E that pieces of the laid out network may be expanded to make room for reinsertion of other pieces. Meanwhile the word "emend" comes from Latin *ex-*, plus *mendum*, meaning "defect" or "fault"; thus, to emend is to remove defects. That is another task for this phase. A bit like building a piece of furniture, this final phase is all about polishing, refining, and finishing each separate piece, as well as fitting the pieces back together.

In HOLA and ACA

In HOLA Phase-E took place from Step 3b to the end of the algorithm. The core is planarised in order to expand the faces and make room for reinsertion of the trees. Finally we created any alignments that were near enough to appear to have been overlooked, minimised the neighbour-stress function in order to achieve more even distribution of nearby nodes, rotated for aspect ratio, and removed dummy nodes left over from the expansion process.

Application Guidelines

At this time no further measures or alternative techniques are recommended for Phase-E beyond what was already demonstrated in HOLA. Future DiAlEcT layouts might take alternative approaches expanding, and might extend this phase by any reasonable measures designed to remove what appear as defects in the layout after the completion of Phase-A.

7.4 The Relative Constraint Matrix

Besides a layout, Phases D, A, E produce a set of constraints, namely, the final set of separation constraints that were imposed on the graph in order to achieve the final layout. We must provide these constraints as input to the interactive Phase-T, since transforming the layout will be a matter of altering constraints and applying DESCEND.

In order to facilitate the transformations of Phase-T it is essential that the constraints be described using a formalism we call the *relative constraint matrix*. This permits a kind of algebra of constraints and their transformations.

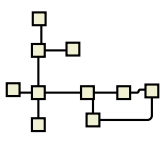
The relative constraint matrix is so called because it has an entry for every pair of nodes, describing the constrained relation that has been set up between those two nodes, if any. Separations between nodes are given as multiples of a *grid size*. Therefore, together with a chosen grid size G, the matrix describes all the constraints on the graph.

In this section we give a mathematically precise definition of the relative constraint matrix and of the expressions that make up its entries. This is foundational for an understanding of how Phase-T works, for describing the constraints applied in Phase-A when we develop a layout algorithm for SBGN in Chapter 8, and for developing interaction methods in Chapter 9.

For a given relative constraint matrix M and nodes a, b, we denote by $M(a, b)$ the *directed relation* from a to b under M. A directed relation from node a to node b is a symbolic expression indicating the constraints relating the two nodes (if any), in both the x- and y-dimensions, and understood as operating in the direction from the first node a toward the second node b.

Directed relations are written using the *lateral direction letters* $\mathbb{R}, \mathbb{D}, \mathbb{L}, \mathbb{U}$, and the *cardinal direction letters* $\mathbb{E}, \mathbb{S}, \mathbb{W}, \mathbb{N}$, the meanings of which are given in Table 7.3. Essentially, cardinal direction letters let us constrain one node to lie in a cardinal compass direction from another, as we have been doing since Chapter 3, whereas lateral direction letters allow us to set a weaker constraint that involves a side, but not an alignment. For example $M(a, b) = \mathbb{R}$ means that node b must lie on the right-hand side of node a, whereas $M(a, b) = \mathbb{E}$ means this plus horizontal alignment. It is the same as one of the *separated alignments* that we applied in Chapter 3.

Furthermore any direction letter may be given a subscript '=' to indicate that its separation constraint is exact (i.e. involves an equation rather than inequality), and may be given an exponent equal to any non-negative real number r meaning that the gap G in the separation constraint is to be multiplied by r. As an example, the various forms are spelled out for letters $\mathbb{R}$ and $\mathbb{E}$ in Table 7.4.



Lateral Direction Letters		
Letter	Idea	Meaning
$\mathbb{R}$	right	$a_x + G \leq b_x$
$\mathbb{D}$	down	$a_y + G \leq b_y$
$\mathbb{L}$	left	$b_x + G \leq a_x$
$\mathbb{U}$	up	$b_y + G \leq a_y$
Cardinal Direction Letters		
Letter	Idea	Meaning
$\mathbb{E}$	east	$a_x + G \leq b_x \wedge a_y = b_y$
$\mathbb{S}$	south	$a_y + G \leq b_y \wedge a_x = b_x$
$\mathbb{W}$	west	$b_x + G \leq a_x \wedge a_y = b_y$
$\mathbb{N}$	north	$b_y + G \leq a_y \wedge a_x = b_x$

Table 7.3: The lateral and cardinal direction letters are used in the directed relation expressions that form the entries of a relative constraint matrix. In this table G is the grid size, a and b are nodes with coordinates (a_x, a_y) and (b_x, b_y) respectively, and the constraints interpret the direction letters as operating from node a toward node b.

Letter	Meaning
$\mathbb{R}$	$a_x + G \leq b_x$
$\mathbb{R}_{=}$	$a_x + G = b_x$
$\mathbb{R}^r$	$a_x + rG \leq b_x$
$\mathbb{R}^r_{=}$	$a_x + rG = b_x$
$\mathbb{E}$	$a_x + G \leq b_x \wedge a_y = b_y$
$\mathbb{E}_{=}$	$a_x + G = b_x \wedge a_y = b_y$
$\mathbb{E}^r$	$a_x + rG \leq b_x \wedge a_y = b_y$
$\mathbb{E}^r_{=}$	$a_x + rG = b_x \wedge a_y = b_y$

Table 7.4: Direction letters may take subscripts and exponents. In this table examples are shown for the letters $\mathbb{R}$ and $\mathbb{E}$. G is the grid size, $r \geq 0$ is a non-negative real number, a and b are nodes with coordinates (a_x, a_y) and (b_x, b_y) respectively, and the constraints interpret the direction letters as operating from node a toward node b.

Letter	$\mathbb{R}$	$\mathbb{D}$	$\mathbb{L}$	$\mathbb{U}$	$\mathbb{E}$	$\mathbb{S}$	$\mathbb{W}$	$\mathbb{N}$
Cardinal Strengthening	$\mathbb{E}$	$\mathbb{S}$	$\mathbb{W}$	$\mathbb{N}$	-	-	-	-
Lateral Weakening	-	-	-	-	$\mathbb{R}$	$\mathbb{D}$	$\mathbb{L}$	$\mathbb{U}$
Opposite	$\mathbb{L}$	$\mathbb{U}$	$\mathbb{R}$	$\mathbb{D}$	$\mathbb{W}$	$\mathbb{N}$	$\mathbb{E}$	$\mathbb{S}$

Table 7.5: All direction letters have opposites. Lateral letters have cardinal strengthenings, while cardinal letters have lateral weakenings.

Each entry $M(a, b)$ in a relative constraint matrix M expresses the directed relation from node a to node b using zero, one, or two direction letters, together with subscripts and exponents. Before defining the format of these expressions precisely, we consider a few examples:

- If no constraints have been set relating a and b, then we write
$$M(a, b) = \varepsilon.$$
- If b is constrained to lie at least two grid units east of a, then
$$M(a, b) = \mathbb{E}^2.$$
- If b is constrained to lie exactly one unit right, and at least three units up from a, then
$$M(a, b) = \mathbb{R}_{=}\mathbb{U}^3.$$

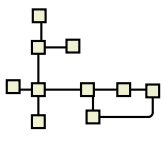
In order to define the expressions precisely we first introduce some terminology. The letters $\mathbb{R}, \mathbb{L}, \mathbb{E}, \mathbb{W}$ (right, left, east, west) are called *horizontal* direction letters, and the letters $\mathbb{D}, \mathbb{U}, \mathbb{S}, \mathbb{N}$ (down, up, south, north) are called *vertical* direction letters. Thus we have *two* ways of categorising direction letters: they may be lateral or cardinal (see Table 7.3), and they may be horizontal or vertical.

If $\mathbb{X}$ is any direction letter and $r \geq 0$ a non-negative real number, then the following are referred to as *augmented direction letters*: $\mathbb{X}$, $\mathbb{X}_{=}$, $\mathbb{X}^r$, $\mathbb{X}^r_{=}$. Finally, the precise definition of the legal *directed relation expressions* is as follows:

1. ε is a directed relation expression
2. Every augmented direction letter is a directed relation expression
3. The concatenation of any augmented horizontal lateral direction letter with any augmented vertical lateral direction letter is a directed relation expression.

Note that the cardinal compass directions are used only as a convenience, since, for example, $\mathbb{E}^r$ (east at least r units) is equivalent to $\mathbb{R}^r\mathbb{U}^0_{=}$ (right at least r units and up exactly 0 units) and to $\mathbb{R}^r\mathbb{D}^0_{=}$ (right at least r units and down exactly 0 units) as well. Note also that despite its reliance on just the four cardinal and four corresponding lateral directions, the relative constraint matrix actually permits arbitrary two-dimensional offsets between nodes, not just orthogonal ones. For example, this is one way to encode the diagonal offsets that appear in the *parallel chain groups* of Section 8.4.1.

Each direction letter has an *opposite* letter—see Table 7.5. We may observe then that any relative constraint matrix M must be anti-symmetric, meaning that for all nodes a, b, the entry $M(b, a)$ is obtained from the entry $M(a, b)$ by swapping each direction letter for its opposite. Note furthermore that every relative constraint matrix must be equal to ε along the diagonal since we do not set any constraints between a node and itself. Finally, it is sometimes useful to speak of the *cardinal strengthening* of a lateral letter, and vice versa the *lateral weakening* of a cardinal letter—see again Table 7.5.



7.5 Phase-T: Transform

The fourth and final phase of the DiAlEcT framework is the interactive Phase-T, in which users can transform the initial layout. Phase-T must receive as input at least the layout (i.e. node positions and edge routes) and the relative constraint matrix (RCM) computed by the automatic phases; it may also receive data structures representing any GSAs computed during Phase-A.

The way that changes to the layout are made in Phase-T is not by editing node positions directly, but instead by editing the RCM. Each time the user requests a transformation two things happen: first the RCM is updated, and second an application of the DESCEND operation makes the layout settle into a new position in which the desired transformation has taken place. Recalling that DESCEND involves application of PROJECT, if there are any conflicts in the new set of constraints then these will be detected by the PROJECT operation. In that case the user will be notified, and given options for resolving the conflict.

Editing of the RCM is possible on two levels: high and low. On the high level the user can request broad changes in easily understood terms. For example the user can ask that a subtree with EAST growth direction instead be made to grow NORTH. The system automatically translates the high-level request into the required changes to the entries of the RCM. On the low level the user is also allowed to edit any single RCM entry directly.

As was indicated in Section 7.2, high-level transformations are to be provided for switching between the structural and geometric variants of GSAs. It is in order to make such GSA transformations possible that Phase-T may accept as input any special data structures representing the GSAs computed in Phase-A. Whether such a data structure is required depends on whether the GSA in question has sufficient internal structure. For example the *axis-aligned pair* GSA has no internal structure so needs no data structure to represent it, but the *hierarchical symmetric tree* does.

Besides GSA transformations we believe that certain actions like flips and rotations are also natural, and we will examine evidence for this in the Orthowontist dynamic data in Chapter 9. It is also in that chapter that we develop specific interaction methods for Phase-T.

7.6 Conclusions

The DiAlEcT layout framework provides a way of designing automatic and interactive layout tools to achieve network layouts conforming to the conventions of any real-world layout dialect. It consists of four phases, Distribute, Arrange, Emend, and Transform, of which the first three are automatic, and the final one is interactive.

The automatic phases are motivated by generalisation from the processes of HOLA (Chapter 5), and of ACA (Chapter 3). *Application guidelines* were given in Sections 7.1, 7.2, and 7.3 for each of the three automatic phases, indicating how they should be designed when countenancing new layout dialects in the future. The interactive phase is a part of making the computer into a good assistant, which can quickly and easily make desired changes to the initial layout, and specific interaction methods will be developed in Chapter 9.

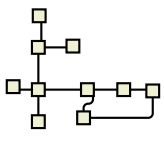
Phases D and E help to embody a human approach to pure layout *aesthetics*, without regard to special *perceptual organisations* of nodes. Phase D may be employed to strike a trade-off between stress minimisation and crossing reduction. Phase E allows opportunistic improvements like aligning nearby and slightly dis-aligned nodes.

Meanwhile when we study a new layout dialect we design Phases A and T to capture its conventional *perceptual organisations* in graph substructure arrangements (GSAs) and their transformations. A small catalogue of GSAs has been extracted from ACA and

HOLA (Table 7.2), and this is expected to grow as new DiAlEcT layouts are designed. The catalogue should be useful in future efforts, since many GSAs may be common across various layout dialects. For each GSA, both *structural variants* and *geometric variants* are to be identified.

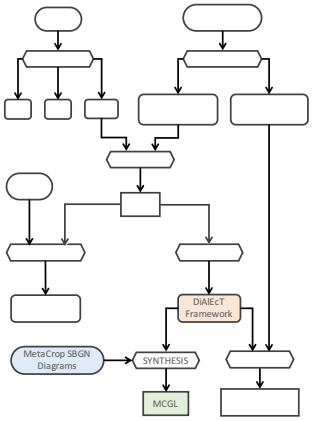
The constraint generation loop of Phase-A is expected to adhere to the five *arrangement principles* identified on page 105. In particular it is to be decided which structural variants of each GSA will be generated automatically (the "standard" variants). Interaction techniques should then be provided in Phase-T to permit the user to choose different structural and geometric variants.

The guidelines articulated in this chapter are meant to facilitate a scenario like that imagined in Section 1.3, where Firm B developed layout tools for Firm A. We test the system in the next chapter, where we try applying the DiAlEcT framework to a layout dialect of the SBGN language.



Chapter 8

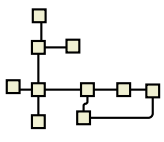
DiAlEcT Layout for SBGN



The previous chapter introduced the DiAlEcT layout framework, and in order to verify the usefulness of that framework we must now demonstrate how it can be applied to a new layout dialect. In the present chapter we do just that, selecting as our test case a dialect of the SBGN or Systems Biology Graphical Notation layout language. Developed in 2005 [KFMO05, KA06, LHM+09], SBGN is used by systems biologists to understand and communicate things like plant and animal metabolism, and the connections between genes and disease pathogenesis. The chosen dialect is that demonstrated in the layouts of the MetaCrop system [SCC+12], published by the Leibniz Institute of Plant Genetics and Crop Plant Research in Gatersleben, Germany. A representative diagram from the MetaCrop system won the "Best Map" award in the first annual SBGN layout competition [Com]. The algorithm developed in this chapter is called MCGL (pronounced "McGill"), for MetaCrop Graph Layout.

A clarifying remark is in order: There are in fact three distinct languages defined in the SBGN standard, intended for representing different types of information about biological systems. These are known as the Process Description (PD), Entity Relationship (ER), and Activity Flow (AF) languages. What concerns us in this chapter is the PD language only, and it is this that is meant by all references made here to "SBGN". Process Description diagrams describe things like metabolic processes, such as the breakdown of sugars in plant or animal cells (Figure 8.1).

Not only conventions of the MetaCrop dialect, but rules of the SBGN language itself will bear on our design of the phases of DiAlEcT. This will likely prove typical of applications of the framework. In this case the special features of the language that demand attention are *compartments*, *ports*, and *orientable nodes*, and these features are examined in Sections 8.1 and 8.2. Note that in our treatment of ports we borrow ideas from our



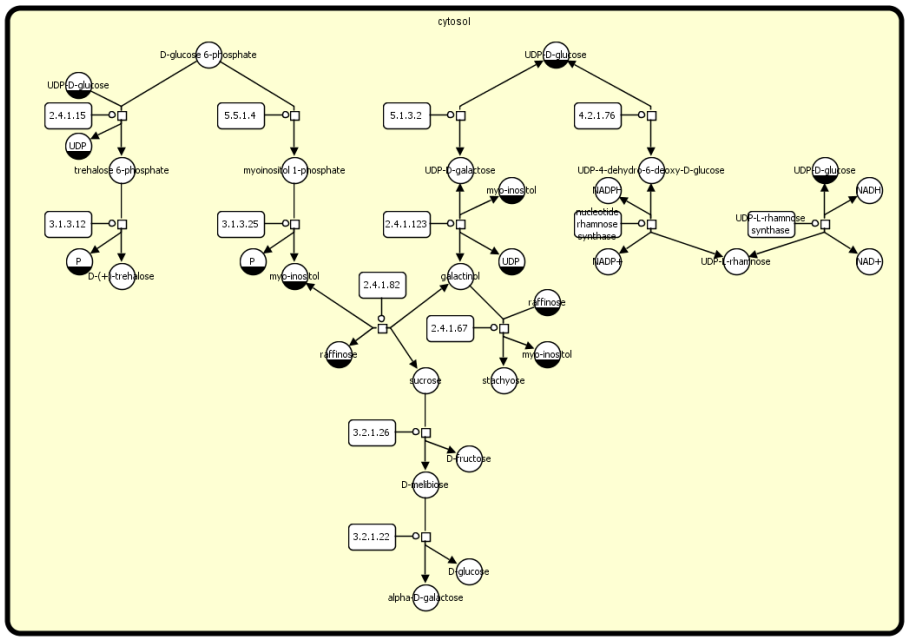


Figure 8.1: Sugar metabolism process diagrammed in SBGN, courtesy of MetaCrop [Met]

publication [RKD+14] on layout of Data Flow diagrams. That work is described briefly for the sake of clarity in Section 8.2, but was developed separately from this work on SBGN layout.

After this, Sections 8.3 through 8.5 address the three automatic phases D, A, and E of the DiAlEcT framework. We postpone our look at Phase-T for SBGN diagrams until Chapter 9, where it can be given a uniform treatment along with more generic interaction methods.

The corpus of existing hand-made SBGN network layouts on which the work in this chapter is based consists of twenty-one diagrams representing amino acid metabolism, carbohydrate metabolism, and energy metabolism pathways in plant cells. Besides studying these hand-curated layouts we also spoke to the experts who created them, and these steps served as a replacement for the initial formative user study of the human-centred methodology. As for the final normative user study, we did not have time to carry that out for the MCGL algorithm, and this remains for future work. The main purpose of the present chapter is only to demonstrate the methodical applicability of the DiAlEcT framework, and show how it allows us to carry out the second step of the human-centred methodology (algorithm design).

We can sort SBGN diagrams into two categories, which in this thesis we call *Type I* and *Type II*, and it is important to note that all the diagrams in the corpus are Type I diagrams. The definition is simple: a diagram is Type I if all modulators (see Section 8.1.1) are leaves; otherwise it is Type II.

Since the MCGL algorithm developed in this chapter is designed to create MetaCrop-style layouts only of Type I diagrams, it must be viewed as the first of two or more steps toward algorithmically mastering the style. In Section 8.6 we observe MCGL's excellent performance on Type I diagrams. We also consider its behaviour on a Type II diagram and consider what kinds of additional features may be called for in a "MCGL II" algorithm that could work just as well on diagrams of this kind.

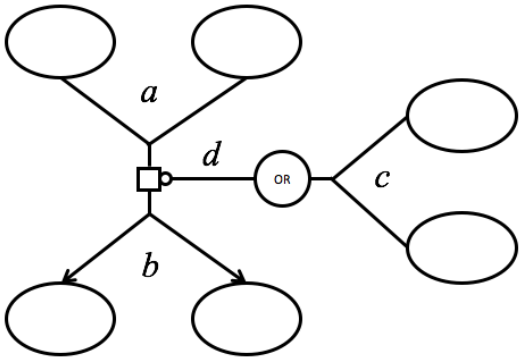


Figure 8.2: Oval nodes are entity nodes, the small square node is a process node, and the node labelled OR is a logic node. Among the arcs are (a) consumption arcs, (b) production arcs, (c) logic arcs, and (d) a modulation arc.

8.1 Basics of SBGN Diagrams

The SBGN specification defines many different types of node and many different types of edge (or "arc" as is more common in SBGN parlance), but for layout purposes this typology can be collapsed into a much coarser set of node and arc types, given in Table 8.1.

To begin with, SBGN defines several different "entity pool nodes" to represent things like "simple chemicals", "macromolecules", and "nucleic acid features"—biochemicals that participate in metabolic and other processes—as well as more abstract node types like "sources" and "sinks", but for layout purposes we may treat all of these in the same way, and we refer to them all by the name *entity nodes*.

Next, SBGN defines several different types of node to represent processes in which chemicals combine or break apart, or react to produce other chemicals, but for layout purposes we refer to all of these as *process nodes*. Process nodes connect to entity nodes representing the chemicals that take part in the represented process. They also connect to entity nodes representing chemicals that *modulate* the process, such as enzymes.

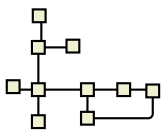
Finally, SBGN has nodes representing "logic gates", AND, OR, and NOT, which serve to show how various modulating chemicals combine. For example if two enzymes must both be present in order for a certain reaction to be stimulated, these two would each connect to an AND node, which in turn would connect to a process node. We refer to these as *logic nodes*.

SBGN defines several types of *arc* (edge), but again we do not need to know about all of these for layout purposes. For layout it is enough to identify the *consumption arc* connecting a consumed entity node to a process node, the *production arc* connecting a produced entity node to a process node, the *modulation arc*, connecting a process node to a logic node or to an entity node acting as an enzyme or modulator, and finally the *logic arc*, connecting an entity acting as modulator to a logic node. See Figure 8.2. Table 8.1 also features the *submap* and *equivalence arc*, which are discussed in Section 8.1.2.

8.1.1 Orientable Nodes

Process nodes and logic nodes have ports. In SBGN these are realised via two "spikes" extending on opposite sides of the node, as in Figure 8.3. The ports on process and logic nodes serve to partition the neighbours of these nodes into meaningful groups, corresponding to the type of arc that connects them.

All consumption arcs connected to a process node must attach at one of the ports, and all production arcs must attach to the opposite port, while all modulation arcs must



SBGN element	For layout purposes	Type code
unspecified entity simple chemical macromolecule nucleic acid feature perturbing agent source/sink complex multimer phenotype	entity node	EN
process omitted process uncertain process association dissociation	process node	PN
and operator or operator not operator	logic node	LN
submap	submap	SN
consumption arc	consumption arc	CA
production arc	production arc	PA
modulation arc stimulation arc catalysis arc inhibition arc necessary stimulation arc	modulation arc	MA
logic arc	logic arc	LA
equivalence arc	equivalence arc	EA

Table 8.1: Mapping from SBGN elements to a coarser classification that is sufficient for layout purposes. Each element of the coarse classification gets a *type code*, which is used in Section 8.2.3 when we introduce our formal model of SBGN graphs.

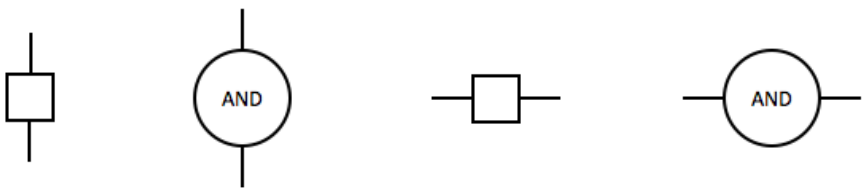


Figure 8.3: Process nodes and logic nodes have ports located at the ends of two "spikes" that stick out on opposite sides. The spikes can be oriented vertically or horizontally.

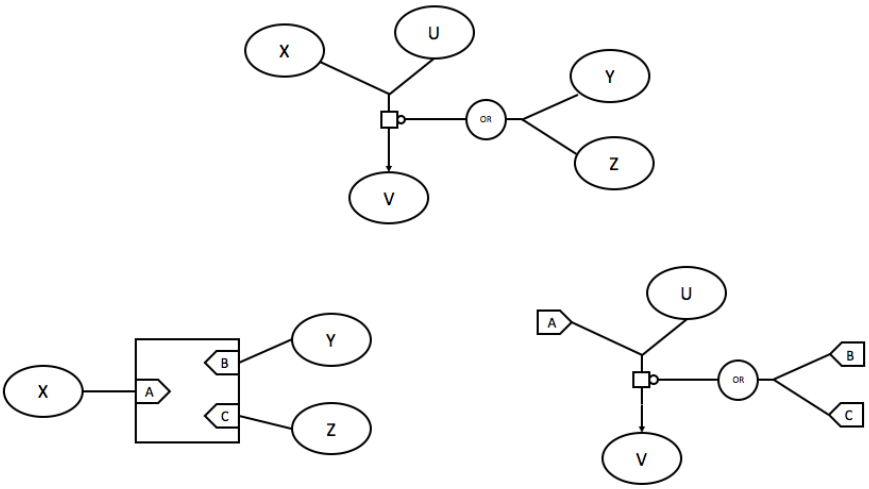


Figure 8.4: The SBGN diagram at the top can be decomposed into those on the bottom left and bottom right, using a submap. Bottom left shows the three entity nodes X, Y, and Z connecting to a submap. Bottom right shows the internal contents of the submap, and the tags A, B, and C serve to indicate how elements of this diagram connect to nodes outside the submap.

meet the node on either of the two remaining sides, i.e. the two sides of the box that do not have spikes. (See for example modulation arc d in Figure 8.2.)

Similarly all logic arcs connecting to a logic node must attach at one of its ports, while it is constrained to have a single modulation arc attached to its opposite port (representing the "output" of the "logic gate").

The presence of a pair of ports and the ends of two spikes makes process and logic nodes *orientable*, since as a part of the layout process it must be decided for each process and logic node whether its spikes will be aligned horizontally or vertically. We refer to logic and process nodes collectively as *orientable nodes*, or O-nodes for short.

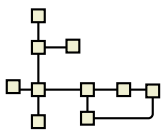
When v is a process node it will be useful to be able to refer specially to the neighbours of v that are connected by modulation arcs. We refer to such neighbours as *modulators*, and denote the set of all modulators of process node v by $\mathrm{Mod}(v)$. Recall that our definition of Type I and Type II SBGN diagrams is given in terms of modulators; namely, a diagram is Type I if and only if all modulators are leaves.

8.1.2 Submaps

SBGN provides a way to hide detail by representing a subgraph by a single node called a *submap*. Nodes outside the suppressed subgraph attach to the submap node at labelled terminals. These serve as reference points that can be replicated as *tags* in a separate diagram showing the subgraph. All connections to tags are via *equivalence arcs*. See Figure 8.4.

8.1.3 Compartments

For biologists it is often important to know *where* a biochemical reaction takes place, for example within a certain organelle inside a cell. For this SBGN features the *compartment*, a bounded region of arbitrary shape—though in MetaCrop always drawn as a rounded rectangle—into which nodes can be placed, and which can be nested into other compartments. This introduces a hierarchical structure to SBGN diagrams. See for example the left-hand side of Figure 8.11 on page 128, which features five compartments, distinguished by colour.



Compartments carry labels naming the structure or region they represent. For example there could be a "Mitochondrion" compartment nested inside a "Membrane" compartment, inside a "Cytosol" compartment, inside another "Membrane" compartment. The nodes of the diagram can be assigned to any of these. For layout purposes this means that each node must appear within the boundaries of the compartment to which it is assigned.

8.2 Ports and Orientability

A challenge in laying out SBGN diagrams is the presence of ports, and the fact that logic and process nodes need to be assigned an orientation as illustrated in Figure 8.3. This section introduces a formal model for dealing with these issues. The formalism is in places quite heavy, but it is necessary in order to make precise our definition of a *well-oriented* node (Section 8.2.4). This notion is fundamental in the design of the MCGL algorithm.

Ports play three roles in a layout process:

1. *Routing*: Once a position is assigned to it, a port represents an end point for a connector routing algorithm.

2. *Node Positioning*: When choosing positions for nodes, placement of port p on a particular side of node v makes it preferable that neighbours connecting at p go on the same side.

3. *Logical*: Ports provide a way to partition the neighbours of a given node into groups, and a way to express constraints on connection positions.

For the DiAlEcT layout developed in this chapter the first of these roles, routing, is simply handled by the POLY-ROUTE operation of ADAPTAGRAMS at the end of the layout process. By that time port positions will have been chosen, and these can simply be passed to the POLY-ROUTE operation as "routing pins".

Regarding the second point, node positioning, we make use of the technique of "port dummy nodes" we developed in collaboration with Rüegg for data flow diagrams [RKD+14]. The technique is reviewed in Section 8.2.1.

As for the logical model, SBGN presents a tricky special case that cannot adequately be handled by standard port models in the literature. This motivates our development of a special model tailored to SBGN ports in Section 8.2.3. Section 8.2.2 provides additional supporting definitions, and finally Section 8.2.4 defines the notion of a *well-oriented* orientable node.

8.2.1 Port Dummy Nodes

In work together with lead author Rüegg [RKD+14] we applied the ACA process of Chapter 3 to *data flow diagrams*. The algorithm was called CoDaFlow, for "Constrained Data Flow" layout. Data flow diagrams are commonly used to model movement of data between components in complex hardware and software systems [LNW03]. Complex systems are modelled graphically by composing *actors*, i.e. reusable block diagrams representing well-defined pieces of functionality. Data flow is shown by directed edges from the source port where the data is constructed to the target port where the data is consumed. By convention, the edges are drawn orthogonally and the ports are fixed in position on the actors' boundaries.

While there were pre-existing techniques based on the Sugiyama method [STT81] for laying out data flow diagrams [SSvH14], the purpose of CoDaFlow was to try an approach based on stress minimisation and the ACA process. We believed this would result in more

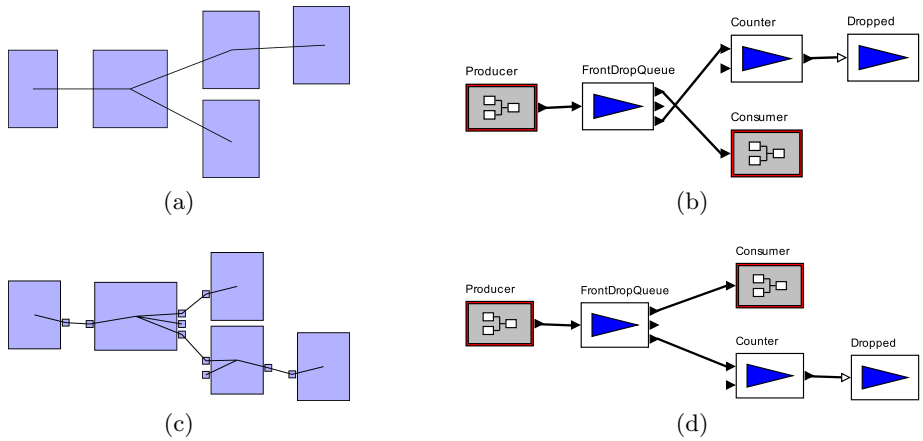


Figure 8.5: Awareness of ports is important to achieve good node positioning. (a) and (c) show internal representations of what is passed to the layout algorithm, (b) and (d) show the resulting drawings of data flow diagrams from [RKD+14]. (a) is unaware of ports and yields node positions that introduce an edge crossing in (b). In (c) ports are considered and the unnecessary crossing is avoided in (d). Note, however, while the chance is higher that (d) is crossing-free, it is not guaranteed.

compact layouts, and our results bore this out [RKD+14]. A major component of this work was determining how to use stress minimising layout to compute port positions.

We began with a simple formal model of a graph with ports, consisting of a quadruple $G = (V, E, P, \pi)$, where nodes $v \in V$ were connected by edges $e \in E \subseteq P \times P$ through ports $p \in P$, and $\pi : P \to V$ mapped each port p to what we called the "parent" node $\pi(p) \in V$ to which it belonged. An edge $e = (p_1, p_2)$ was directed, outgoing from port p_1 and incoming to port p_2.

The key idea in the layout process was then to create a small node to represent each port, called a *port node* or *port dummy*, as in Figure 8.5c. If D is the set of all these, and $\delta : P \to D$ maps each port to the dummy node that represents it, we construct a new graph $G' = (V', E')$ where $V' = V \cup D$, and

$$E' = \{(\delta(p_1), \delta(p_2)) : (p_1, p_2) \in E\} \cup \{(\pi(p), \delta(p)) : p \in P\}$$

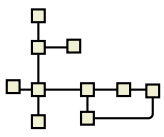
includes one edge representing each edge of the original graph, and an edge connecting each port dummy to its parent node.

The use of port nodes allows the constrained stress-minimising layout algorithm to untangle the graph while being aware of relative port positions. This tends to result in fewer edge crossings, as illustrated in Figure 8.5. We will use this technique in our DiAlEcT layout for SBGN. In particular we create a port node for each port of a submap node (Section 8.1.2), and two port nodes for each logic and process node, one for each of their spikes. We connect modulators directly to process nodes.

8.2.2 Directions

Before our port model for SBGN diagrams can be described in Section 8.2.3 we need to introduce some notation for working with compass directions in the plane.

As is conventional when discussing plane graphics, our positive x-axis extends to the right, while our positive y-axis extends downward, not up. Consistent with this, when we speak of rotational processes we will always think of clockwise rotation as the forward direction, this being the direction from the positive x-axis toward the positive y-axis. Accordingly the four quadrants of the plane are numbered 1, 2, 3, and 4 in the order in which they are encountered in clockwise rotation starting from the positive x-axis. Thus, counter to the conventional mathematical numbering, Quadrant 1 is in the lower-right



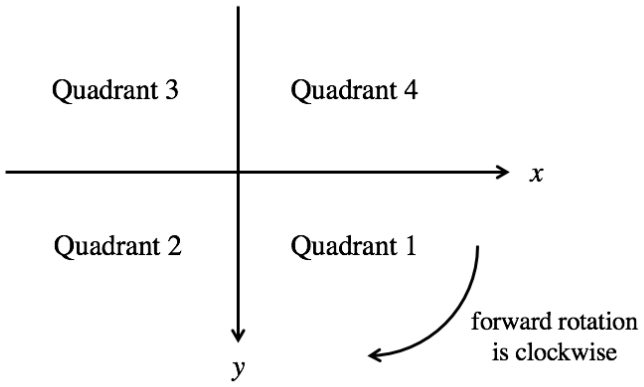


Figure 8.6: For graphics, the positive y-axis points downward. Accordingly, the forward direction for rotation is clockwise, and the four quadrants are numbered clockwise, starting from the lower-right.

instead of the upper-right. This also means that lines of positive slope run from upper left to lower right, while lines of negative slope run from upper right to lower left. See Figure 8.6.

We will be interested in the cardinal compass directions, and we will usually list these in the order $\mathbb{E}, \mathbb{S}, \mathbb{W}, \mathbb{N}$, since this is the clockwise progression starting from the positive x-axis. Note that, despite the downward pointing y-axis, $\mathbb{N}$ still means "above"; thus, one object is north of another when its y-coordinate is smaller, not larger.

We will also be interested in the so-called ordinal directions, which we denote by combinations of the cardinal letters: $\mathbb{SE}$, $\mathbb{SW}$, $\mathbb{NW}$, $\mathbb{NE}$. We introduce the notation $\mathscr{C}_4 = \{\mathbb{E}, \mathbb{S}, \mathbb{W}, \mathbb{N}\}$ for the set of cardinal compass directions, and $\mathscr{C}_8 = \mathscr{C}_4 \cup \{\mathbb{SE}, \mathbb{SW}, \mathbb{NW}, \mathbb{NE}\}$ for the full set of all cardinal and ordinal directions. We sometimes need to talk about rotating these directions and so define the map

$$\mathrm{cw}_4 : \mathscr{C}_4 \to \mathscr{C}_4$$

rotating each cardinal direction clockwise 90 degrees, and similarly the map

$$\mathrm{cw}_8 : \mathscr{C}_8 \to \mathscr{C}_8$$

rotating each direction clockwise 45 degrees. We will use these maps with exponents to indicate repeated application, so that for example $\mathrm{cw}_8{}^3$ means clockwise rotation by 135 degrees.

When considering two points p, q in the plane we will often be concerned with the compass direction from p to q in a categorical sense. If the vector $q - p$ lies on the positive x-, positive y-, negative x-, or negative y-axes, then we say that the direction from p to q is $\mathbb{E}, \mathbb{S}, \mathbb{W}$ or $\mathbb{N}$, respectively. Otherwise the vector lies in one of the open Quadrants, 1, 2, 3, or 4, in which case we say that the direction from p to q is $\mathbb{SE}$, $\mathbb{SW}$, $\mathbb{NW}$, or $\mathbb{NE}$, respectively. We refer to this as the *categorical direction from p to q*, and denote it by

$$\mathrm{catdir}(p, q) \in \mathscr{C}_8.$$

In fact the catdir function is polymorphic and its arguments may be points, nodes, or anything else that has an assigned position. For example if u, v are any nodes that have a current position during a layout process, then by $\mathrm{catdir}(u, v)$ we will mean the categorical direction from the centre point of u to that of v.

8.2.3 SBGN Port Model

Section 8.2.1 mentioned the simple, generic model of a directed graph with ports $G = (V, E, P, \pi)$ that was used in our CoDaFlow algorithm. Other similar models have been proposed in the literature, for example by Spönemann et al. [SFVHM09], who also described four types of port constraints of increasing specificity:

1. ports are free to go anywhere on the perimeter of the node;
2. each port is constrained to lie on one of the four sides of the node (and nodes are represented by rectangular bounding boxes of fixed dimensions);
3. ports are constrained to sides, and their (clockwise) order is also fixed;
4. each port must go in a fixed position on the perimeter of the node.

For SBGN layout these models—of both ports and constraints—fall short in two ways. To begin with, ports represent needless clutter for the many SBGN nodes that have no ports at all (such as entity pool nodes), whereas in the existing models *all* connections are mediated via ports.

A more serious issue is that, while SBGN logic and process nodes do have something that we think of as "ports," the simple models mentioned above fail to capture the tricky rules that govern these. To begin with, logic and process nodes do not even have a fixed bounding box; instead the dimensions of their bounding boxes change depending on how we decide to orient the nodes and position their ports (again refer to Figure 8.3). Furthermore while some of a process node's neighbours attach to the ports at the ends of the two spikes, other neighbours, namely modulators, attach in a disjunctive way: they may attach to either of two (other) ports, representing the two sides of the process node box that do not have spikes.

In order to handle these complications we introduce a special formal model for SBGN graphs. To begin with there are a set V of vertices or nodes, a set E of edges or arcs, and a set P of ports. Each edge is represented by an ordered pair and its endpoints may be nodes or ports; therefore $E \subseteq (V \cup P) \times (V \cup P)$. Introducing the set

$$\mathcal{T} = \{\mathsf{EN}, \mathsf{PN}, \mathsf{LN}, \mathsf{SN}, \mathsf{CA}, \mathsf{PA}, \mathsf{MA}, \mathsf{LA}, \mathsf{EA}\}$$

of SBGN layout types as given in Table 8.1 above, our formal model also includes a map $\tau : V \cup E \to \mathcal{T}$. This map assigns a type to each node and arc in the graph, and must satisfy the obvious condition that for all $v \in V$ we have $\tau(v) \in \{\mathsf{EN}, \mathsf{PN}, \mathsf{LN}, \mathsf{SN}\}$, while for all $e \in E$ we have $\tau(e) \in \{\mathsf{CA}, \mathsf{PA}, \mathsf{MA}, \mathsf{LA}, \mathsf{EA}\}$. Inverse images under the map τ can then be used to refer to the set of all nodes of a given type, or arcs of a given type. For example $\tau^{-1}\,\mathsf{PN} \subseteq V$ is the set of all process nodes, and $\tau^{-1}\,\{\mathsf{PN}, \mathsf{LN}\} \subseteq V$ the set of all orientable nodes in a given SBGN network.

Process and logic nodes have predefined sizes but other node types do not; therefore we have *width* and *height* maps

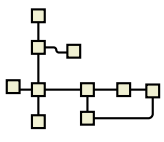
$$w : \tau^{-1}\,\{\mathsf{EN}, \mathsf{SN}\} \to \mathbb{R}_+$$

$$h : \tau^{-1}\,\{\mathsf{EN}, \mathsf{SN}\} \to \mathbb{R}_+$$

assigning dimensions to these nodes.[1]

As in the CoDaFlow model there is a *parent* map $\pi : P \to V$. Only submaps, logic nodes, and process nodes have ports. Therefore $\pi(P) \subseteq \tau^{-1}\,\{\mathsf{SN}, \mathsf{LN}, \mathsf{PN}\}$. If v is any node

[1] As for the indeterminate bounding boxes of logic and process nodes mentioned earlier, the only variable is how the node is to be oriented; once that is decided then the dimensions of the box are known.



then we will refer to any other node u as a *neighbour* of v if and only if there exists an edge $e = (r, s) \in E$ whose endpoints r, s are either u and v themselves, or ports belonging to these nodes, or any mixture thereof; in other words $\{r, s\} \subseteq \{u, v\} \cup \pi^{-1}\{u, v\}$.

While submap nodes can have any number of ports, logic nodes always have precisely two ports, and process nodes precisely three. We introduce special notation with which to refer to these. If v is a logic node then we denote its two ports by $p_v^{(-1)}$ and $p_v^{(1)}$. If v is a process node then we likewise denote two of its ports by $p_v^{(-1)}$ and $p_v^{(1)}$, while by $p_v^{(0)}$ we denote its third port—a kind of "pseudo-port" to represent connections to modulators.

For all orientable nodes v (i.e. both logic and process nodes) we refer to the two ports $p_v^{(-1)}$, $p_v^{(1)}$ as *outer ports*, while for process nodes we refer to the port $p_v^{(0)}$ as the *centre port*. In terms of the final graph drawing the outer ports are, naturally, those at the ends of the "spikes" in the glyphs for logic and process nodes. Meanwhile the centre port of a process node is where modulators connect disjunctively. Before the layout is complete it will be decided to which of the two available sides (without spikes) each modulator connects.

If v is any orientable node and u a neighbour of v, then by $p_v(u)$ (read "port of v for u") we denote the port of v to which u connects. This slight overloading of the "p_v" notation should not cause any confusion. For example $p_v(u) = p_v^{(1)}$ simply indicates to which spike port of node v node u connects.

Like nodes, ports must be assigned positions during the layout process. As soon as a process node v is assigned a position its centre port $p_v^{(0)}$ has that same position, but its outer ports $p_v^{(-1)}$, $p_v^{(1)}$ do not have positions until the node v has been oriented. The same goes for the ports of a logic node. Accordingly, we define the following *orientation* function for orientable nodes v:

$$\mathrm{ori}(v) = \mathrm{catdir}(v, p_v^{(1)}),$$

i.e. the orientation of node v is the categorical direction from the centre of v to its outer port $p_v^{(1)}$.

Finally, as for the port constraints that may (optionally) be imposed by the user before the layout process begins, we allow different options for submaps and for orientable nodes. For submaps the user may specify port constraints at any of the four levels defined by Spönemann et al. as presented above. Meanwhile for each orientable node v the user may specify a nonempty subset $C_v \subseteq \mathscr{C}_4$ indicating the acceptable orientations for v, i.e. imposing the constraint that $\mathrm{ori}(v) \in C_v$.

8.2.4 Well-Orientation

Most of the formalism just introduced is in service of a very important notion for SBGN layout, that the orientable nodes be what we will call *well-oriented*.[2] Roughly, what we will mean by this is that the orientable node's neighbours lie on the appropriate sides of it, given the ports to which they attach, and given its orientation. See Figure 8.7

In order to make this notion precise we begin by defining a function $\Omega : \mathscr{C}_8 \to \wp\mathscr{C}_4$ from compass directions to sets of cardinal directions, as in Table 8.2. We can then use the Ω function to define well-orientation in the following way. Suppose v is an orientable node and u a neighbour of v. We define the predicate $\mathrm{wo}(v, u)$, read v *is well-oriented*

[2]Mimicking the tradition of Cantorian set theory—in which a set may be *well-ordered* and we may speak of a given order relation being a *well-ordering*—here we will speak of a given orientation being a *well-orientation*, and we will speak of the act of *well-orienting* a node. While not standard English constructions, such usages are firmly established in the mathematical tradition.

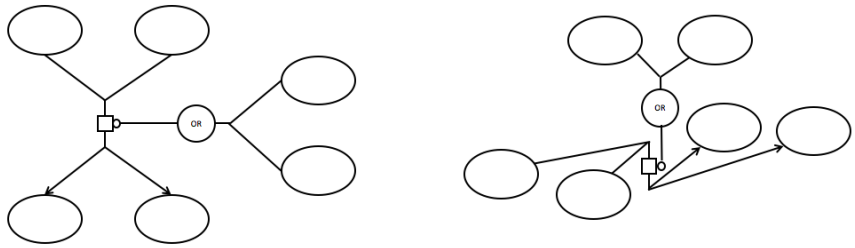


Figure 8.7: In the layout on the left the process node is well-oriented because its neighbours lie on the appropriate sides, considering the ports to which they attach. In the layout on the right neighbouring nodes lie on inappropriate sides, and the process node is not well-oriented with respect to its neighbours.

Direction d	Directions in $\Omega(d)$			
	$\mathbb{E}$	$\mathbb{S}$	$\mathbb{W}$	$\mathbb{N}$
$\mathbb{E}$	✓	✓		✓
$\mathbb{SE}$	✓	✓		
$\mathbb{S}$	✓	✓	✓	
$\mathbb{SW}$		✓	✓	
$\mathbb{W}$		✓	✓	✓
$\mathbb{NW}$			✓	✓
$\mathbb{N}$	✓		✓	✓
$\mathbb{NE}$	✓			✓

Table 8.2: Definition of the Ω function for well-orienting. In words, if d is a cardinal direction then $\Omega(d)$ contains all but the opposite direction, while if d is an ordinal direction then $\Omega(d)$ consists of the two cardinal components of d.

with respect to u, as follows:

$$\mathrm{wo}(v,u) \equiv \begin{cases} \mathrm{catdir}(v,u) \notin \left\{\mathrm{catdir}(v,p_v^{(j)}) : j = \pm 1\right\} & \text{if } u \in \mathrm{Mod}(v) \\ \mathrm{catdir}(v,p_v(u)) \in \Omega(\mathrm{catdir}(p_v(u),u)) & \text{if } u \notin \mathrm{Mod}(v). \end{cases} \tag{8.1}$$

Unpacking this, the relation depends on whether the neighbour u is a modulator of v or not. If u is a modulator then well-orientation simply means that the categorical direction from v to u is not among the categorical directions from v to its two outer ports; see Figure 8.8, right. Meanwhile if u is not a modulator then it attaches to one of v's outer ports, and then well-orientation requires that the direction from v to this port be among those in the Ω set for the direction from this port to u; see Figure 8.8, left, and Table 8.2.

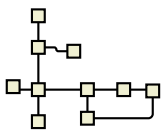
We say that v *is well-oriented* and write $\mathrm{wo}(v)$ if v is well-oriented with respect to all of its neighbours.

Furthermore, it will be useful to be able to speak hypothetically about whether v *would be* well-oriented with respect to u if u were placed at a given point q in the plane, and for this we introduce another form of the wo predicate:

$$\mathrm{wo}(v,u,q) \equiv \begin{cases} \mathrm{catdir}(v,q) \notin \left\{\mathrm{catdir}(v,p_v^{(j)}) : j = \pm 1\right\} & \text{if } u \in \mathrm{Mod}(v) \\ \mathrm{catdir}(v,p_v(u)) \in \Omega(\mathrm{catdir}(p_v(u),q)) & \text{if } u \notin \mathrm{Mod}(v) \end{cases} \tag{8.2}$$

which we read as, v *is well-oriented with respect to* u *at* q. Note that $\mathrm{wo}(v,u) \equiv \mathrm{wo}(v,u,u)$.

Consider for example a process node v with $\mathrm{ori}(v) = \mathbb{E}$, and suppose $p_v^{(1)}$ is v's production port, i.e. the production arcs connected to v attach at port $p_v^{(1)}$ on its east side. See



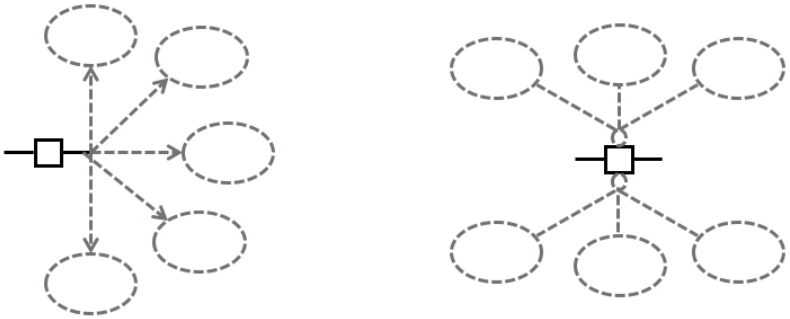


Figure 8.8: Suppose a process node v is oriented with its production port on the east. Then v is well-oriented with respect to a product neighbour u if and only if the categorical direction from port $p_v^{(1)}$ to node u is anything other than $\mathbb{W}$, $\mathbb{SW}$ or $\mathbb{NW}$ (left). Meanwhile v is well-oriented with respect to a modulator w if and only if the categorical direction from v to w is neither $\mathbb{E}$ nor $\mathbb{W}$ (right).

Figure 8.8. If u is a product neighbour of v, then v is well-oriented with respect to u if and only if the categorical direction d from $p_v^{(1)}$ to u is such that $\mathbb{E} \in \Omega(d)$. By Table 8.2, this means d can be any direction except $\mathbb{W}$, $\mathbb{SW}$ or $\mathbb{NW}$ (consider the column headed "$\mathbb{E}$" in the table). On the other hand, if w is a modulator of v then v is well-oriented with respect to w if and only if the categorical direction from v to w is neither $\mathbb{E}$ nor $\mathbb{W}$.

Well-Orientation in MetaCrop

Existing SBGN diagrams from the MetaCrop database show very few process nodes that are not well-oriented; i.e. well-orientation seems to be a high-priority layout goal in the MetaCrop dialect. It seems reasonable that well-orientation would be favoured for the sake of visual ease in path-following, and it is likely a feature of SBGN diagrams whatever the dialect. Recall that in the Orthowontist study too, path-following seemed a reasonable hypothesis to explain users' preference for creating bends in edge routes rather than at nodes. Accordingly, creating well-orientations will be a major goal of Phase-A, as discussed in Section 8.4.

8.3 Phase-D

Having introduced the special features of SBGN diagrams in Section 8.1 and developed a formalism for dealing with ports, orientable nodes, and well-orientation in Section 8.2, we may now set about applying the DiAlEcT framework to develop the MCGL layout algorithm.

We begin with Phase-D. Following the guidelines given in Section 7.1, Phase-D involves both decomposition and distribution.

Skeleton and Orbits

Recall that the HOLA algorithm begins with a peeling process (Section 5.3.1) in which the leaves are repeatedly stripped off the graph until none remain, thereby decomposing the graph into *core* and *trees*. It is then only the core graph that is distributed by stress minimisation, making HOLA's Phase-D.

For SBGN diagrams a similar approach is called for, but with a modification. This time only the *first* layer of leaves is stripped off the graph. We refer to these leaves as the *satellites*, and to each set of satellites having a common neighbour in the full graph as an *orbit*. The terms "satellite" and "orbit" are chosen to reflect the typical layout of these

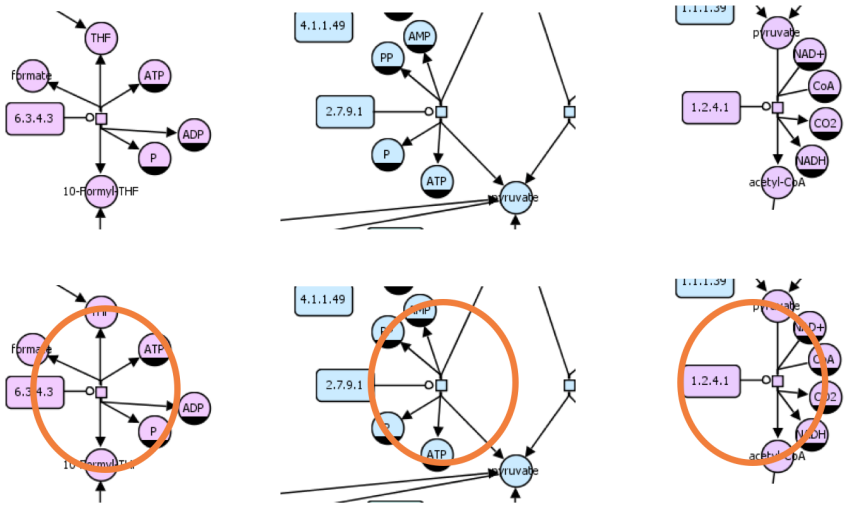


Figure 8.9: In MetaCrop diagrams leaf nodes ("satellites") are typically arranged on roughly elliptical arcs around their parent node.

nodes in MetaCrop diagrams, on a roughly elliptical arc around their parent node in the skeleton graph. See Figure 8.9.

After the orbits are removed, the remaining graph is called the *skeleton*. Thus, skeleton and orbits play similar roles in MCGL as core and trees did in HOLA. Analogous to HOLA, the distribution via stress minimisation in Phase-D of MCGL operates only on the skeleton.

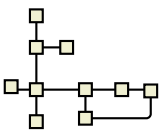
There are two reasons for taking this approach in which only the first layer of leaves is peeled off, the first reason being the typically large percentage of leaves in SBGN diagrams. (Recall for example Figure 6.16d, page 95.) This is normal, since human designers deliberately use *cloning* (Section 6.4.2) to create many of these leaves and achieve the desired level of sparseness in the graph. After peeling just one layer of leaves it is common for a very simple skeleton graph to remain.

The second reason for taking this approach is that in the observed MetaCrop layouts it was normal for the skeleton subgraph to contain trees that were *not* laid out in hierarchical form, i.e. were not given the *hierarchical symmetric tree* arrangement (Table 7.2). A simple and frequently occurring case is when the skeleton is a mere chain, for which a straight layout is preferred, or a Y-shaped graph, for which a roughly Y-shaped layout is preferred. These layouts are best achieved by simply applying a DESCEND operation to the skeleton.

Distributing in Compartments

The ADAPTAGRAMS layout library (Section 2.2.4) supports *clustering*. Nodes can be added to clusters, and the latter can be nested inside one another. On each iteration of the DESCEND process containment constraints are generated between the borders of each cluster, and non-overlap constraints are generated between siblings (both nodes and clusters) within each cluster.

This allows the hierarchical structure of SBGN compartments to be represented. However, starting from random initial positions, a few passes are needed in order to get the nodes properly separated into their respective clusters. See Figure 8.10. For this we employ a technique we call *cluster gravity*, in which a dummy node is added to each cluster, attached to every other node in that cluster. This technique is well-known (see for example the survey by Brockenauer and Cornelsen [BC01]) but it is worth examining here how to combine it with the ADAPTAGRAMS overlap-prevention technique, where constraints can be automatically generated just between sibling clusters, or between all sibling clusters and nodes.



Pass	Cluster Gravity	Overlap Prevention
1	on	off
2	off	just clusters
3	off	clusters and nodes

Table 8.3: Sequence of DESCEND operations achieving distribution in the presence of SBGN compartments (clusters)

Graph Substructure	Graph Substructure Arrangement
parallel chains	parallel chain group (PCG)
face	regular polygonal face (RPF)
orbit	sector-partitioned orbit (SPO)

Table 8.4: Graph substructures and their arrangements in DiAlEcT layout for MetaCrop SBGN

Cluster gravity and overlap prevention are combined in three DESCEND passes as described in Table 8.3 in order to achieve the distribution of the skeleton graph. The first pass with cluster gravity and no overlap prevention allows nodes to pass through one another and reach a position in which the clusters form disjoint convex hulls. Then in the second pass cluster overlap prevention can be activated without conflict; moreover this overlap prevention means that cluster gravity is no longer needed to keep clustered nodes together. On the other hand, it is important that for this second pass there be no overlap prevention between *nodes* (as opposed to clusters), so that these can pass through one another and nodes can reach a well-distributed position within each cluster. Finally overlap prevention between nodes is switched on in a third pass, achieving a basic overlap-free distribution of nodes within their appropriate SBGN compartments. This completes Phase-D.

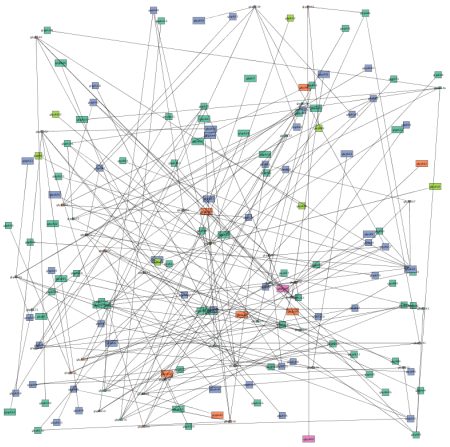
8.4 Phase-A

Following the summary given in Section 7.2.3, the tasks in designing Phase-A are to define the graph substructure arrangements (GSAs), and to describe the constraint generation loop (CGL). These must conform to the five arrangement principles identified on page 105.

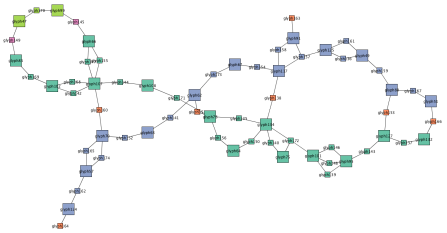
In fact just three new GSAs are identified for the MetaCrop SBGN dialect, and these are listed in Table 8.4. The first of these, the parallel chain group, is the subject of Section 8.4.1, while the regular polygonal face is covered in Section 8.4.2, and the sector-partitioned orbit in Section 8.4.3. The CGL is described in Section 8.4.4.

Each section introducing a new GSA follows a consistent format: it begins by introducing and defining the new GSA; next there is a section headed "Structural and Geometric Variants" where the variants are defined; next comes "Constraints" describing how a chosen arrangement is to be enforced; finally a section headed "Variant Selection" tells how we choose a variant during the constraint generation loop. Describing the variant selection routine proves to be the most complex part of defining a GSA, and requires an understanding both of the possible variants and of the constraints that will be used to enforce them.

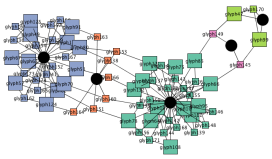
The arrangement protocols for the new GSAs include one major departure from the policies of HOLA's Phase-A: In HOLA, in honour of arrangement principle **Faithfulness** (recall the enumeration of arrangement principles on page 105) there was no time at which we deliberately altered the rotation system, i.e. the cyclic ordering of neighbours of a given node. In MCGL on the other hand, the importance of well-orienting (Section 8.2.4) is deemed so high as to outweigh the concerns expressed in Section 5.2.2 (page 62), and



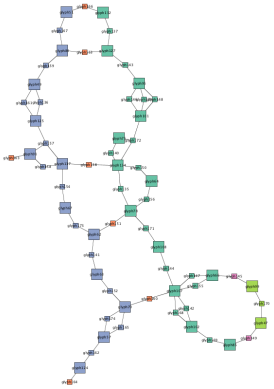
(a) We start with random initial positions. Nodes are colour-coded by SBGN compartment.



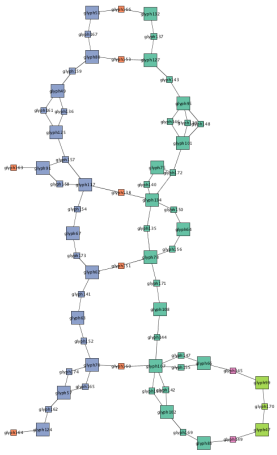
(b) If we applied DESCEND immediately we could get into trouble. In this particular instance the layout winds up "twisted" (the blue and green clusters cross one another). Attempting to now enforce cluster non-overlap would simply lead to conflicting constraints.



(c) Instead, we begin with cluster gravity. A new node (black, circular) is added to each cluster, connected to every node. Then DESCEND tends to separate the clusters as shown here.

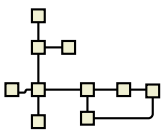


(d) For illustrative purposes only (this is not a step in our process) we examine what happens after the cluster gravity nodes are removed and DESCEND is applied without any constraints. The nodes are well distributed, but the bounding boxes of the clusters still overlap.



(e) Now switching on cluster overlap prevention, the clusters are well separated from one another, so that SBGN compartment boundaries can now be drawn in the diagram, as desired.

Figure 8.10: Clustering techniques and careful generation of non-overlap constraints are required in order to separate nodes into their SBGN compartments during the distribution phase. In this example we look at the skeleton subgraph of the glycolysis-gluconeogenesis pathway shown on the left in Figure 8.11.



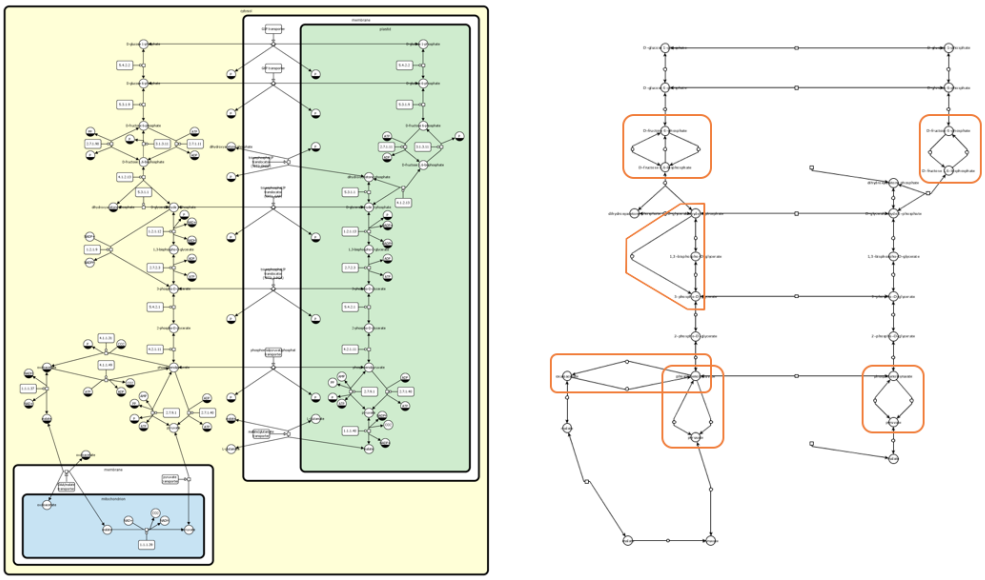


Figure 8.11: MetaCrop layout of glycolysis-gluconeogenesis pathway is shown on the left. On the right are noted several parallel chain groups visible in the skeleton subgraph.

making nodes well-oriented may sometimes require the imposition of constraints to alter a given node's cyclic neighbour ordering.

This case illustrates what should be a general feature of future applications of the DiAlEcT framework: the needs of a particular layout dialect must in general be weighed against the five arrangement principles, and trade-offs must be made. In MCGL we will continue to honour **Faithfulness** in other ways, in particular in the *cost functions* we use when selecting geometric variants, as described in the following subsections.

8.4.1 Parallel Chain Groups

Layouts in the MetaCrop SBGN dialect emphasise series-parallel structure in the skeleton via special node arrangements. In order to locate the relevant graph substructures we begin by identifying all *chains* in the skeleton graph, as defined in Section 4.4. Once all chains have been identified we group them into *parallel chain groups*. A parallel chain group (PCG) is a maximal set of two or more *coterminal* chains, where two chains C_1, C_2 are said to be coterminal when their boundaries (see Section 4.4) are equal and nonempty:

$$\partial C_1 = \partial C_2 \neq \varnothing.$$

Several parallel chain groups are visible in Figure 8.11, and the dialect-conventional way of arranging these in MetaCrop is demonstrated.

Meanwhile *lone chains* do not enter into PCGs because they do not share their terminal nodes with any other chains. Lone chains may receive *orthogonally routed chain* or *axis-aligned pair* GSAs (Table 7.2), and the choice may be left as a configurable option.

Among PCGs there are two exceptional cases to note. To begin with there is what we call a "pseudo-PCG". As depicted in Figure 8.12, there is a pseudo-PCG for every chain C for which $|\partial C| = 1$, i.e. each chain C for which there is a single node v that C attaches to at both its ends. As hand-made layout examples like Figure 8.12 show, it is desirable to give a pseudo-PCG the same diamond-shaped layout as a proper two-chain PCG. This arrangement is achieved by selecting a node u as near as possible to the centre of C and treating u as a "pseudo terminal", then configuring C as if it consisted of two chains running between the terminals u and v.

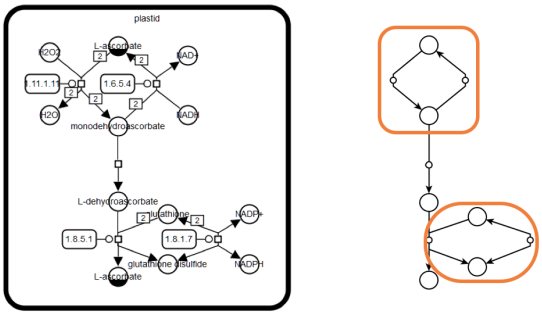


Figure 8.12: MetaCrop layout of ascorbate-glutathione cycle pathway is shown on the left. On the right are noted two pseudo-PCGs in the skeleton subgraph.

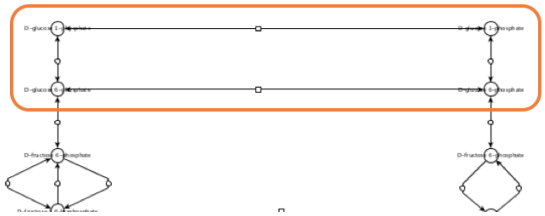


Figure 8.13: A close examination of the nodes at the top of the skeleton from Figure 8.11 reveals that the boxed nodes actually constitute a PCG. However, in such a case the conventional PCG arrangement pattern highlighted in Figure 8.11 is to be avoided, as a different metaphor dominates. The "transporter reactions" lying in "membrane" compartments, whose neighbours span multiple compartments, are better arranged linearly as in this figure.

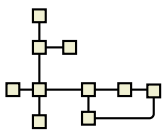
The other exceptional case is a PCG that spans multiple SBGN compartments. We ignore these, and do not attempt to give them any special arrangement. The reason for this relates to the possibility of their containing what we call "transporter reactions", and this is therefore a *semantic* consideration, tailored to the application domain of SBGN diagrams. Transporter reactions move entities across cell or organelle membranes, and are represented by process nodes whose neighbours lie in distinct compartments. Observation of MetaCrop layouts suggests it is desirable that such processes and their neighbours be configured linearly in a direction perpendicular to the compartment boundaries that they cross, as illustrated in Figure 8.13. We leave it up to lone chain GSAs to achieve this, instead of arranging such chains as parts of PCGs.

Structural and Geometric Variants

The geometric variants for PCGs are quite simple: there are only four, corresponding to the direction in which the PCG is aligned. The structural variants are more complicated, involving the choice of the order in which the several chains are to be arrayed beside one another, and the choice of which of the chains, if any, is to be aligned with the terminal nodes.

We arbitrarily designate one of the terminal nodes of a given PCG as the *source* terminal s, and the other as the *target* terminal t. Then the four geometric variants place t in one of the four cardinal compass directions from s. See Figure 8.14.

In all variants, nodes s and t are aligned. We refer to the direction from s to t as the *axial direction* of the PCG. Meanwhile the dimension in which the coordinates of nodes s and t differ is called the *axial dimension*, while that in which they are the same is the *transverse dimension*. For example, when the axial direction is $\mathbb{S}$ as in the case far-left in Figure 8.14, then the axial dimension is y and the transverse dimension x.



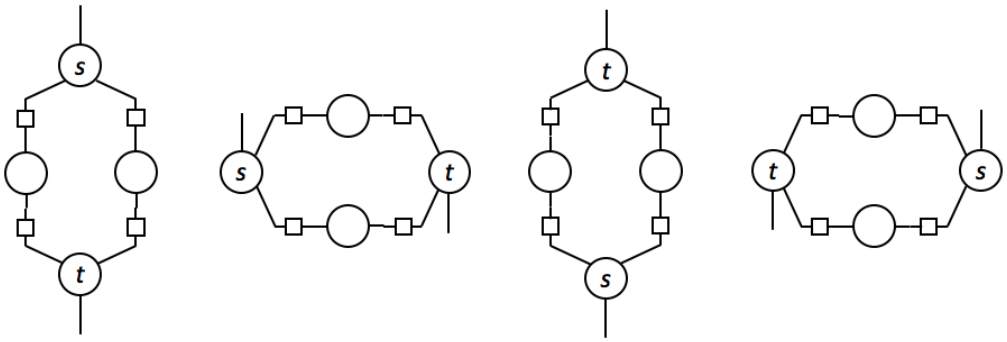


Figure 8.14: After arbitrarily designating the terminal nodes of a PCG as "source" s and "target" t, we can orient the PCG in any of the four cardinal compass directions. From left to right in this figure, the PCG points south, east, north, and west. In a case like the one illustrated here, in which nodes s and t were aligned vertically with their respective external neighbours, MCGL would attempt to create either the north or south orientation, i.e. the first or third options in this figure reading from the left.

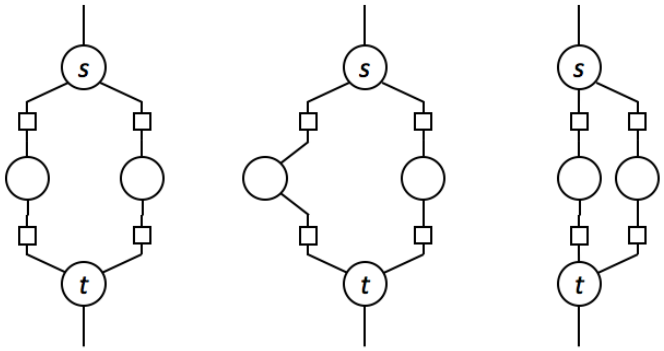


Figure 8.15: Once a direction is chosen for a PCG, there may be many different ways to arrange the chains within it. In MCGL we would select the leftmost arrangement in this figure as the standard structural variant.

As for structural variation, there are many conceivable ways in which we could arrange each of the chains within the PCG, but according to **Standardisation** we select a simple standard alternative, and leave the variations up to Phase-T.

To begin with, we ensure that the orientation of each O-node (orientable node) occurring within the PCG is compatible with the direction in which the whole PCG points, i.e. that the spikes of the O-node are aligned in the axial dimension. For example if the axial direction is $\mathbb{S}$ then each O-node v in the PCG must satisfy $\text{ori}(v) \in \{\mathbb{N}, \mathbb{S}\}$. This confines us to configurations like those illustrated in Figure 8.15. Furthermore we opt to make each of the chains in the PCG perfectly straight. This eliminates cases like the one illustrated in the centre of Figure 8.15.

Finally there is the question as to how the chains should be distributed in the transverse dimension. For example, for a two-chain PCG we could balance the chains as on the left of Figure 8.15, or we could put one chain in the centre, and the other off to one side, as on the right of that figure. However, while the latter option could conceivably be desirable for semantic reasons in a given case, we leave such choices up to Phase-T; in Phase-A we honour **Standardisation** by simply working toward the structural good of symmetry as far as permitted by the number of chains in the PCG and the length of each. This leaves us to order the chains in essentially the same way we ordered the c-trees in the Manning-Atallah algorithm used to create the *hierarchical symmetric tree* (HST) arrangement (Table 7.2). As with HSTs, for PCGs we again favour putting longer chains closer to the centre.

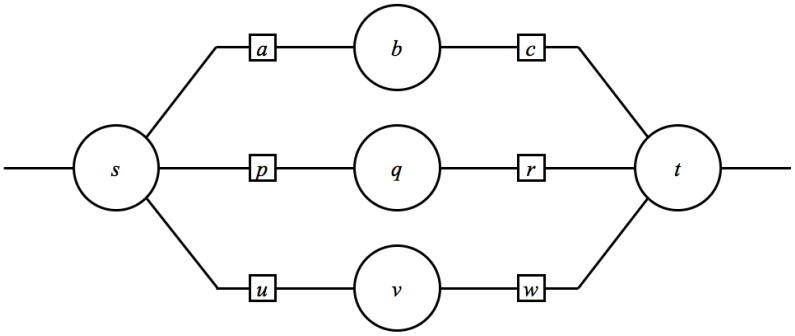


Figure 8.16: A PCG with axial direction $\mathbb{E}$, with a centre chain

Choosing the ordering of the chains within a PCG means choosing the cyclic ordering of the neighbours of the two terminal nodes; however, like HSTs, PCGs can be arranged essentially in isolation from the remainder of the graph, and for this reason there is no special need here to preserve cyclic ordering, i.e. no essential conflict with **Faithfulness**.

Constraints

The constraints enforcing a PCG arrangement are easy to define, and we start by considering an example. For the case illustrated in Figure 8.16 with axial direction $\mathbb{E}$, the entries of a relative constraint matrix M (Section 7.4) would be as follows:

1. The source and target nodes s and t are aligned in the axial direction. Thus $M(s,t) = \mathbb{E}$.

2. The internal nodes of each chain (i.e. all but the terminal nodes) are aligned in the axial direction. Thus $M(a,b) = M(b,c) = M(p,q) = M(q,r) = M(u,v) = M(v,w) = \mathbb{E}$.

3. The first and last nodes of each chain are placed by separation constraint on the proper side of their neighbouring terminal node. Thus $M(s,a) = M(s,p) = M(s,u) = \mathbb{R}$ and $M(t,c) = M(t,r) = M(t,w) = \mathbb{L}$.

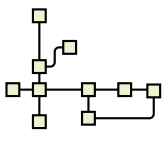
4. If there is a centre chain, then its nodes are aligned with s and t in the axial direction. Thus $M(s,p) = \mathbb{E} = M(r,t)$ and this overrides the definitions of these entries given in the last item.

5. The first (in order from source to target) internal nodes of the several chains are given ordering constraints in the transverse dimension in order to keep the chains aligned beside one another in the desired order. Thus $M(a,p) = M(p,u) = \mathbb{D}$.

In order to give the precise definitions of the constraints in general, let s and t again be the source and target terminal nodes. Let the chains of the PCG be $C_1, C_2, \ldots, C_n$ in the order in which they are encountered in clockwise rotation around the source node s. For each chain C_i, let $v_i^{(0)}, v_i^{(1)}, \ldots, v_i^{(m_i)}$ be all its nodes, including its terminals, ordered from source to target; thus $v_i^{(0)} = s$ and $v_i^{(m_i)} = t$ for all i. Let $\mathbb{A} \in \mathscr{C}_4$ be the axial direction of the PCG and let $\mathbb{B}$ be the lateral weakening of $\mathbb{A}$ (see Table 7.5, page 109). Let $\mathbb{C} = \mathrm{cw}_4(\mathbb{B})$. Then the constraints are as follows:

1. $M(s,t) = \mathbb{A}$.

2. For $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m_i - 2$, $M(v_i^{(j)}, v_i^{(j+1)}) = \mathbb{A}$.



3. For $i = 1, 2, \ldots, n$, $M(s, v_i^{(1)}) = \mathbb{B} = M(v_i^{(m_i-1)}, t)$.

4. If there is a centre chain C_{i_0} then $M(s, v_{i_0}^{(1)}) = \mathbb{A} = M(v_{i_0}^{(m_{i_0}-1)}, t)$, and this overrides the previous rule.

5. For $i = 1, 2, \ldots, n-1$, $M(v_i^{(1)}, v_{i+1}^{(1)}) = \mathbb{C}$.

Note that, except where exact alignments are desired, the separation constraints employ inequalities, not equations, according to **Minimality**.

Variant Selection

The approach for assessing a potential geometric variant for a PCG is as follows: First it is to be determined whether it is possible to give each orientable node in the PCG a well-orientation in the proposed geometric variant. We will call the variant *feasible* if and only if this is so. Next, for each feasible variant a cost is assessed by some measure of displacement from the existing layout; for infeasible ones, the variant is rejected altogether. All that we need indicate then is (1) how to determine whether a proposed geometric variant is feasible, and (2) how to assess the cost of a feasible one. It is in this section that the formalism developed in Section 8.2 begins to serve its purpose.

For PCGs infeasibilities can arise only in relation to the terminal nodes s, t, in the case that one or more of them is an O-node. On the other hand, any O-node v occurring interior to a chain can always be given a well-orientation whatever the proposed axial direction $\mathbb{A}$ of the PCG. This is because as an internal chain node v has, by definition, degree 2 in the skeleton graph, and hence exactly one neighbour connected to each of its two outer ports (since each of these always has at least one neighbour). Considering for example the case in which $\mathbb{A} = \mathbb{S}$, among the two neighbours of an O-node v one is to be placed above v and the other below; v can always be given $\mathrm{ori}(v) \in \{\mathbb{N}, \mathbb{S}\}$ in order to permit this. The other cases $\mathbb{A} \in \{\mathbb{E}, \mathbb{W}, \mathbb{N}\}$ are similar.

Let us therefore consider the well-orientation of the terminal nodes, in the case that either (or both) of them is orientable. Let $\mathbb{A}$ be the proposed axial direction for the PCG. For terminal node u we denote by $I(u)$ the set of *interior port numbers* for node u, by which we mean the set of i such that at least one chain in the PCG attaches to node u at port $p_u^{(i)}$. The orientations of node u that permit it to be well-oriented with respect to the chains of the PCG can then be determined by considering the set $I(u)$, and there are three possible cases:

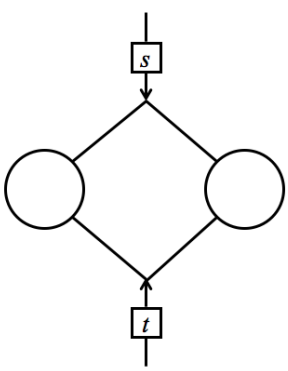
- Positive Axial Case: $I(u) = \{1\}$. We need $\mathrm{ori}(u) = \mathbb{A}$ if $u = s$, and $\mathrm{ori}(u) = \mathrm{cw}_4{}^2\,\mathbb{A}$ if $u = t$. See Figure 8.17a.

- Negative Axial Case: $I(u) = \{-1\}$. We need $\mathrm{ori}(u) = \mathrm{cw}_4{}^2\,\mathbb{A}$ if $u = s$, and $\mathrm{ori}(u) = \mathbb{A}$ if $u = t$. See Figure 8.17b.

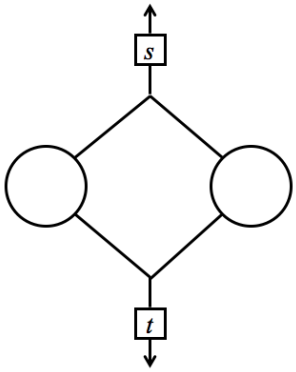
- Transverse Case: $0 \in I(u)$ or $|I(u)| > 1$. In this case we need $\mathrm{ori}(u) \in \{\mathrm{cw}_4\,\mathbb{A}, \mathrm{cw}_4{}^3\,\mathbb{A}\}$. See Figure 8.17c.

Whereas the two axial cases are decisive in themselves, the transverse case leaves a decision to be made, as to which of two possible orientations the terminal node u is to be given. If precisely one terminal is in the transverse case (while the other is either in an axial case or else is not orientable at all), then either of the two transverse orientations may be chosen, and the proposed axial direction $\mathbb{A}$ for the PCG is feasible. See Figure 8.18.

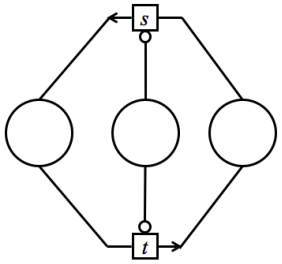
This leaves only one case to be considered, in which both terminals are O-nodes, and both are in the transverse case. We refer to this as a *double-transverse PCG*. (See for example Figure 8.17c, as well as the pseudo-PCG at the bottom of Figure 8.12.) If either



(a) If all chains attach to the source terminal at port $p_s^{(1)}$ then well-orientation requires $\mathrm{ori}(s) = \mathbb{S}$. If all chains attach to the target terminal at port $p_t^{(1)}$ then well-orientation requires $\mathrm{ori}(t) = \mathbb{N}$.

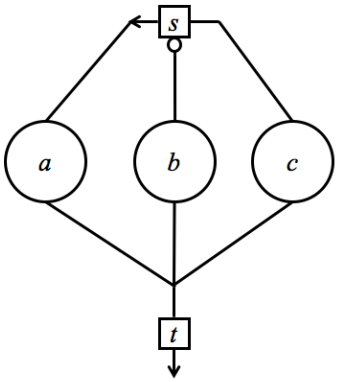


(b) If all chains attach to the source terminal at port $p_s^{(-1)}$ then well-orientation requires $\mathrm{ori}(s) = \mathbb{N}$. If all chains attach to the target terminal at port $p_t^{(-1)}$ then well-orientation requires $\mathrm{ori}(t) = \mathbb{S}$.



(c) If chains attach to two or more of the ports at the source terminal, or if any chain attaches to the source terminal at port $p_s^{(0)}$, then well-orientation requires that $\mathrm{ori}(s) \in \{\mathbb{W}, \mathbb{E}\}$. The same holds for the target terminal.

Figure 8.17: Different ways of orienting the terminal nodes are depicted for a PCG with axial direction $\mathbb{S}$. In this and subsequent figures, an arrow on a spike leaving an orientable node u indicates that this is port $p_u^{(1)}$.



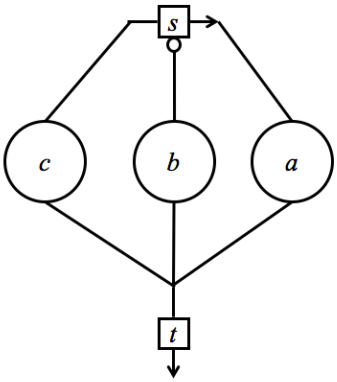
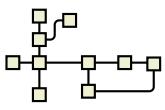


Figure 8.18: When exactly one terminal node is in the transverse case, then it may take either of the two available orientations. The chains must simply be ordered accordingly, in order to avoid crossings.



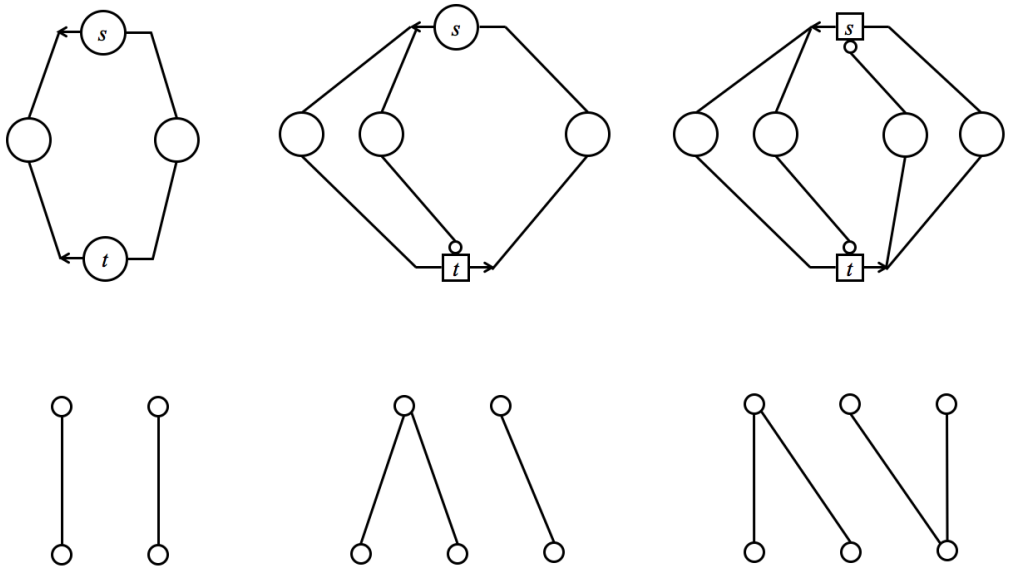


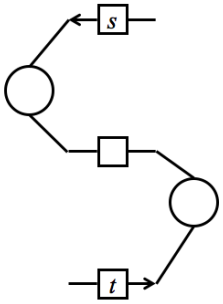
Figure 8.19: Representing each terminal port by a vertex and each chain by an edge, a double-transverse PCG (top row) is modelled by a bipartite graph with two or three vertices in each part (bottom row).

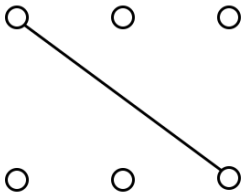
of the terminals u is a logic node then it has only two ports, $p_u^{(1)}$ and $p_u^{(-1)}$, whereas if it is a process node then it has in addition a third port $p_u^{(0)}$. Each chain of the PCG runs from some port of terminal s to some port of terminal t. Therefore, modelling each port by a vertex and each chain by an edge, the topology of a double-transverse PCG is modelled by a bipartite graph each side of which contains either two or three vertices. See Figure 8.19.

Like all GSAs, the PCG arrangement is meant to add clarity to the layout, and help demonstrate the structure of the network. In the double-transverse case we are forced to confront the possibility, to which all GSAs are susceptible, that in some cases the arrangement may become so muddled as to defeat its purpose, and in such cases the arrangement ought to be simply abandoned. In light of this we expand our definition of infeasibility, and call infeasible any double-transverse PCG in which either (a) a chain is forced to cross from one side of the PCG to the opposite side, as in Figure 8.20a, or else (b) two chains are forced to cross one another, as in Figure 8.20b.

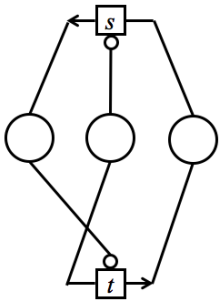
Since each terminal node has two possible orientations in the transverse case, there are a total of four combinations for a double-transverse PCG. As regards well-orientation of the terminals with respect to other nodes outside the PCG, all four combinations must be considered; however, as regards internal feasibility there are only two essentially distinct cases: that in which the two terminals are given the same orientation, and that in which their orientations are opposite.

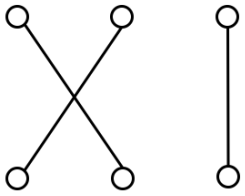
For a given choice of terminal orientations, the two new feasibility conditions (a) and (b) given above are easily checked. We illustrate how this can be done for the case in which both terminals are process nodes and therefore have three ports each; the other cases are similar. Once the terminal orientations are fixed each chain may be said to connect at the *left*, *centre*, or *right* port at the source terminal, and likewise at the *left*, *centre*, or *right* port at the target terminal. Let the former be indicated by ℓ, c, r, and the latter by ℓ', c', r', respectively. Each chain is then of one of nine possible types, being represented by an ordered pair in $\{\ell, c, r\} \times \{\ell', c', r'\}$. Feasibility condition (a) says that types (ℓ, r') and (r, ℓ') are prohibited. In other words, if any chain is of either of these types then the given choice of orientations is infeasible. As for feasibility condition (b), it is satisfiable (by some ordering of the chains) unless either of two pairs of chain types coexist; namely, if there is any chain of type (c, ℓ') then there may not be any chain of type (ℓ, c'), and vice



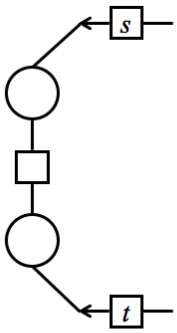


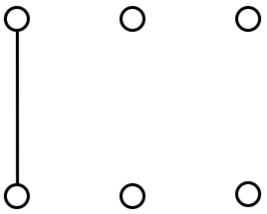
(a) Cases in which the extreme nodes in the corresponding bipartite graph are connected are considered infeasible, since such cases are deemed not to capture the intended metaphor of the PCG.



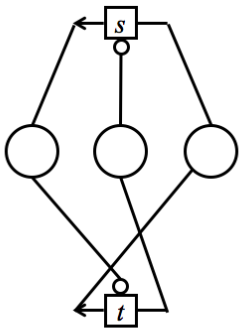


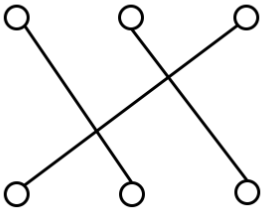
(b) Cases in which the corresponding bipartite graph is nonplanar are considered infeasible, since the muddiness introduced by the crossings are deemed to cancel the benefit of creating a PCG arrangement.





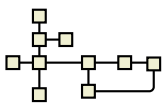
(c) In some cases the infeasibility can be removed by reorienting one of the terminals (compare Figure 8.20a).

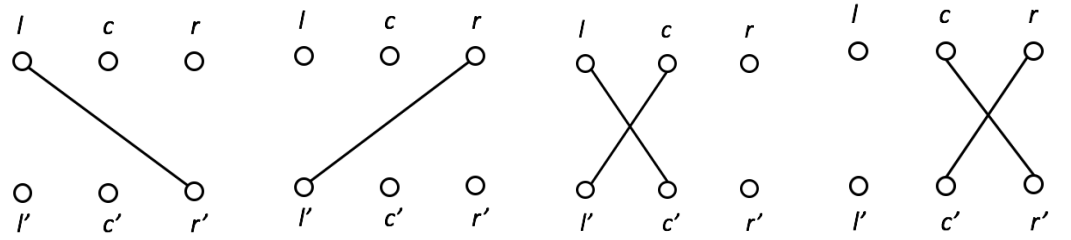




(d) In other cases the infeasibility cannot be removed even if one of the terminals is reoriented (compare Figure 8.20b).

Figure 8.20: For double-transverse PCGs we expand the definition of infeasibility to include cases where chains cross from one side to the opposite side, or cross one another.





(a) Edge (ℓ, r') is prohibited.

(b) Edge (r, ℓ') is prohibited.

(c) Edges (ℓ, c') and (c, ℓ') may not occur together.

(d) Edges (r, c') and (c, r') may not occur together.

Figure 8.21: Prohibited edges and edge pairs in the bipartite graphs corresponding to double-transverse PCGs

versa; and likewise if there is any chain of type (c, r') then there may not be any chain of type (r, c'), and vice versa. See Figure 8.21.

This completes the determination of whether a given geometric variant (as determined by its axial direction $\mathbb{A}$) is feasible for a PCG, and we have only to say how a cost is assigned to a feasible variant.

According to **Faithfulness** the cost is designed to approximate the departure from stress-minimal positions necessitated by the arrangement. Without knowing in advance the exact positions to which the nodes of the PCG will be moved after the constraints are applied, we can approximate the disturbance by considering instead the anticipated angular displacements of nodes relative to one another.

Let the chains of the PCG be $C_1, C_2, \ldots, C_n$, again in the order in which they are encountered in clockwise rotation around the source node s, as when we defined the constraints in the last section. And for each chain C_i, let $v_i^{(0)}, v_i^{(1)}, \ldots, v_i^{(m_i)}$ be all its nodes, as before.

For any two vectors p, q, let $\alpha(p, q) \in [0, \pi]$ be the smaller non-negative angle between them,

$$\alpha(p, q) = \arccos \frac{p \cdot q}{|p|\,|q|}.$$

For any two nodes u, v we will use the same symbols to refer to their centre points; thus $v - u$ is the vector pointing from the centre of node u to the centre of node v. Let G be the intended grid size for the layout. We will use the cardinal and lateral direction letters $\mathbb{E}$, $\mathbb{R}$, etc. (Section 7.4) to represent vectors pointing in the corresponding direction and having length G. Thus for example $\mathbb{E} = (G, 0) = \mathbb{R}$. Recall that $\mathbb{A}$ is the axial direction of the PCG. Let $\mathbb{C} = \mathrm{cw}_4\, \mathbb{A}$.

For each pair of adjacent nodes u, v in the PCG, there is an *idealised vector* $\eta(u, v)$ for the arrangement. See Figure 8.22. For example for a pair of consecutive nodes $v_i^{(j)}$, $v_i^{(j+1)}$ internal to a chain the idealised vector is $\mathbb{A}$. If chain C_{i_0} is to be a centre chain, then we also have

$$\eta(s, v_{i_0}^{(1)}) = \mathbb{A}.$$

If chain C_{i_1} is to be placed beside chain C_{i_0} in the clockwise direction around s then

$$\eta(s, v_{i_1}^{(1)}) = \mathbb{A} + \mathbb{C}.$$

Similarly if chain C_{i_k} is to be placed k chains clockwise (relative to s) from the centre chain then

$$\eta(s, v_{i_k}^{(1)}) = \mathbb{A} + k\mathbb{C}$$

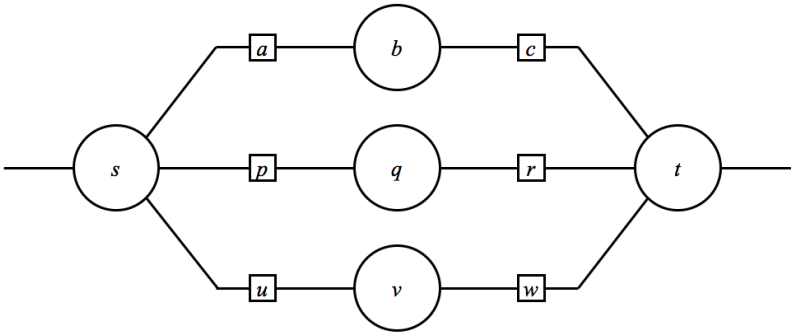


Figure 8.22: The idealised vectors $\eta(u, v)$ for any GSA reflect how we expect nodes u and v to lie relative to one another in the intended arrangement. For computing costs we only need these for nodes connected by an edge. In this example the idealised vectors would be $\eta(s, a) = \mathbb{E} - \mathbb{S}$, $\eta(s, p) = \mathbb{E}$, $\eta(s, u) = \mathbb{E} + \mathbb{S}$, $\eta(a, b) = \mathbb{E}$, and so forth.

and this holds for anti-clockwise chains as well if k takes negative values for these. Finally, if I is the set of edges internal to the PCG then the cost of the arrangement is

$$cost = \sum_{(u,v)\in I} \alpha\left(v - u, \eta(u, v)\right).$$

That is, it sums the angular displacement for each pair of nodes, from the existing position, to the idealised arranged position.

This completes our description of how to choose a geometric variant for a PCG during Phase-A.

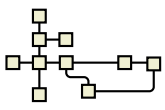
8.4.2 Regular Polygonal Faces

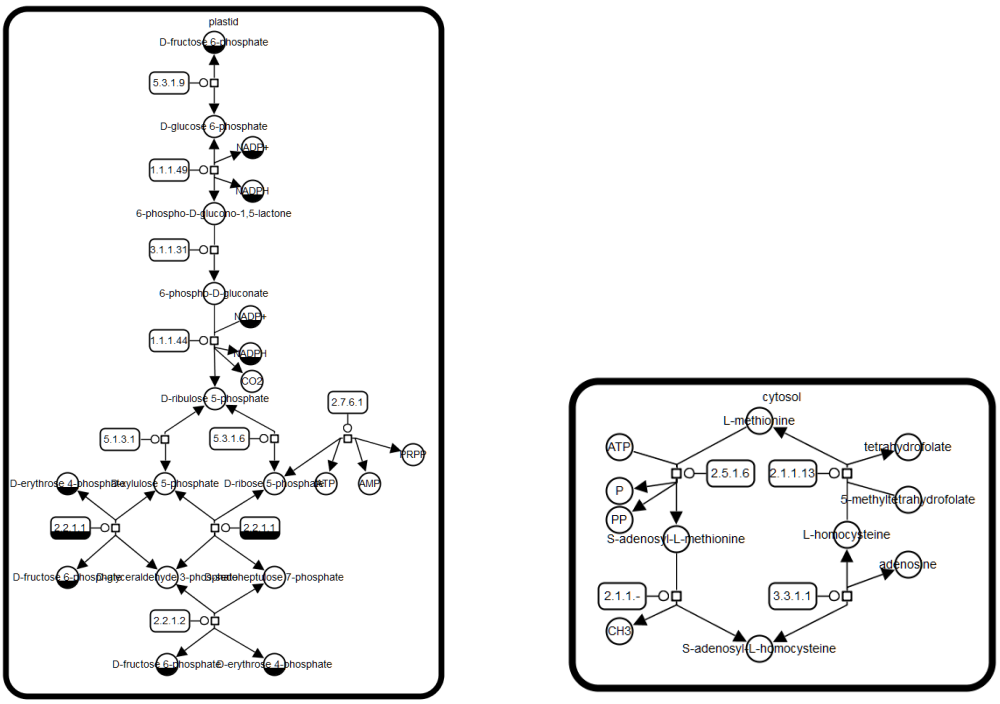
In diagrams in the MetaCrop database it is common to find faces of the skeleton subgraph configured in the shape of regular polygons, as illustrated in Figure 8.23. As detailed in the figure, the number of sides of the polygon need not equal the number of nodes in the circuit (though it cannot exceed it). For example a circuit of eight nodes may be arranged as a hexagon rather than an octagon. This is an excellent example of different possible *structural variants* for a GSA as defined in Section 7.2.1, and the need to pick a standard plan for Phase-A. According to **Standardisation** it suffices to have the automatic layout always choose configuration as a regular n-gon for circuits of n nodes, and leave it to Phase-T to provide handy ways for the user to change this if desired.

Of course, before we can properly speak of the faces of a given embedding it must be planar. Therefore in order to identify faces we temporarily planarise the current embedding of the graph by adding dummy nodes at edge crossings. As in Section 5.3.3, the Bentley-Ottmann algorithm achieves this in $\mathcal{O}((m + k) \log m)$ time, where there are m edges and k crossings. We then identify all faces in the resulting planar graph, and throw away any that involve dummy nodes. We also discard faces that span multiple compartments, for the same reason we discard compartment-spanning PCGs, as explained in Section 8.4.1.

The remaining faces are to be configured as regular polygons. What this means is that for each face F consisting of n nodes we must choose a particular n-gon G in the plane, and then apply constraints that will tend to place the nodes of F upon the vertices of G, with more or less rigidity according to **Minimality**.

Choosing an n-gon means choosing a projective linear transformation of the *basic n-gon*, which is equilateral, is centred at the origin, and has one vertex at $(1, 0)$. We define the *radius* of an n-gon to be the distance from its centre to any of its vertices, so the basic n-gon has radius 1. See Figure 8.24.





(a) Pentose phosphate pathway (MetaCrop) (b) Methionine recycling pathway (MetaCrop)

Figure 8.23: At the centre of Figure 8.23a one face of the skeleton subgraph has been given the shape of a (roughly) regular hexagon. Two others are (roughly) regular quadrilaterals. In Figure 8.23b a face of the skeleton subgraph containing eight nodes is configured as a hexagon (not an octagon) and, although it is not a regular hexagon, it has two axes of reflection symmetry.

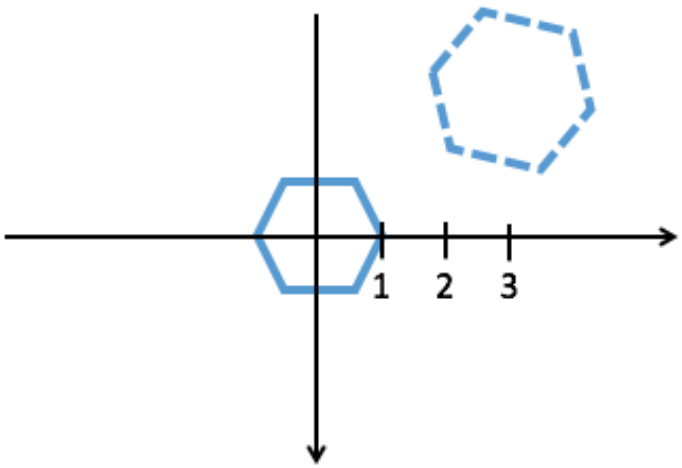


Figure 8.24: The basic n-gon (hexagon illustrated here, in solid lines) is equilateral, is centred at the origin, and has a vertex at $(1, 0)$. Any other n-gon (for the same n, illustrated in dashed lines in this figure) is obtained via a projective linear transformation of this one; that is, via some composition of rotation, translation, and dilation.

Applying a projective linear transformation means first mapping points (x, y) in the plane $\mathbb{R}^2$ to the corresponding points $(x, y, 1)$ in three-space $\mathbb{R}^3$ and then applying a linear transformation T whose matrix representation can be factored as a product of three 3×3 matrices,

$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which apply a rotation of θ radians, a dilation by factor d, and a translation by (a, b). The transformed point $T[x, y, 1] = [u, v, 1]$ can then be mapped back into the plane as (u, v). (Working in three-space is necessary to allow translation by linear transformation.) The decomposition into rotation, dilation, and translation provides an easy way to think about choosing a particular n-gon. We must choose its centre and radius (i.e. translation and dilation) and decide how its vertices are rotated about its centre.

Let us think about rotation first. While there may be no objectively right answer, we can make a principled set of rules describing which rotations to allow by comparing to the case of a single pair of nodes connected by an edge. Returning to the grid-like aesthetics examined in Chapter 3, we feel that such a pair of nodes is most neatly arranged when it is aligned vertically or horizontally, while placing it at a 45-degree angle is either just as good as axis-alignment (as in the popular "octilinear" layout style [NW11b]), or is perhaps a second-best alternative. Meanwhile there are infinitely many other angles at which the edge could possibly lie, but none of the others seems to deserve to be called "neatly configured in a principled way".

What this means is that we think that a single edge is neatly arranged when it is given one of four possible rotations: vertical, horizontal, or one of the two 45-degree diagonals. We might call these the four *canonical rotations* for an edge.

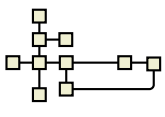
It seems reasonable to think in a similar way about the possible rotations of an n-gon. Which rotations seem somehow neat or principled? Generalising the case of a single edge, we could (A) look for rotations in which one or more of the sides of the n-gon were axis-aligned. Alternatively, it could be that (B) an n-gon appears neatly rotated when one or more of its "radii" are axis-aligned, those being the (imaginary) lines from its centre to its vertices. To put this second condition differently, one or more vertices should point in a cardinal compass direction.

Based on these principles, the number of canonical rotations for an n-gon depends on the residue class of n mod 4. See Figure 8.25. If n is 1 or 3 mod 4 (i.e. odd) then both principles (A) and (B) lead to the same result: there are four canonical rotations. If n is 2 mod 4 then again principles (A) and (B) agree; this time there are two rotations. Finally, if n is 0 mod 4 then principles (A) and (B) give different answers: they each say that there is precisely one canonical rotation, but they disagree on what it is. See Figure 8.25a. Since both principles seem reasonable, we accept both answers.

Structural and Geometric Variants

Let $v_0, v_1, \ldots, v_{n-1}$ be n nodes forming a face cycle, listed in clockwise order. Let $p_0, p_1, \ldots, p_{n-1}$ be the vertices of the basic n-gon (see Figure 8.24), also in clockwise order, and with p_0 being the vertex that lies on the positive x-axis. Note that our choice to use clockwise ordering is consistent with our choice to follow the graphics convention in which the positive y-axis points down, not up, as explained in Section 8.2.2.

We must choose a transformation T of the n-gon and we must choose a mapping M of the nodes v_i onto the transformed vertices $T(p_j)$. According to **Faithfulness**, we will consider only those mappings M that preserve clockwise order, meaning that M is



(a) When n is 0 mod 4 there are two canonical rotations: one in which there is a vertex pointing in each of the cardinal directions, and one differing from that by π/n radians, in which there are two pairs of opposite, axis-aligned sides.

(b) When n is 2 mod 4 there are two canonical rotations: one in which a pair of vertices point north and south, and one in which a pair of vertices point east and west. Each rotation also has one pair of opposite, axis-aligned sides.

(c) When n is 1 or 3 mod 4 (i.e. odd) there are four canonical rotations, in each of which a single vertex points in one of the four cardinal directions, and a single side is axis-aligned.

Figure 8.25: The number of canonical rotations of an n-gon depends on the residue class of n mod 4.

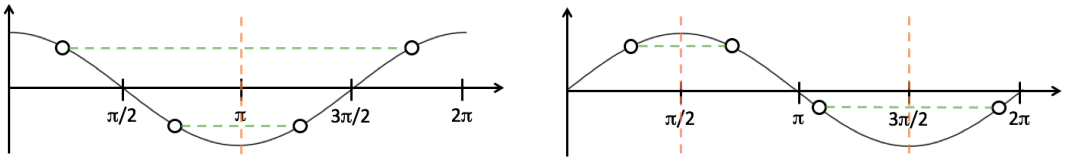
completely determined as soon as we choose the value of $M(v_0)$. Each geometric variant for the regular polygonal face (RPF) is therefore determined by four parameters:

- the centre point c of the transformed n-gon;
- the radius r of the transformed n-gon;
- the canonical rotation ρ of the transformed n-gon (see Figure 8.25); and
- an integer $b \in \{0, 1, \ldots, n-1\}$ called the "mapping base", which defines the mapping M_b from nodes to transformed vertices by $v_i \mapsto T(p_{b+i})$ (where indices are understood mod n).

Note that, since we enforce arrangements using only relative constraints between nodes, the centre point c does not matter when computing constraints; on the other hand it does matter when computing costs in order to choose a geometric variant.

Let $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$ be the existing coordinates of the n nodes, i.e. the coordinates as of the current iteration of the constraint generation loop. Then for the centre point c we take the coordinate-wise mean of these n points, and for the radius r we take the mean distance from each of these points to c.

For each of the canonical rotations ρ we then get two potential geometric variants. To define these we find among the transformed points $T(p_j)$ the two $T(p_b)$, $T(p_{b'})$ that are closest to the point (x_0, y_0). In most cases we expect one of these points to lie in the clockwise direction from (x_0, y_0) (relative to c), and the other in the counterclockwise direction, and it is reasonable to try rotating the nodes $v_0, v_1, \ldots, v_{n-1}$ in either direction in mapping them onto the vertices of the polygon. Therefore the two mappings M_b, $M_{b'}$ define the two desirable geometries for rotation ρ. Altogether we have either four or eight potential geometric variants, depending on whether n is even or odd, respectively (again,



(a) Two unequal angles in the interval $[0, 2\pi)$ have the same cosine if and only if their mean is π.

(b) Two unequal angles in the interval $[0, 2\pi)$ have the same sine if and only if their mean is $\pi/2$ or $3\pi/2$.

Figure 8.26: Graphs of cosine and sine, showing for what sort of pairs of unequal angles they take on equal values

see Figure 8.25). We will select one of these variants based on both feasibility and cost, as we did with PCGs.

Constraints

We define the alignment and separation constraints that enforce a regular polygonal shape on the nodes making up an RPF, beginning with the alignments. These depend only on n and the chosen canonical rotation, so we may define them for the basic n-gon and indicate how those for the chosen rotation may be obtained from these. Recall that the basic n-gon is centred at the origin, has unit radius, and has a vertex on the positive x-axis. Beginning with that vertex and proceeding clockwise, the vertices have coordinates

$$\left(\cos\left(\frac{2\pi}{n}k\right), \sin\left(\frac{2\pi}{n}k\right)\right)$$

for $k = 0, 1, \ldots, n-1$. We must create a vertical alignment for each pair of vertices with common x-coordinate, and a horizontal alignment for each pair with common y-coordinate. Therefore let $0 \leq k_1 < k_2 < n$ be given, and let us consider the conditions under which the two angles $\alpha_1 = 2\pi k_1/n$ and $\alpha_2 = 2\pi k_2/n$ have equal cosine or sine.

Considering the graph of cosine over the interval $[0, 2\pi)$ (see Figure8.26a) we see that the two angles $\alpha_1 \neq \alpha_2$ have the same cosine if and only if their mean $(\alpha_1 + \alpha_2)/2$ equals π; equivalently, if and only if

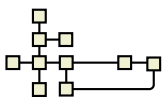
$$k_1 + k_2 = n. \tag{8.3}$$

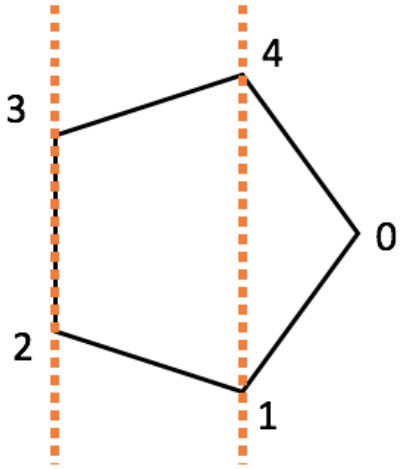
Meanwhile the graph of sine (Figure 8.26b) shows that $\alpha_1 \neq \alpha_2$ have the same sine if and only if their mean is either $\pi/2$ or $3\pi/2$; equivalently, if and only if

$$k_1 + k_2 \in \left\{\frac{n}{2}, \frac{3n}{2}\right\}. \tag{8.4}$$

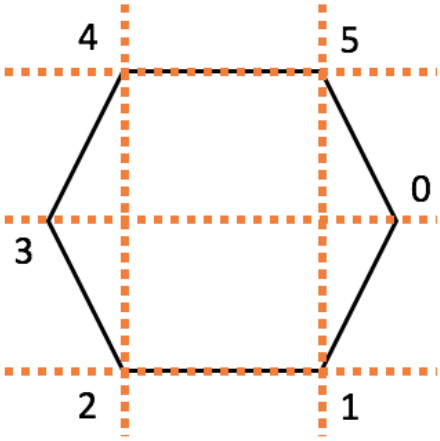
We can now consider the several cases, as illustrated in Figure 8.27. This tells us on which pairs of vertices we must apply alignment constraints.

With the alignments in place we need only add separation constraints, and honouring **Minimality** we enforce minimum, not exact, separations. The values of these separations are easily read off of the coordinates of the points $T(p_0), T(p_1), \ldots, T(p_{n-1})$ of the transformed n-gon. In other words, we enforce as minimum separations the actual separations between the points $T(p_i)$, in both dimensions.

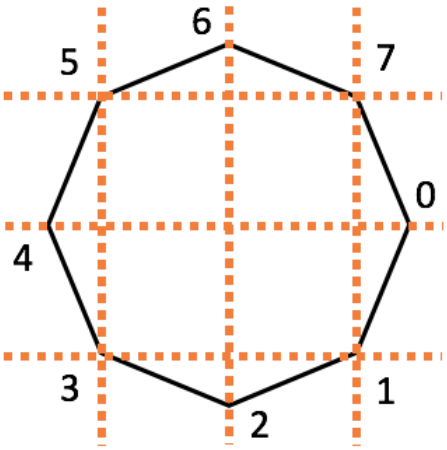




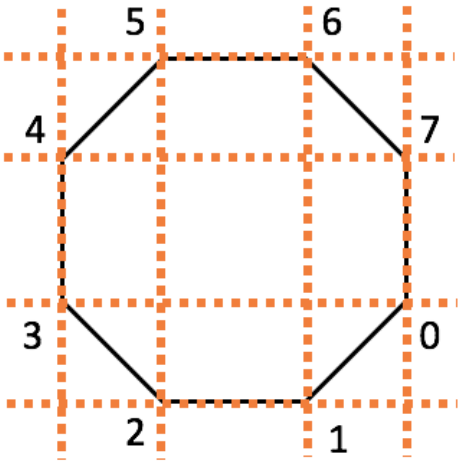
(a) When n is odd the basic n-gon has $(n-1)/2$ vertical alignments, according to Equation (8.3). Meanwhile, as Equation (8.4) predicts, there are no horizontal alignments when n is odd. Each of the three remaining canonical rotations is obtained from the basic one by one or more 90-degree rotations, and the alignments are easily obtained from the basic case.



(b) When n is 2 mod 4 the basic n-gon has $n/2$ horizontal alignments, and $(n-2)/2$ vertical alignments, as predicted by Equations (8.3) and (8.4). Again, the other canonical rotation differs by 90 degrees, so its alignments are easily obtained from the basic case.



(c) When n is 0 mod 4 the basic n-gon has $(n-2)/2$ alignments in each dimension. The second canonical rotation differs not by a multiple of 90 degrees but by $180/n$ degrees, so requires special treatment; see Figure 8.27d.



(d) The second canonical rotation for $n \equiv 0$ mod 4 is the oddball case. Following the same analysis we used with Equations (8.3) and (8.4) but with a phase shift of π/n shows that we get vertical alignments when $k_1 + k_2 = n - 1$ and horizontal when $k_1 + k_2 \in \{(n-2)/2, (3n-2)/2\}$. We get $n/2$ alignments in each dimension.

Figure 8.27: Alignments for regular n-gons. In each figure the vertices are numbered, and these numbers play the role of the k_i in Equations (8.3) and (8.4)

Variant Selection

As with PCGs, we have two tasks: (1) to indicate how a potential geometric variant is to be checked for feasibility, meaning that it permits all O-nodes to be well-oriented, and (2) how to assess a cost for each feasible variant.

For RPFs the cost computation is simple. Since we have the exact points at which the nodes of the RPF are to be positioned, we assess as cost the sum of the distances by which the n nodes $v_0, v_1, \ldots, v_{n-1}$ are displaced from their current positions under the transformation T and mapping M_b that define the geometry, i.e.

$$cost = \sum_{i=0}^{n-1} |v_i - M(v_i)| . \tag{8.5}$$

It remains only to say how feasibility is to be checked. For i from 0 to $n-1$ let $q_i = T(p_{b+i})$, so that $q_0, q_1, \ldots, q_{n-1}$ are the transformed polygon points and $M_b : v_i \mapsto q_i$. Suppose v_i is an orientable node. Its neighbours in the RPF are v_{i-1}, v_{i+1} (throughout the discussion indices are understood mod n), and the question whether v_i can be well-oriented in the given geometric variant depends on the relative positions of the points q_{i-1}, q_i, q_{i+1}. For ease of expression we set $u, v, w = v_{i-1}, v_i, v_{i+1}$ and $p, q, r = q_{i-1}, q_i, q_{i+1}$.

If v has already been given an orientation in the layout process up to this point, then we must simply check whether $\mathrm{wo}(v, u, p)$ and $\mathrm{wo}(v, w, r)$ both hold (see equation (8.2), page 123). If either condition fails then node v cannot be well-oriented with respect to the proposed geometry, so the geometry is assigned infinite cost.

Otherwise v has not yet been oriented and we must check whether any well-orientation is possible. Theoretically this means checking all four possible orientations, but in fact we can check them two at a time, since it turns out that $\mathbb{E}$ orientation is possible if and only if $\mathbb{W}$ is, and likewise $\mathbb{N}$ if and only if $\mathbb{S}$.

Let $H = \{\mathbb{E}, \mathbb{W}\}$ and $V = \{\mathbb{S}, \mathbb{N}\}$. We can write down a predicate $F(A)$ such that $F(H)$ holds if and only if there is a feasible orientation for v in the set H, and similarly $F(V)$ holds if and only if there is a feasible orientation for v in the set V. We define $d_p = \mathrm{catdir}(q, p)$ and $d_r = \mathrm{catdir}(q, r)$, and we use the Ω function defined in Table 8.2, page 123. There are five cases to consider, and an example is illustrated in Figure 8.28.

We first ask how many among v's neighbours u and w are modulators. If both u and w are modulators then

$$F(A) \equiv \{d_p, d_r\} \cap A = \varnothing. \tag{8.6}$$

If u is a modulator and w is not, then

$$F(A) \equiv d_p \notin A \wedge \Omega(d_r) \cap A \neq \varnothing. \tag{8.7}$$

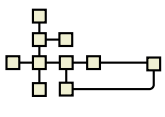
Obviously if the roles are reversed and w is a modulator while u is not, then the predicate F is similar. Finally, consider the case in which neither u nor w is a modulator. Then we must ask whether $p_v(u) = p_v(w)$, i.e. whether u and w connect to v at the same port. If $p_v(u) = p_v(w)$, then

$$F(A) \equiv \Omega(d_p) \cap A \cap \Omega(d_r) \neq \varnothing, \tag{8.8}$$

whereas if $p_v(u) \neq p_v(w)$ then

$$F(A) \equiv A \subseteq \Omega(d_p) \cup \Omega(d_r). \tag{8.9}$$

After formulating the predicate $F(A)$ we check whether $F(H) \vee F(V)$ holds. If not then we assign the geometry infinite cost; otherwise v has a well-orientation for the geometry, so we move on to check the next orientable node v in the RPF. If all orientable nodes in



the RPF can be well-oriented for the geometry in question, then the assigned cost is given by Equation 8.5.

8.4.3 Sector-Partitioned Orbits

Positioning the satellite nodes $s_1, s_2, \ldots, s_n$ of a given orbit $\mathscr{O}$ relative to their common parent node p in the skeleton graph is much easier than handling the GSAs we have looked at in the previous sections. The satellites are leaf nodes and can straightforwardly be placed where desired.

Our goal is to achieve a roughly circular distribution of the satellites around the parent node p (recall Figure 8.9, page 125), subject to the conditions that:

1. p is well-oriented,
2. modulators are evenly distributed between alternative ports,
3. on any side of p that gets an odd number of satellites, if there is no aligned skeleton neighbour then one satellite is aligned with p.

Since a circular distribution is naturally achieved by unconstrained stress-minimisation we try to meet the above conditions using a minimum of necessary constraints.

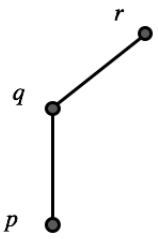
This gives rise to the idea of the *sector-partitioned orbit*, in which lateral (not cardinal) constraints serve to partition the satellites s_i into four sectors of angular space around the parent node p. Consider for example a process node p with $\mathrm{ori}(p) = \mathbb{S}$, and having among its satellites s_i exactly two consumption, two modulation, and two production neighbours, as illustrated in Figure 8.29a. If these are $s_0, s_1, s_2, s_3, s_4, s_5$ respectively, then in order to partition the satellites into sectors around p the lateral constraints expressed as entries in the relative constraint matrix in Table 8.29b will suffice. The general case follows in an obvious way from this example.

Note also that in this example there are two modulators and we have divided them evenly among the two sides of the process node glyph where they may attach. As a result both the $\mathbb{E}$ and $\mathbb{W}$ sides of the process node have precisely one neighbour each; since this is an odd number, those neighbours are aligned with the process node. Similarly had there been, say, an odd number of consumption neighbours, then one of those would have been aligned $\mathbb{N}$ of the process node.

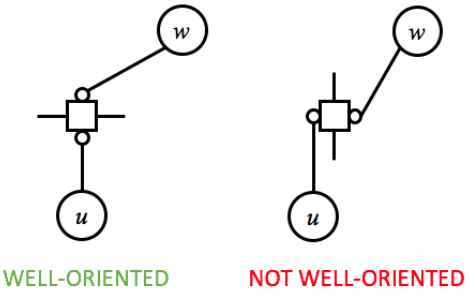
Once the sector-partitioning and alignment constraints have been added, stress minimisation will generally distribute the satellites evenly in the available angular space around their common parent node.

One difficulty, examples of which will be seen in Section 9.3, is that this process sometimes results in edge-node overlaps involving satellites. Considering the freedom with which positions may be assigned to leaf nodes, it seems reasonable to try computing satellite positions by direct geometric computations rather than by minimising stress, and thereby attempting to avoid such overlaps. However, cursory investigations have revealed such direct computations to be difficult, and so it remains for future work to properly attempt such an approach.

There are two issues involved: (1) How to avoid edge-node overlaps, and (2) How to position the satellites around their parent node. In HOLA we could easily deal with Problem (1) since it was orthogonal layout. There, each connector segment is axis aligned, and we can represent it by a long, thin node. Then the existing node-node overlap removal system of ADAPTAGRAMS solves the problem. With the diagonal edges present in SBGN and other non-orthogonal layouts, we need a new solution. We have considered representing diagonal edges by sets of small square nodes and applying the same technique, but this leaves the question of choosing the square sizes and of updating their positions as the proper nodes move.

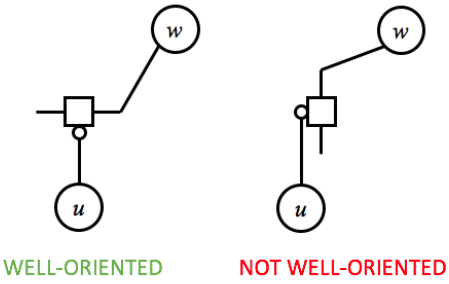


We examine a case in which points p, q, r lie on the vertices of a regular octagon, with $d_p = \text{catdir}(q,p) = \mathbb{S}$ and $d_r = \text{catdir}(q,r) = \mathbb{NE}$. In this case $\Omega(d_p) = \{\mathbb{E},\mathbb{S},\mathbb{W}\}$ and $\Omega(d_r) = \{\mathbb{E},\mathbb{N}\}$. Nodes u and w are to be placed at points p and r respectively, and process node v is to be placed at point q.



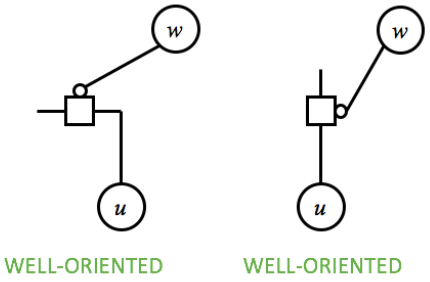
Case: u, w modulators.

$$\begin{aligned} F(H) &\equiv \{\mathbb{S},\mathbb{NE}\} \cap H = \varnothing \\ &\equiv \top \\ F(V) &\equiv \{\mathbb{S},\mathbb{NE}\} \cap V = \varnothing \\ &\equiv \bot \end{aligned}$$



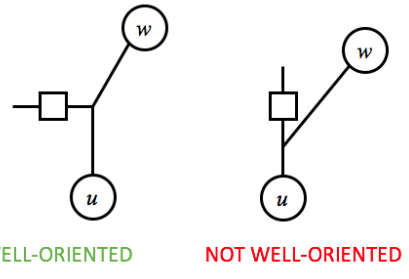
Case: u modulator, w not.

$$\begin{aligned} F(H) &\equiv \mathbb{S} \notin H \wedge \{\mathbb{E},\mathbb{N}\} \cap H \neq \varnothing \\ &\equiv \top \\ F(V) &\equiv \mathbb{S} \notin V \wedge \{\mathbb{E},\mathbb{N}\} \cap V \neq \varnothing \\ &\equiv \bot \end{aligned}$$



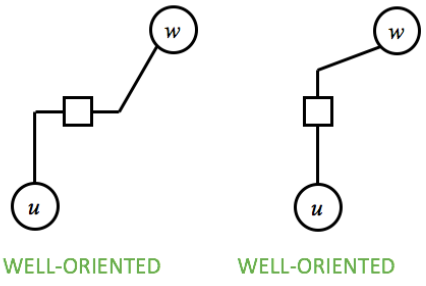
Case: w modulator, u not.

$$\begin{aligned} F(H) &\equiv \mathbb{NE} \notin H \wedge \{\mathbb{E},\mathbb{S},\mathbb{W}\} \cap H \neq \varnothing \\ &\equiv \top \\ F(V) &\equiv \mathbb{NE} \notin V \wedge \{\mathbb{E},\mathbb{S},\mathbb{W}\} \cap V \neq \varnothing \\ &\equiv \top \end{aligned}$$



Case: no modulators, shared port.

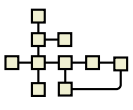
$$\begin{aligned} F(H) &\equiv \{\mathbb{E},\mathbb{S},\mathbb{W}\} \cap H \cap \{\mathbb{E},\mathbb{N}\} \neq \varnothing \\ &\equiv \top \\ F(V) &\equiv \{\mathbb{E},\mathbb{S},\mathbb{W}\} \cap V \cap \{\mathbb{E},\mathbb{N}\} \neq \varnothing \\ &\equiv \bot \end{aligned}$$

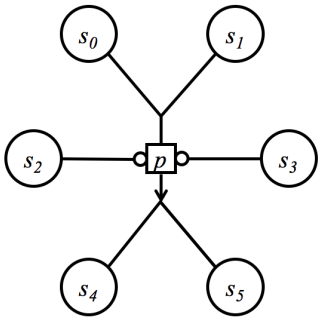


Case: no modulators, opposite ports.

$$\begin{aligned} F(H) &\equiv H \subseteq \{\mathbb{E},\mathbb{S},\mathbb{W}\} \cup \{\mathbb{E},\mathbb{N}\} \\ &\equiv \top \\ F(V) &\equiv V \subseteq \{\mathbb{E},\mathbb{S},\mathbb{W}\} \cup \{\mathbb{E},\mathbb{N}\} \\ &\equiv \top \end{aligned}$$

Figure 8.28: Feasibility tests for an example RPF geometry, using Equations (8.6) through (8.9)





(a) For a process node p oriented $\mathbb{S}$ with two consumption, two modulation, and two production satellites $s_0, s_1, s_2, s_3, s_4, s_5$ as illustrated here, the requisite constraints are given in Table 8.29b.

	p	s_0	s_1	s_2	s_3	s_4	s_5
p		$\mathbb{U}$	$\mathbb{U}$	$\mathbb{W}$	$\mathbb{E}$	$\mathbb{D}$	$\mathbb{D}$
s_0			ε	$\mathbb{DL}$	$\mathbb{DR}$	$\mathbb{D}$	$\mathbb{D}$
s_1				$\mathbb{DL}$	$\mathbb{DR}$	$\mathbb{D}$	$\mathbb{D}$
s_2					$\mathbb{R}$	$\mathbb{DR}$	$\mathbb{DR}$
s_3						$\mathbb{DL}$	$\mathbb{DL}$
s_4							ε

(b) The table gives relative constraint matrix (RCM) entries partitioning satellites into sectors for the case illustrated in Figure 8.29a. The entry in row a, column b is to be understood as the entry $M(a, b)$ in matrix M; in other words, this gives the constrained directions from node a to node b. Since all RCMs are antisymmetric and equal to ε along the main diagonal, only the upper triangular entries are given.

Figure 8.29: Example sector-partitioned orbit arrangement, with relative constraint matrix entries that enforce its sector partitioning and alignments

As for Problem (2), positioning satellites around their parent node, this seems like easy geometry as long as we envision simple cases like four to six satellites, all of the same size, and distributed symmetrically among the parent node's ports. In full generality however, it is a tricky optimisation problem. The problem is: Given a centre point c, to pack n rectangles R_i of various (given) dimensions w_i by h_i as close to c as possible while keeping certain rectangles R_i assigned to certain angular sectors $[a_k, b_k]$, and also keeping the angular spacing between the rectangles as even as possible. Considered in this light, it seems reasonable to abandon attempts at direct geometric computation in favour of a simple optimisation technique like stress minimisation with overlap removal. Due to the imperfections that remain, this is an area for future research.

8.4.4 Constraint Generation Loop

Having described the new GSAs for MetaCrop SBGN diagrams, we can finally indicate the operation of Phase-A by describing the CGL, or constraint generation loop. This includes both initialisation steps that must be carried out before the loop begins, and the loop itself.

Initialisation

- Before the CGL can begin it is necessary to identify the targeted substructures in the network, and this must be done with care. In particular we search for PCGs first. Then after finding all RPFs we discard any of these that happens to be a subgraph of any PCG. It is important that we operate in this order, or else two-chain PCGs would be identified as faces, and would not receive the desired arrangement.

- To manage the order in which we consider the targeted substructures we build an auxiliary graph A with a vertex representing each structure and an edge connecting the vertices for any structures that share nodes or edges in the original network. For example if two PCGs share a terminal node then their vertices are neighbours in A.

Constraint Generation Loop

- We traverse each connected component C of the auxiliary graph A by breadth-first search starting from a central vertex. Centrality of a vertex $v \in C$ is measured by computing the shortest path from v to each other vertex $u \in C$, and taking the reciprocal of the maximum of the lengths of these paths. Thus a central vertex is one whose maximal distance to any other vertex in the component is a short as possible.

  If there are several vertices in C with maximal centrality, the tie is broken first by favouring a vertex representing a largest substructure in the original network, i.e. one containing the greatest number of nodes, and secondarily (if there are several of the largest size) by an arbitrary but not random means, such as choosing one containing a node of minimal ID (where all nodes in the graph have been assigned unique integer IDs by some mechanism).

  It is by this ordering of the graph substructures that we address arrangement principle **Cooperation**. For each connected component C, the central structure with which the process begins stands the greatest chance of achieving a desirable arrangement, since at this stage there are not yet any potentially conflicting constraints in effect[3]. From this starting point the breadth-first search causes each subsequent graph substructure to be handled at a time when at least one of its neighbours already has been arranged (or failed to take on any arrangement due to infeasibility). This means that decisions made earlier in the process have a chance to propagate smoothly and influence each subsequent decision.

  It is expected that good arrangements are most important in the most central substructures since these will appear centrally in the final diagram, and be among the most eye-catching parts of the layout. This is also the reason for breaking ties by favouring larger substructures when selecting the starting point.

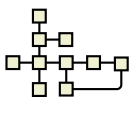
  Moreover as the influence of prior arrangements propagates outward from the first arranged substructure, the centrality of that one tends to minimise the number of steps taken before this propagation reaches each other substructure in the graph. This further tends to minimise the influence of earlier-arranged substructures on later ones, by minimising the number of pre-existing arrangement constraints that may come to bear. All of this goes toward allowing each substructure to have a fair chance to be arranged, in proportion to its centrality in the graph, thus addressing **Cooperation**.

- It was noted at the beginning of this Section 8.4 on Phase-A that because of the importance of well-orientation for SBGN we are willing to apply arrangements that change the cyclic ordering of neighbours relative to a given node, as necessary in pursuit of that goal. However, considering **Balance** and referring back to the demonstration given in Figure 5.2 (page 63), we do apply DESCEND after any PROJECT that alters the rotation system.

- After arranging all PCGs and RPFs in the skeleton we must conduct a pass in which we choose an orientation for any orientable nodes that did not already receive one. In terms of graph substructures, this may for uniformity's sake be viewed as a consideration of *hubs* (Table 7.2), but only those whose centre nodes are orientable.

  The manner of arrangement of these "orientable hubs" is simple. We consider each of the four possible orientations and choose one of least cost, where cost measures

[3] Alternatively, the user may be allowed to pre-define constraints before the layout process begins, in which case those will be the only ones in effect at this stage



angular displacement of neighbours relative to the orientable node in question, just as was used in the cost function for PCGs (see Section 8.4.1).

This completes the enumeration of special considerations for the constraint generation loop of our DiAlEcT layout for SBGN diagrams in the MetaCrop dialect. Apart from these considerations the loop is conducted in the routine manner prescribed in Section 7.2.2.

8.5 Phase-E

In MCGL's Phase-E we begin by expanding to make room for the orbits. This is much easier than the expansion for trees in HOLA since an orbit is centred on its parent node, rather than set off to one side of it in one of the faces to which it belongs. Therefore expansion for an orbit $\mathscr{O}$ with parent node p is easily achieved by the following steps:

1. Increase the size of p to equal the size of the bounding box of the laid out orbit $\mathscr{O}$.
2. Apply OVERLAP-REMOVAL.
3. Return p to its original size and re-attach $\mathscr{O}$ to p.
4. DESCEND with overlap-prevention.

We finish up Phase-E with emending operations, and this begins with the same operations used in HOLA. Nearly missed alignments are created, and DESCEND with Neighbour Stress is applied for better node distribution.

In addition we introduce a new technique tailored to MetaCrop diagrams which we call *Alignment Continuation*. Here we search for any orientable node u that is aligned with a neighbouring node v attached to one of its two outer ports $p_u^{(1)}$ or $p_u^{(-1)}$, but *not* aligned with any node on its opposite port. For example, consider the process node bottom-left in Figure 8.30c, which is vertically aligned with node 12 to its north side but not vertically aligned with any node on its south side. We found that in MetaCrop diagrams such node alignments were "continued" when a leaf node was available on the opposite side of the orientable node, like node 11 in this example. Our Alignment Continuation step creates such alignments as in Figure 8.30b, matching the MetaCrop layout in Figure 8.30a.

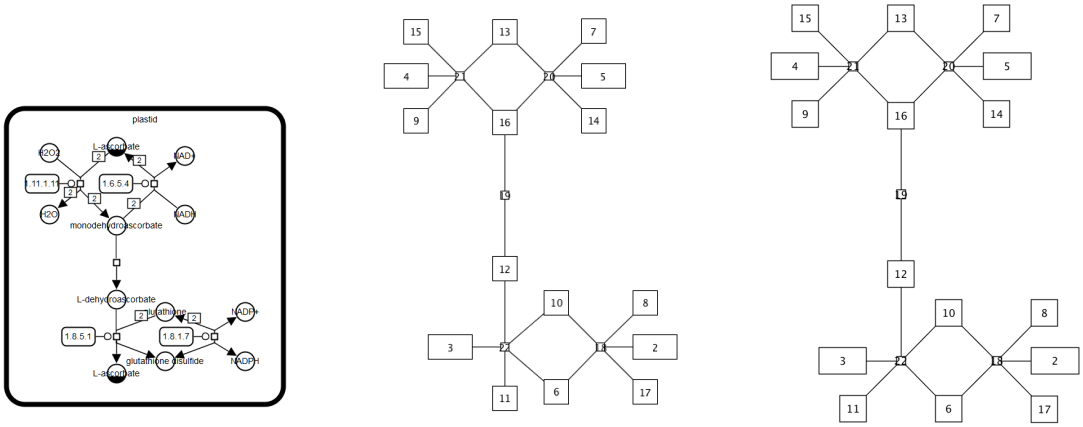
8.6 Results

As noted in this chapter's introduction, MCGL has been designed to work on Type I SBGN diagrams in the MetaCrop style. We begin by observing MCGL's near-perfect mimicry on three of the more interesting diagrams from the example corpus, as shown in Figures 8.30, 8.31, and 8.33.

We take these excellent results as strong indication that, were a normative user study (Step 3 of the human-centred methodology) to be carried out, MCGL would perform comparably to human layout for Type I diagrams. Note however that such a study is not meant to be a part of this thesis, the purpose of the present chapter being only to demonstrate the systematic applicability of the DiAlEcT framework, as a way of carrying out Step 2 of the methodology (algorithm design).

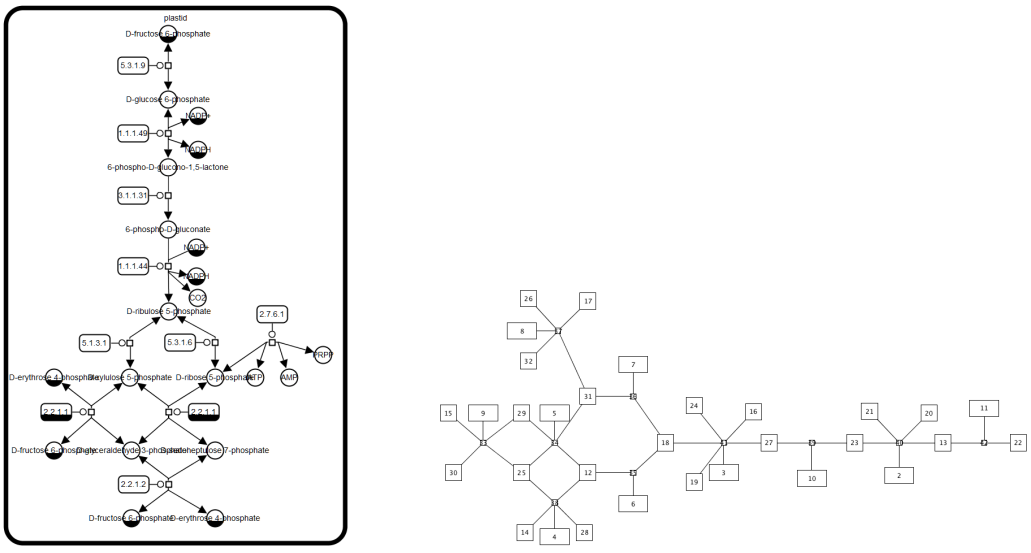
As for Type II SBGN diagrams, we can briefly begin to understand the challenges by examining the performance of MCGL on a simple example. For this we select the "MAPK Cascade", a popular example which was featured in the first edition of the SBGN user manual.

As Figure 8.34 shows, we have already run into a basic syntactic issue. At the top of the diagram the constraints selected by MCGL have enforced an edge-node overlap, where



(a) MetaCrop Layout (b) MCGL Layout (c) MCGL Layout before Alignment Continuation (Section 8.5)

Figure 8.30: Ascorbate-Glutathione Cycle pathway



(a) MetaCrop Layout (b) MCGL Layout

Figure 8.31: Pentose Phosphate pathway

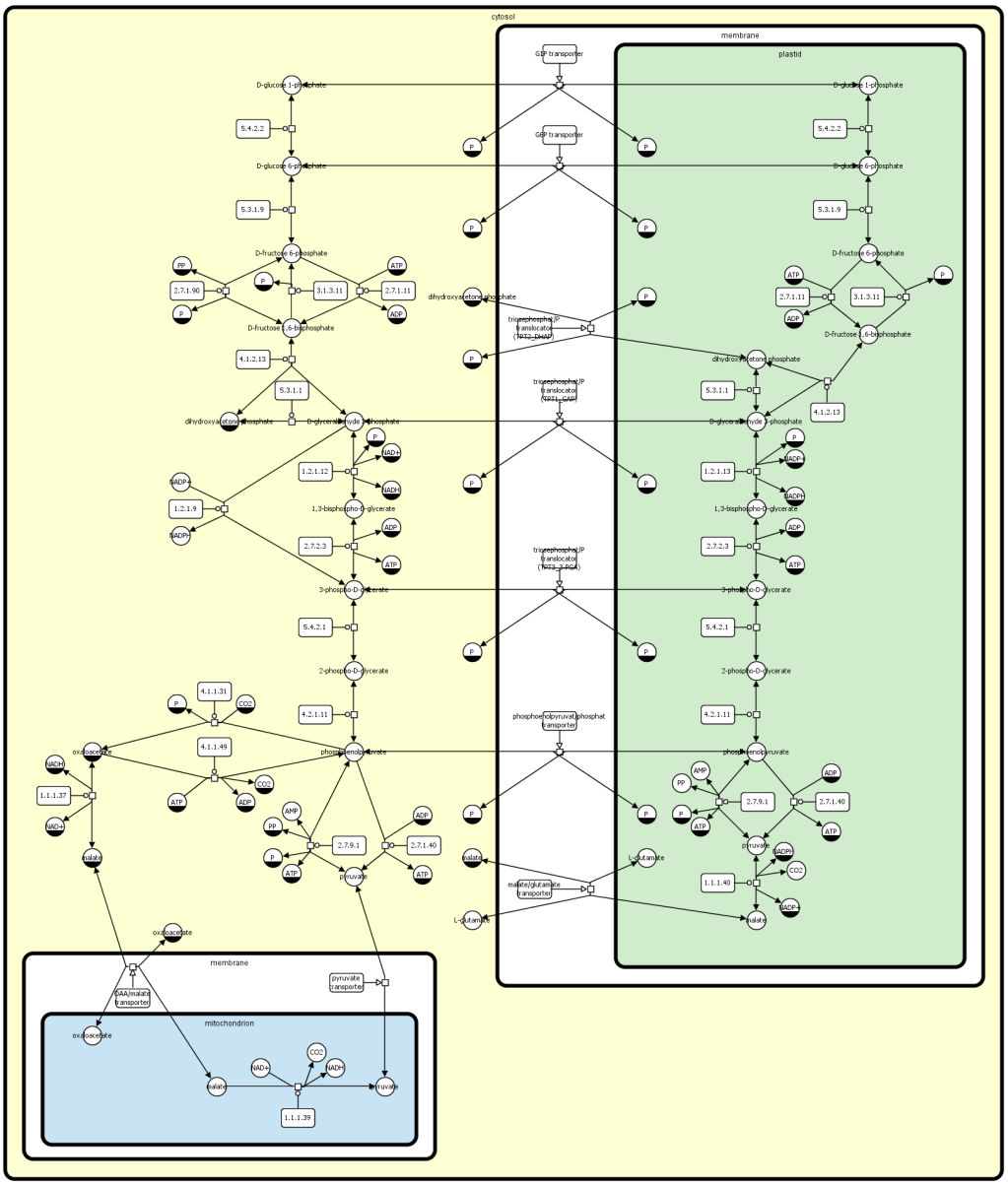


Figure 8.32: MetaCrop layout of Glycolysis-Gluconeogenesis pathway. Compare Figure 8.33.

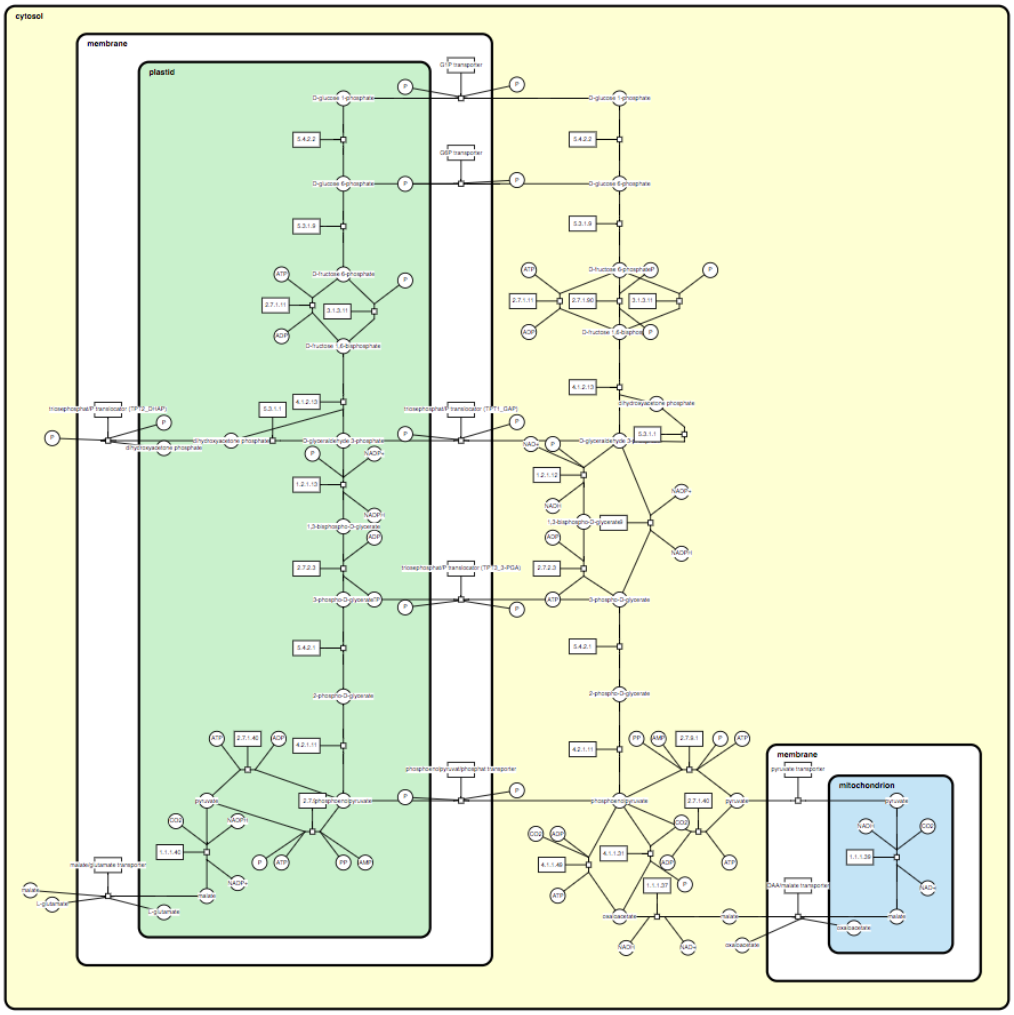


Figure 8.33: MCGL layout of Glycolysis-Gluconeogenesis pathway. Compare Figure 8.32.

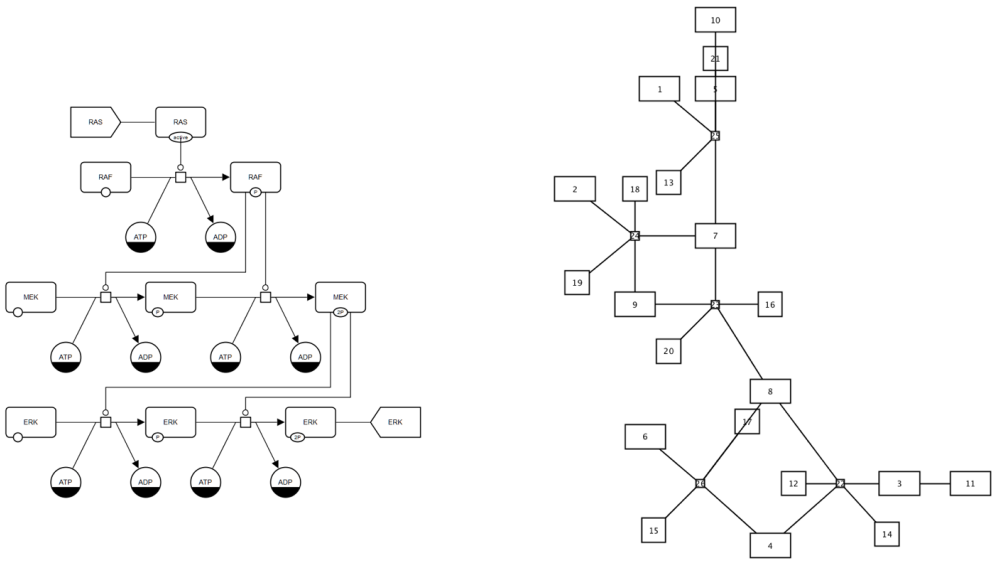


Figure 8.34: Hand-made layout (left) of the MAPK pathway shows a *cascade* pattern: the output of the first process (say process A) serves to catalyse the next two processes (say B_1 and B_2), whose output in turn catalyses the final two processes (say C_1 and C_2). The layout shows three *rows*, with process A in the top row, processes B_1, B_2 in the next, and processes C_1, C_2 in the last. This kind of system, in which the output of one sequence stimulates the next sequence is always represented in SBGN by Type II diagrams. MCGL layout (right) was designed only for Type I diagrams and predictably misses the cascade metaphor. This represents a goal for future work. As discussed in Section 8.6 MCGL also makes a syntactic error caused by the presence of non-leaf modulators in this Type II diagram.

both nodes 5 and 21 have been aligned north of the top-most process node. The basic issue is that in the Type I diagrams for which MCGL was designed, modulators are never present in the skeleton subgraph; they are always pruned in Phase-D. Here however, node 5 is a modulator that remains present in the skeleton of this Type II network. Therefore when the chain at the top of the diagram is aligned vertically a syntactic error creeps in as this modulator is aligned on one of the "spike port" sides of a process node. Since modulators were not expected to be present, the code was never designed to catch such an error. Finally, when MCGL carries out the Alignment Continuation step of Phase-E it aligns node 21 north of the process node at the top of the diagram, failing to even see that node 5 is already aligned there since, again, such a node is not supposed to be present in the skeleton at all.

While this syntactic error is enlightening in its demonstration of how sensitive Phase-A can be to the expected structure of the network, the problem would be relatively easy to solve. The current implementation would simply have to be made to operate with an awareness of the possible presence of modulators in the skeleton subgraph.

A more serious difficulty arises on the semantic side, since on this Type II diagram MCGL also fails to capture the desired perceptual organisation. MCGL identifies the two quadrilateral faces in the skeleton graph and creates square RPF arrangements for them in Phase-A, but this fails to realise the *cascade* metaphor displayed in the hand-made layout, as explained in Figure 8.34.

One possible approach to handling Type II diagrams might be to do more decomposition in Phase-D. After removing the satellite nodes, each remaining modulation arc in the skeleton could be temporarily severed. Phase-A could then be applied to each of the resulting components separately, while Phase-E would be responsible for reconnecting the parts in proper cascade patterns, or other desirable patterns, as appropriate. Exploring such an approach must however await future research.

8.7 Conclusions

This first deliberate application of the DiAlEcT framework has demonstrated much that may prove typical of future applications.

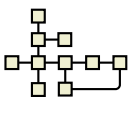
The SBGN language presented new challenges in the form of compartments, ports, and orientability. While the methods we developed to handle these features will likely be useful in handling future languages that possess similar features, a lesson learned is that within the guidelines of DiAlEcT there can be significant new challenges in each application.

Our analysis of GSAs was long. For example, just analysing the geometric and structural variants of the parallel chain group structure, and deciding how to choose a variant and how to enforce it via constraints, took about nine pages and required the introduction of a great deal of special terminology (e.g. "double-transverse pseudo PCG") and formalism. This too may prove typical in future applications.

Many choices had to be made about GSAs. It might be good to make such choices into configurable options, so that "speakers" of the layout dialect may fine-tune the algorithm to their desired layout conventions. For example we made some choices about infeasibility conditions for potential geometric variants of both the parallel chain group and regular polygonal face GSAs. We rejected PCG geometries that made chains cross; certain users might be willing to accept such crossings. We rejected RPF geometries if any O-nodes could not be well-oriented; some users might not care.

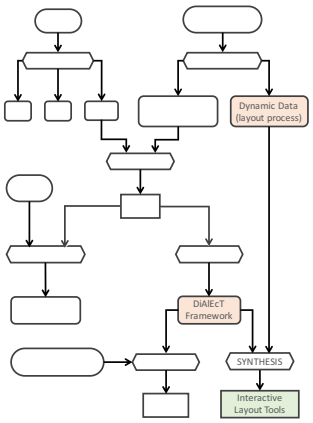
In terms of our human-centred methodology such examples teach us something about how to better conduct our formative user studies. Given the opportunity to quiz human graphic designers about the house diagramming style, we should anticipate the kinds of choices that will come up when we design Phase-A, and ask relevant questions. For example an interview with the designers of MetaCrop diagrams might involve showing Figure 8.28 (page 145) and asking whether the "not well-oriented" layouts really should be prohibited. Our examination of existing diagrams suggests that it is so, but an expert might be able to point out some exceptions.

Most of the work in applying DiAlEcT to MetaCrop diagrams fell in the design of Phase-A. Meanwhile Phases-D and -E succeeded for the most part in directly borrowing ideas already explored in HOLA. As for Phase-T, we finally examine that both in general and for SBGN in particular in the next chapter.



Chapter 9

Interaction



Chapter 7 described the DiAlEcT framework, and while the basic idea of Phase-T ("Transform") was indicated there, it is in this chapter that we begin to design specific interaction methods.

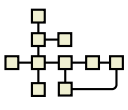
The methods we develop belong to two categories: *natural transformations*, and *GSA transformations*. The former consists of common types of transformation that people like to perform when editing network layouts by hand, such as flips and rotations. We are fortunate to have the *dynamic data* from the Orthowontist experiment (Chapter 4), which we inspect in Section 9.1 in order to motivate development of natural transformations. Recall that by dynamic data we mean the full layout processes recorded in the experiment, as opposed to the static data, by which we mean the final layouts created by those processes.

Meanwhile GSA transformations are special techniques for switching among the geometric and structural variants of the GSAs (graph substructure arrangements) created in Phase-A of DiAlEcT. As an example we develop transformations in Section 9.2 for the parallel chain group and regular polygonal face patterns from Chapter 8.

An illustrative example in Section 9.3 shows how the transformations developed in this chapter could be used in a particular case, and also points out difficulties.

9.1 Natural Transformations

In Sections 9.1.1 and 9.1.3 we examine two key examples selected from the dynamic Orthowontist data, to motivate the design of interaction methods. The first of the layouts to be considered below (Section 9.1.1) shows a "straightforward" approach, in that the process aligns well with the four phases of the DiAlEcT framework. This was the highest-voted layout of Graph 8 from the Orthowontist study, so it is referred to as Layout 8A—"8"



for the graph number, "A" meaning highest-voted. The second layout to be considered below (Section 9.1.3) shows an "iterative" approach, beginning with an "initial layout" and followed by what we may interpret as several "transformations". This was the third-highest-voted layout of Graph 6 from the study, so it is referred to as Layout 6C.

Recall that when we examined the static data (final layouts) from the study in Chapter 4, we considered various metrics like stress, symmetry, and others. As we now examine dynamic data (layout process) we will look at stress and symmetry again. This will help us to understand what is happening as the layout proceeds. In the charts considered in this section the horizontal axis will give the "frame number" i.e. the number of steps through the layout process, with each node drag or edge drag counting as a step, while the vertical axis will give the value of the metric in question. In other words, these charts show the change in a particular metric over time as a designer (i.e. study participant) works toward what they deem to be a good layout.

Symmetry is measured in the same way it was in Chapter 4 (see Section D.2). For stress measurements (Section D.1) we take as the ideal edge length the one suggested by the final frame of the layout, i.e. the average straight-line distance between neighbouring nodes in that frame. The rationale for this choice is that when study participants finished their layouts they were indicating that this layout looked good, in particular that the nodes were now spaced as they should be. Defining the ideal edge length in this way does not guarantee low stress in the final layout; rather the final stress is bounded below by the final variance in distance between neighbouring nodes.

9.1.1 A Straightforward Approach

There is little reason to suppose *a priori* that when human beings create network layouts by hand they should operate according to the phases defined by the DiAlEcT framework. On the other hand, we should not be surprised to find that they do, since the system was designed to produce human-like results. In order to demonstrate behaviour conforming to DiAlEcT a human layout designer would have to (1) quickly decrease the stress of the layout (Phase- D), then (2) choose an arrangement for the nodes (Phase-A), and (3) fix up any obvious errors (Phase-E), before possibly (4) transforming the layout (Phase-T). In fact the designer of Layout 8A did something quite close to this.[1]

The initial layout of Graph 8 is as in Figure 9.1a. From here the designer immediately began to untangle the graph, and by Frame 20 the nodes had been brought into a reasonable distribution in the plane (Figure 9.1b). This corresponds to the precipitous drop in stress visible in Figure 9.1g, reaching a low by Frame 20 that was roughly maintained for the remainder of the process.

This initial activity would make for a reasonable "Phase-D" except that "Phase-A" appears to have been going on simultaneously. The basic arrangement of nodes was nearly settled already by Frame 20, and in seven more frames (Figure 9.1c), it was complete. Note also that symmetry (Figure 9.1h) had already risen somewhat by this point.

It seems reasonable that a human layout designer would begin creating an arrangement even while decreasing stress. If you drag nodes one at a time, you must choose a place to drop each one before picking up the next. This provides a strong incentive to make reasonable choices as you go: each node drop necessitates a choice right now, and it pays to make a good one.

[1] Interestingly however, among the text responses recorded in Appendix C the response from the designer of Layout 8A (number 6) does not seem nearly as close a description of phases D-A-E as does response number 11. The latter states, "Mostly I was trying to make the links as clear as possible, and then achieve some kind of overall shape. Towards the end I began to focus more on the aesthetics rather than just the practicalities."

Comparing the layouts in Frames 27, 41 and 54 (Figures 9.1c, 9.1d and 9.1e) shows a clear "Phase-E": the basic arrangement of the nodes stays the same, but by Frame 41 defects in alignment have been removed, and by Frame 54 defects in spacing have been removed. This corresponds quite closely with the two activities of aligning and distributing in HOLA's Phase-E.

Finally, something happened in the last four frames of the layout (compare Frames 54 and 58) going outside of the D-, A-, and E-phases. Here the designer demonstrated the need for "Phase-T" by now *transforming* the layout, and choosing a new arrangement for the nodes. Two changes were made, and the symmetry jumped up correspondingly, as is visible in Figure 9.1h.

The two final transformations to the layout suggest simple interaction techniques that could facilitate the making of such changes.

Neighbour rotation

To begin with, consider the layout that was reached in Frame 54, as shown in Figure 9.2 with two nodes labelled A and B. Supposing this layout had been computed by the automatic phases of a DiAlEcT layout process, and the final relative constraint matrix for that process was M, then we would likely have $M(A, B) = \mathbb{E}$. Our first transformation could then be achieved by changing this constraint to $M(A, B) = \mathbb{W}$ and applying the DESCEND operation.

Any transformation of this kind, which seeks to alter the compass-direction constraint between two nodes, could be achieved by a simple interaction gesture. As illustrated in Figure 9.2, a *pivot node* p (circled in red) could be marked, and a *radial node* r (circled in green) could be dragged around the pivot node to the desired point and then dropped. Then the new direction from p to r would be set in M as a compass direction, overwriting the previous value of $M(p, r)$.

While this interaction could be performed simply enough with a mouse, clicking once to mark the pivot and then dragging the radial node, it seems like a very natural application for a touch-enabled device. In that case the user could simply touch one finger on the pivot node and drag the radial node with another finger.

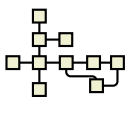
Chain rerouting

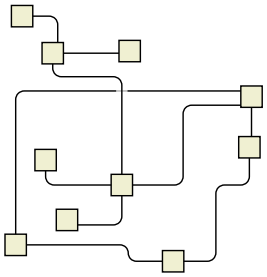
In Figure 9.3, nodes A, B, C form a chain, with node T as its terminal node at each end (making it a cycle). A simple interactive chain rerouting mode might allow the user to first select a chain (perhaps by selecting its terminal nodes), and then have the orthogonal chain routing procedure from HOLA continually applied automatically, as nodes are moved manually.

As Figure 9.3 illustrates, in this editing mode the user could simply drag node A to the right after selecting node T as terminal. The system would then automatically determine the best places for the bends in the chain routing, considering the node positions as fixed, and it would create the two bend points seen in Figure 9.1f, as desired.

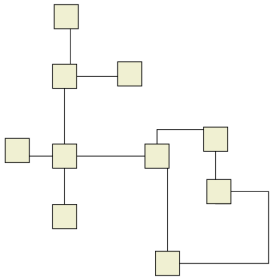
9.1.2 Stress Phenomena

Discussion of the dynamic Orthowontist data is aided by introducing a way of referring to phenomena in the stress graph associated with the layout process. Figure 9.1g exhibits three clear examples of what we may call a *stress hill*: a rise followed by a fall in stress. There are small stress hills from Frame 23 to Frame 27 and again from Frame 44 to Frame 49 in that graph; there is a large hill from Frame 0 to Frame 20 within which we can even discern small superimposed hills.

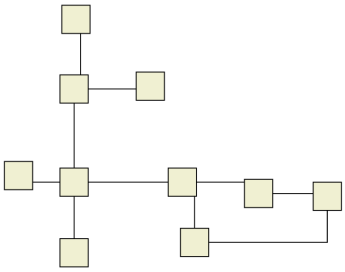




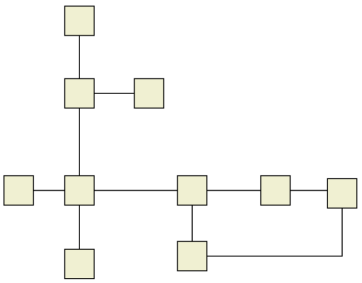
(a) (Frame 0) Graph 8 initially



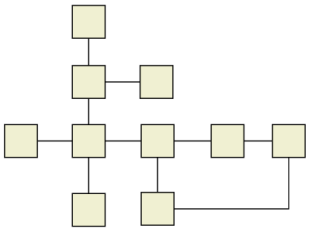
(b) (Frame 20) After reducing stress



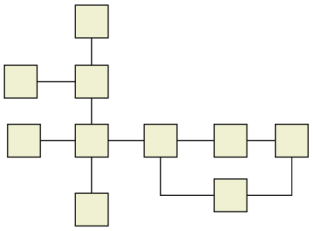
(c) (Frame 27) Configuration has been chosen



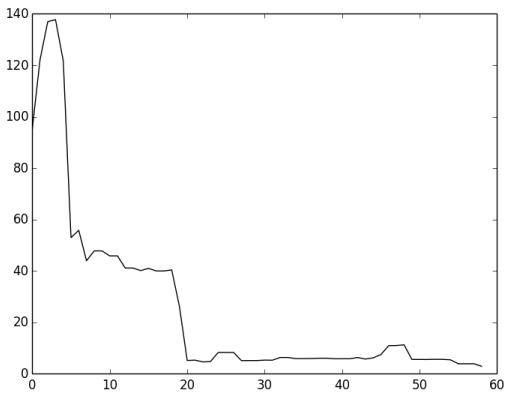
(d) (Frame 41) Defects in alignment have been removed



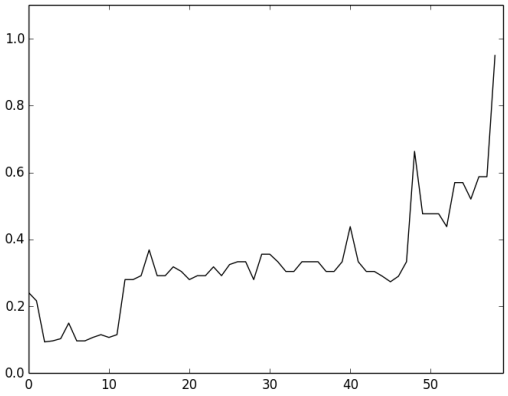
(e) (Frame 54) Defects in spacing have been removed



(f) (Frame 58) The layout has been transformed



(g) Stress



(h) Symmetry

Figure 9.1: Selected stages of Layout 8A, together with charts of stress and symmetry metrics over the course of the layout process

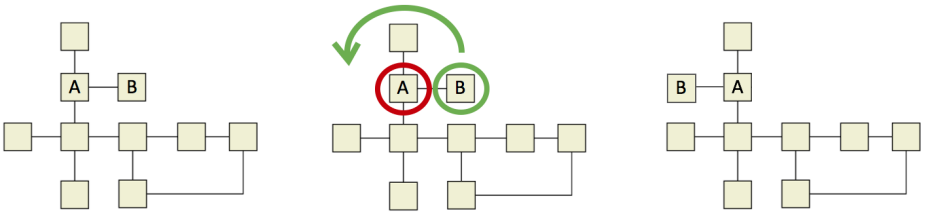


Figure 9.2: Interactive neighbour rotation: Node A is held fixed, while the user drags Node B relative to A. The place where node B is dropped indicates to the system which transformation is desired.

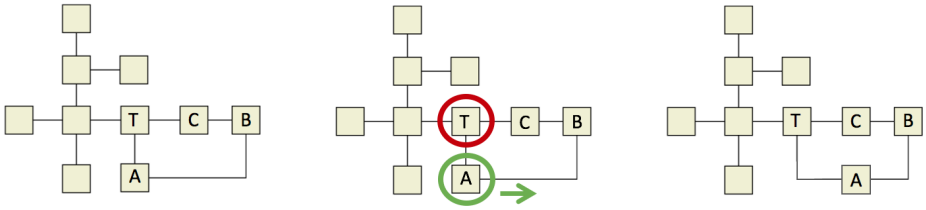


Figure 9.3: Interactive chain rerouting: The terminal node T is selected in order to indicate the chain A, B, C that is to be rerouted. Then the user can drag node A to the right, and the system will automatically choose the connector between nodes A and T as the best place to create a bend point, using the same procedure as in HOLA.

In watching the playback of the Orthowontist layout processes we find that stress hills tend to correspond to one of a few behaviours: (1) compacting (as in Frames 41 to 54 of Layout 8A), (2) distribution and untangling (as in Frames 0 to 20 of Layout 8A), and (3) *transformation*, examples of which we will consider in Section 9.1.3. The two transformations at the end of Layout 8A were simple enough that stress only decreased, but in the next example we will see more drastic transformations with corresponding stress hills.

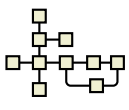
In all cases the explanation for the increase followed by decrease in stress is simple: If several nodes begin near one another but are moved to a new location, again near each other though perhaps arranged differently, the stress first goes up as the nodes are dragged one at a time and the cluster is broken apart, then falls back down as the remainder of the cluster "catches up", thus varying somewhat like the length of an inchworm.

Although participants in the Orthowontist study were provided with a "marquee select" tool with which to select several nodes at once and move them *en bloc*, they tended not to use it, and stress hills resulting from the piecemeal movement just described were common. Interestingly, this suggests that even though the user (probably) has a target arrangement in mind they move towards it via the simplest individual steps rather than the fewest actions.

9.1.3 An Iterative Approach

Whereas the previous example exhibited a relatively straightforward application of the four phases of the DiAlEcT framework, once each, and in the right order, the next example (Figures 9.4 and 9.5) shows something different. Most of what goes on can be filed neatly enough under one of the four activities we refer to in the framework by their initials, D, A, E, and T, but there are repetitions. This time the entire process can be viewed as consisting of an initial layout, followed by three transformations.

The layout begins with a straightforward stress reduction up to Frame 14 (see Figures 9.4b and 9.5d). At this point is seems that the designer suddenly envisioned an entirely different configuration for the layout. It seems likely that the simplicity of the



low-stress layout in Frame 14 was essential in allowing the designer to perceive the structure of the graph clearly enough to now conceive of a new way of arranging the nodes. Perhaps the familiarity developed by this point also helped. The new arrangement seems to have been envisioned at this moment in almost all, if not quite all, of its details, for the layout now converges directly toward the new configuration except for a few details that remain to be played with.

There comes next a massive stress hill peaking at Frame 21 as the designer transformed the layout. As seen in Figure 9.4c, three nodes were drawn far over to the left, and others stretched far down to the lower right in order to make room for other nodes to proceed toward the left.

The stress plummeted back down by Frame 28. If the node arrangement in this frame had been desirable, we would have expected the designer to now commence with *emending*, that is, cleaning up the layout as appropriate to Phase-E; instead, another stress hill followed, corresponding to another transformation. This next hill peaks at Frame 30 (Figure 9.4e), and suggests a desire to move the small node from the top of the layout in Frame 28 down to the bottom. The "plateau" across the top of this stress hill from Frame 30 to 40 corresponds to an "interim Phase-E" inserted here, after which the three nodes sitting together under the oblong node top-right have been laid out neatly, and their connector routes straightened (Figure 9.4f). Finally the stress hill ends in Frame 42 (Figure 9.5a), suggesting a retreat from the idea of moving the third small square node to the bottom side of the layout, instead placing it on the right-hand side.

The retreat could perhaps be attributed to a feeling of "not wanting to bother" with the work it would take to achieve the desired transformation; however, over the final stress hill from Frame 42 to 49 the layout designer appears to have decided that the transformation was "worth the work" after all. By the end of the final hill the layout is transformed as in Figure 9.5b. There are then many frames of emending (Phase-E), reaching the final layout in Frame 75 (9.5c).

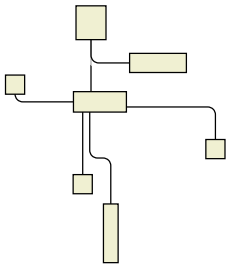
Thus, while we cannot know what this designer was thinking during the layout process, it seems easy to tell a reasonable story to explain what was going on. In particular, if the two hypotheses enunciated above hold any truth then the plan to develop both automatic and interactive layout tools seems well-motivated, insofar as it matches the approach that appears to have been taken in Layout 6C. First, the automatic phases produce a reasonable initial layout (Figure 9.4b), from which the user can hope to at least understand the graph's structure. This layout may be accepted as final, or else the user may employ the interactive phase to transform the layout one or more times. If the user finds it hard to envision the desired layout all at once, the interactive phase can be viewed as iterative and exploratory: several transformations may allow the user to eventually discover the best layout.

Interactions

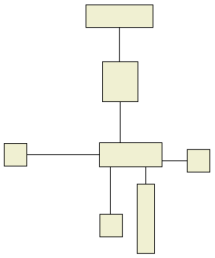
We first consider what sort of interactions could facilitate the transformation from the layout of Frame 14 to that of Frame 42. This suggests expanding the capability of the rotation transformation of Figure 9.2 by allowing a whole chain to be rotated at once. Secondly, we consider how to get from the layout of Frame 42 to that of Frame 75, and for this introduce a technique for flipping a subgraph.

Whole chain rotation

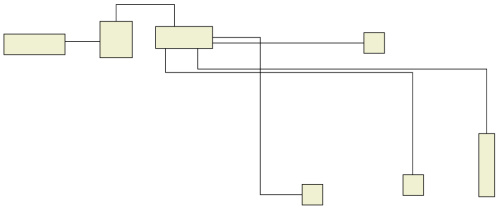
It is desirable to be able to rotate a whole chain of nodes at once. As in the discussion of neighbour rotation associated with Figure 9.2 (page 159), there should again be a pivot node and radial node; however, after the pivot node p is selected the user may be allowed



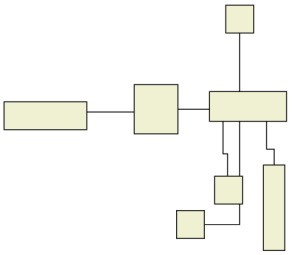
(a) (Frame 0) Graph 6 initially



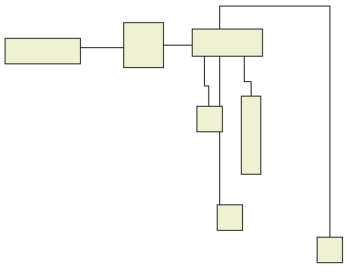
(b) (Frame 14) Initial straightforward stress reduction



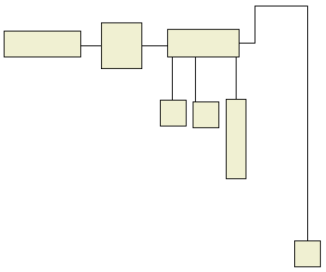
(c) (Frame 21) Network is stretched out on the way toward the new configuration



(d) (Frame 28) Second stress minimum

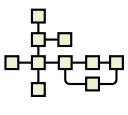


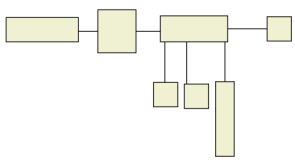
(e) (Frame 30) Second stress peak



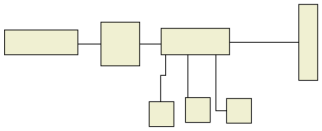
(f) (Frame 40) End of the interim emending plateau

Figure 9.4: Selected stages of Layout 6C. More stages and stress chart are in Figure 9.5.

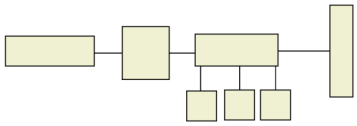




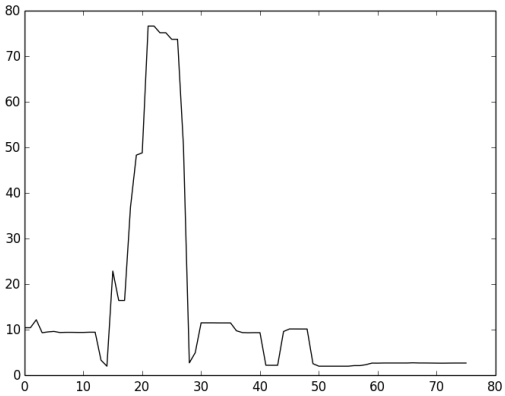
(a) (Frame 42) After second stress hill



(b) (Frame 49) After third stress hill



(c) (Frame 75) Layout complete



(d) Stress

Figure 9.5: Selected stages of Layout 6C are shown, together with stress chart. See also Figure 9.4.

to choose a radial node r that is not an immediate neighbour of p, with the idea that some whole path of nodes from p to r would be selected for rotation.

There are different ways in which the path could be selected. It could be required that there be a unique path from p to r (or else an error message would be given and the choice of r rejected). Alternatively the requirement could be relaxed to the need for a unique *shortest* path from p to r. Or, dropping all restrictions, in the event that there were multiple shortest paths from p to r the several paths could be highlighted in some way and the user asked to choose from among them.

Let us assume then that the nodes p and r and the path between them have been chosen, and write $p = v_0, v_1, \ldots, v_n = r$ for the nodes along this path, including p and r. The user would drag r into some compass direction $\mathbb{X}$ from p and drop it. The system would then set the constraints $M(v_{i-1}, v_i) = \mathbb{X}$ for i from 1 to n, and DESCEND, reorienting the chain as desired.

With this new feature the transformation from Frame 14 to Frame 42 of Layout 6C could be achieved in two motions, as illustrated in Figure 9.6. However, this example raises the question of what happens when the radial node is dropped on a side of the pivot node that already has one or more neighbours. In Figure 9.6 we have simply assumed that the system would evenly distribute all the neighbours on a given side, but it is worth asking whether this would be the desired or expected behaviour. To properly answer that question would require a user study which could be a part of future work, but here we can at least consider the alternatives, which we do in the next section.

Rotation propagation

In pivot-based rotation, what should happen when radial nodes share a side? If we imagine that the pivot node p has four available sides (one in each compass direction), then one

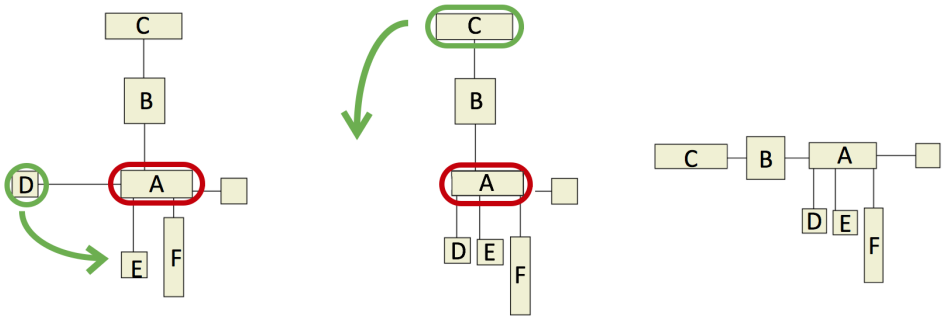


Figure 9.6: With the ability to rotate whole chains, the transformation from Frame 14 to Frame 42 of Layout 6C can be achieved in two motions. First node A is selected as pivot and node D dragged to its south side. Next, with node A still selected as pivot, node C may be dragged to its west side, and the system will automatically rotate the entire chain A, B, C.

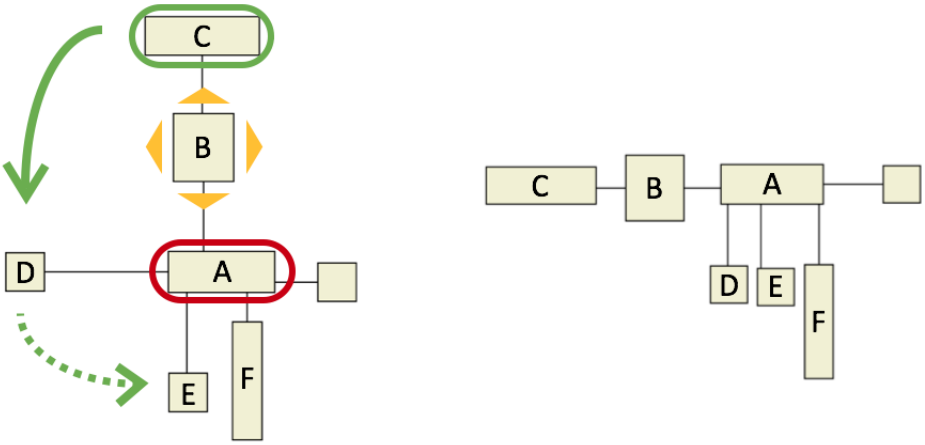
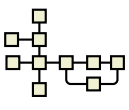


Figure 9.7: In the layout of Frame 14 (Figure 9.4b), node A is set as the pivot, node B is set to be repulsive, and the user drags node C over to the west side of A. It is automatically inferred that the entire chain A, B, C is to be rearranged to point west from node A. Since node B was marked repulsive node D is ejected from the west side of A and sent to the south side. There it is spaced evenly with the other nodes E and F that occupy that side, creating the layout of Frame 42 (Figure 9.5a).

of two things could be expected to happen when we rotate neighbour r_1 into a side of p that already has one or more neighbours $r_2, r_3, \ldots$ occupying it. First, r_1 could simply share this side with the other neighbours. Alternatively, if r_1 were somehow marked as a *repulsive* node, then all current neighbours could be ejected, pushing them onward to the next side in the same rotation direction and allowing r_1 to now be the sole occupant of their former side.

As Figure 9.7 illustrates, for the desired transformation in Layout 6C it would suffice to mark node B as repulsive, set node A as the pivot, and drag radial node C to lie west of node A. In that case the whole chain A, B, C would be reoriented to point west from A, the repulsive node B would eject node D from the west side of A, sending it onward to the south side, and then nodes D, E and F would be readjusted to evenly share the south side of node A.

Using such "rotation propagation" techniques the transformation from Frame 14 to that of Frame 42 of Layout 6C could be achieved by requesting just a single rotation, but additional inputs would be required in order to make repulsiveness settings. Alternatively, as we already saw, the desired transformation can be achieved without any propagation system using two manual rotations. In future work a user study should be conducted in order to determine which of these interaction methods users prefer, and what kind of



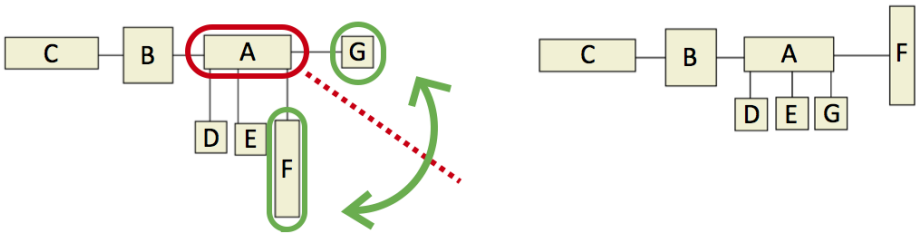


Figure 9.8: In the layout of Frame 42 (Figure 9.5a), node A is marked as a cut node. Then nodes F and G are selected to participate in a diagonal flip. This produces the layout of Frame 75 (Figure 9.5c).

behaviour they find intuitive. For now only simple rotation (without propagation) has been implemented. In any case these methods provide a substantial improvement over the 28 frames the designer spent making this transformation in the study.

Flips

Finally, let us consider how to get from the layout of Frame 42 to that of Frame 75 in Layout 6C. As Figure 9.8 illustrates, what we desire is to *flip* a part of the graph. In this case it would be a *diagonal* flip, but in general we would also want to allow *horizontal*, *vertical*, and the *alternative diagonal* flip as well.

We will in general want to be able to select a subgraph to be flipped, and the present case suggests a reasonable approach. Node A, which lies on the axis of the desired flip (or reflection), can be thought of as a *cut node*, whose deletion would split the remaining graph into several connected components. After selecting the cut node we may then select which of the resulting components is to participate in the flip, or else choose more cut nodes if the components need to be further subdivided.

In Figure 9.8 marking node A as cut node would leave five components, of which we would choose just nodes F and G (which are single-node components) to participate in the flip.

In general, when all cut nodes c_i and all participating components P_j have been selected, then, regarding the P_j as sets of nodes, we would form the set

$$\mathscr{F} = \{c_i\} \cup \bigcup_j P_j$$

and flip the arrangement of all nodes in $\mathscr{F}$ by editing the relative constraint matrix M, followed by a DESCEND operation.

The way in which the entries of M are to be changed is simple, but requires the introduction of some formalism to be expressed precisely. First we must settle on the meaning of the phrases, "horizontal flip" and "vertical flip". By a "horizontal" flip we might mean one in which the horizontal or x-coordinates of nodes are changed, or we might mean one in which nodes are flipped over a horizontal axis. Unfortunately these are opposite meanings. However, two popular vector graphics editors, INKSCAPE and Microsoft POWERPOINT, both assign the name "horizontal flip" to the former meaning, i.e. that in which the x-coordinates of objects will change (as they are flipped over a vertical axis) and we will therefore adopt the same convention.

With this convention in place we denote horizontal and vertical flips by H and V respectively. By D we will mean the diagonal flip over the axis of slope $+1$ in the graphics plane as defined in Section 8.2.2 (thus, running from upper left to lower right), while by D′ we will mean the diagonal flip over the axis of slope -1. Table 9.1 lists for each flip

Flip	Pairs of letters to be swapped in RCM
H	$(\mathbb{E},\mathbb{W})$, $(\mathbb{R},\mathbb{L})$
V	$(\mathbb{S},\mathbb{N})$, $(\mathbb{D},\mathbb{U})$
D	$(\mathbb{E},\mathbb{S})$, $(\mathbb{W},\mathbb{N})$, $(\mathbb{R},\mathbb{D})$, $(\mathbb{L},\mathbb{U})$
D′	$(\mathbb{E},\mathbb{N})$, $(\mathbb{S},\mathbb{W})$, $(\mathbb{R},\mathbb{U})$, $(\mathbb{D},\mathbb{L})$

Table 9.1: For each flip those pairs of direction letters are listed that must be swapped in all entries $M(a,b)$ of the relative constraint matrix (RCM) for which $a,b \in \mathscr{F}$, when the flip is to be performed over the set of nodes $\mathscr{F}$.

Rotation	Cycles of letters to be permuted in RCM
90 degrees clockwise	$(\mathbb{E},\mathbb{S},\mathbb{W},\mathbb{N})$, $(\mathbb{R},\mathbb{D},\mathbb{L},\mathbb{U})$
90 degrees anticlockwise	$(\mathbb{E},\mathbb{N},\mathbb{W},\mathbb{S})$, $(\mathbb{R},\mathbb{U},\mathbb{L},\mathbb{D})$
180 degrees	$(\mathbb{E},\mathbb{W})$, $(\mathbb{S},\mathbb{N})$, $(\mathbb{R},\mathbb{L})$, $(\mathbb{D},\mathbb{U})$

Table 9.2: For each rotation the necessary permutations of letters in the relative constraint matrix (RCM) are given by listing disjoint cycles. Letters must be permuted in all entries $M(a,b)$ for which $a,b \in \mathscr{F}$, when the rotation is to be performed over the set of nodes $\mathscr{F}$.

the pairs of direction letters that must be swapped in all entries $M(a,b)$ of the relative constraint matrix for which $a,b \in \mathscr{F}$, when the flip is to be performed over the set of nodes $\mathscr{F}$.

9.2 GSA Transformations

In addition to the basic interaction methods described in the last section the DiAlEcT framework calls on us to provide special-purpose methods when necessary to switch between the structural and geometric variants of GSAs. In this section we consider appropriate methods for some of the GSAs created by HOLA and MCGL.

9.2.1 Hierarchical Symmetric Trees

Switching between the geometric variants of the hierarchical symmetric tree (HST) arrangement employed in HOLA is already partly served by the techniques described in Section 9.1. Recall that an HST has eight geometric variants, corresponding to the tree's *growth direction* and *flip bit* (see Section 5.3.3, page 69). Starting from a given arrangement of an HST, the flip technique given by Table 9.1 already allows switching among four of the eight variants. In order to reach the other four we are motivated to introduce a similar *rotation* technique.

For this, cut nodes and components may be selected just as in the flip technique, only now the system is asked to rotate the set of nodes $\mathscr{F}$ by 90 degrees in either the clockwise or anticlockwise direction, or else by 180 degrees. The required permutations of letters in the relative constraint matrix are given in Table 9.2. Readers familiar with group theory [Gal16] may note that in Tables 9.1 and 9.2 we are simply describing all the permutations in the *dihedral group* D_4 (except the identity) twice each, once for the cardinal directions $\mathbb{E}$, $\mathbb{S}$, $\mathbb{W}$, $\mathbb{N}$, and once for the lateral directions $\mathbb{R}$, $\mathbb{D}$, $\mathbb{L}$, $\mathbb{U}$.

While this new rotation technique can achieve some of the same things as the pivot-based rotation illustrated in Figures 9.2 and 9.7, the latter has the special feature of managing rotation propagation via repulsive and inert nodes. Moreover, the two techniques are independently valuable since they provide different ways of interacting, which may have differing degrees of intuitiveness in different situations, and for different users.

As for switching among the structural variants of an HST arrangement, we take this to mean changing the ordering of the child trees attached to a given parent node. For this

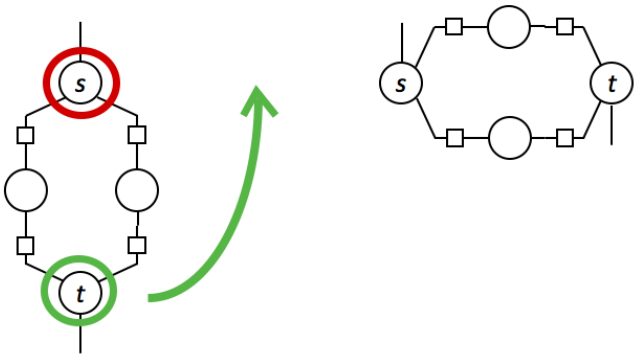


Figure 9.9: Either terminal of a PCG may be selected as pivot, and the opposite terminal dragged to a new cardinal direction with respect to the former (left). The entire PCG is then rotated automatically (right).

we can use a similar technique to the *pivot-based chain re-ordering* technique developed for parallel chain groups in Section 9.2.2 below.

9.2.2 PCG Interactions

Pivot-based Rotation

To the basic pivot-based rotation technique of Section 7.5 we add a new special case for PCGs. Namely, the user may mark either of the terminals s, t of the PCG as pivot, then drag the opposite terminal to lie in a new cardinal direction from the first, and drop it there. This causes the entire PCG to be rotated so that its axial direction is that indicated by the pivot interaction. See Figure 9.9.

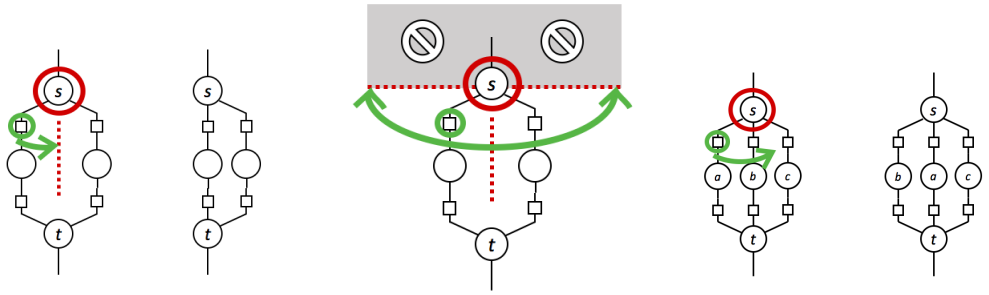
Pivot-based Chain Re-ordering

Another special case is added to the pivot-based rotation technique in order to allow the user to change the transverse order of the chains in the PCG. Again either terminal u is chosen as pivot. Then an internal chain node v adjacent to terminal u is dragged to a new position and dropped. Let C be the chain to which node v belongs. There are three cases to consider:

1. The radial node v is dropped in precisely the axial direction from terminal u, up to some tolerance ε for human-error (say ± 5 or 10 degrees—this can be configured by the user). Then chain C is aligned with both terminal nodes, i.e. is placed into the *centre position*. If another chain C' was already in the centre position then it is swapped with C, i.e. it is now configured to lie where C was before the interaction. All nodes in chain C are now aligned in the axial direction, if they were not so already. See Figure 9.10a.

2. The radial node v is dropped 90 or more degrees to either side of the axial direction of u (but at most 270 degrees). In this case the interaction is simply rejected; nothing changes. See Figure 9.10b.

3. The radial node v is dropped more than the tolerance ε but less than 90 degrees on either side of the axial direction from u. Then if $C_1, C_2, \ldots, C_n$ are the other chains besides C, and $v_1, v_2, \ldots, v_n$ the neighbours of u in these chains, respectively, then the slope of the vector $v - u$ is compared to the slopes of all vectors $v_i - u$ to find its immediate neighbours $v_{i_0} - u$ and $v_{i_1} - u$, and then the chain C is re-positioned between the corresponding chains C_{i_0} and C_{i_1}. See Figure 9.10c.



(a) If a chain is brought to the centre position it is aligned with both terminals.

(b) Rotating 90 degrees or more from the axial direction is prohibited.

(c) Dropping the chain between two others moves it to that position.

Figure 9.10: Either terminal of a PCG may be selected as pivot, and adjacent internal chain nodes may be rotated with respect to it, in order to change the transverse ordering of the chains.

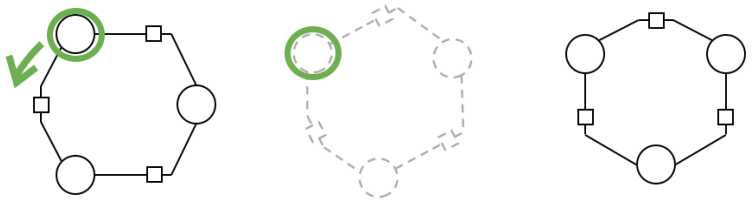


Figure 9.11: In RPF interaction mode, any node of an RPF may be rotated relative to the centre of the polygonal face. During rotation a dashed "ghost" representation is shown, and snaps to the nearest canonical rotation (centre). When the node is dropped then the RPF is rearranged in the new rotation (right), with constraints being altered and orientable nodes being reoriented as needed.

A similar technique could also be used to reorder the subtrees in the HST arrangement, as mentioned in Section 9.2.1.

9.2.3 RPF Interactions

Whereas the new techniques for PCGs introduced above are pivot-based, RPFs warrant introduction of a new, dedicated interaction mode. The user would select this mode, and then interact as described below.

Rotation

In RPF interaction mode, dragging any node of an RPF F in a circular arc relative to the centre c of F will cause F to be rotated. See Figure 9.11. However, only canonical rotations (Figure 8.25, page 140) may be selected in this way. Thus, as the user drags, the preview "snaps" to the nearest canonical rotation.

Dilation

In RPF interaction mode, dragging any node of an RPF F nearer or farther from the centre c of F will cause F to be dilated to the corresponding radius. See Figure 9.12.

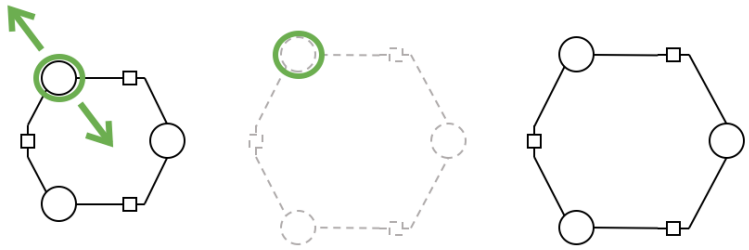


Figure 9.12: In RPF interaction mode, any node of an RPF may be dragged radially, relative to the centre of the polygonal face. While dragging a dashed "ghost" representation is shown (centre). When the node is dropped, the RPF is re-sized by increasing the separations in the constraints that define it (right).

9.3 Illustration and Issues

We illustrate Phase-T with an example, showing how the initial SBGN layout of Figure 8.33 (page 151) can be transformed. See Figures 9.13 through 9.17. All of these transformation methods have been implemented, and the figures show their actual results.

For the most part the figures demonstrate a fast and easy way to transform the layout. A couple of difficulties are also noted, which may point toward future work. For one, Figure 9.13 shows some undesirable edge-node overlaps. A technique should be developed to remove these both in Phase-E and after each transformation applied in Phase-T.

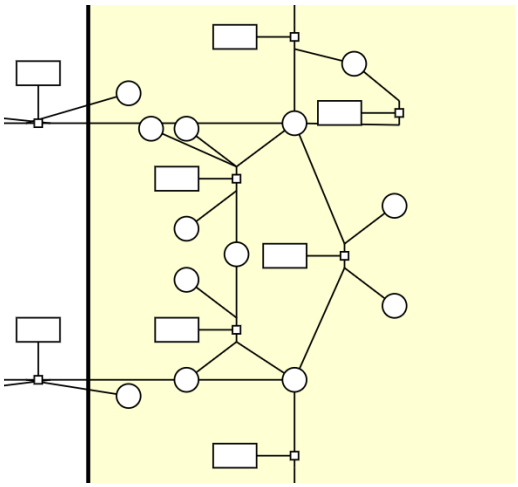
Another interesting issue comes up in Figure 9.16c. After a rotation is performed, one satellite node winds up much farther away from its parent node than we would think appropriate. This results in ugly edge crossings and clutter. This is not however a "bug" in the rotation procedure; it falls within the range of expected (if occasionally undesirable) behaviour. Like all the high-level transformations we have developed, the rotation is put into effect simply by the automatic rewriting of a few entries in the relative constraint matrix. In the present case it so happens that the satellite that went astray simply was never constrained to stay nearby its parent node. Recalling that DESCEND is an iterative procedure in which PROJECT alternates with gradient descent steps, it is easy to imagine how the initial PROJECT could put the stray satellite too far away. While stress minimisation would tend to force the satellite to return to a more desirable position, we are also employing the overlap-prevention technique during the DESCEND, which in this case prevents the satellite from passing through other nodes lying in its way.

The first solution that presents itself is of course to apply DESCEND twice: the first time without overlap prevention, and the second time with it. However, since this kind of problem seems rare, we must ask which is more annoying to the user: having to correct such a problem manually as shown in Figure 9.16c, or always having to wait through two DESCEND operations instead of one? This sort of question could be explored in a proper user study, as a part of future work.

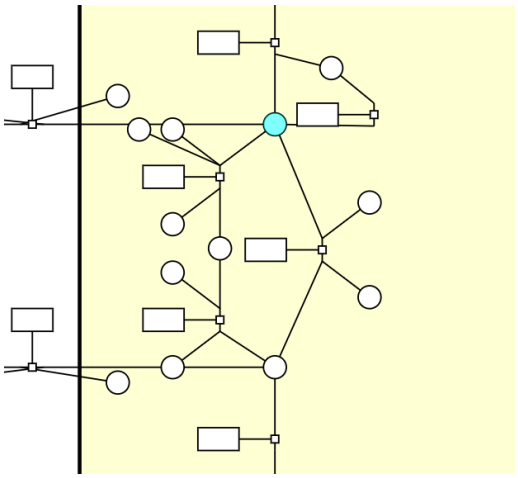
9.4 Conclusions

We begin with a summary of the devised interaction methods:

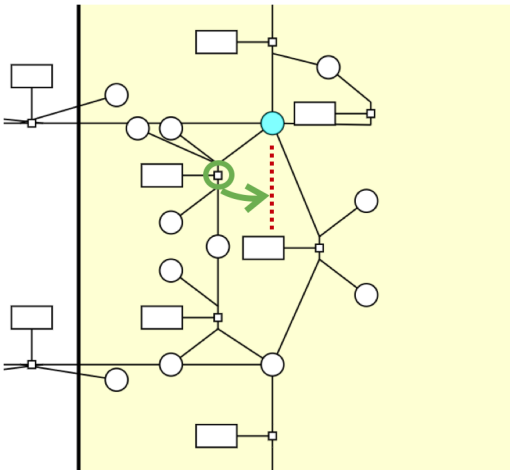
1. Pivot-based Rotation
    - One node is fixed as the *pivot node* p.
    - Another node is chosen as the *radial node* r.
    - The user drags the radial node to a new side of the pivot node.



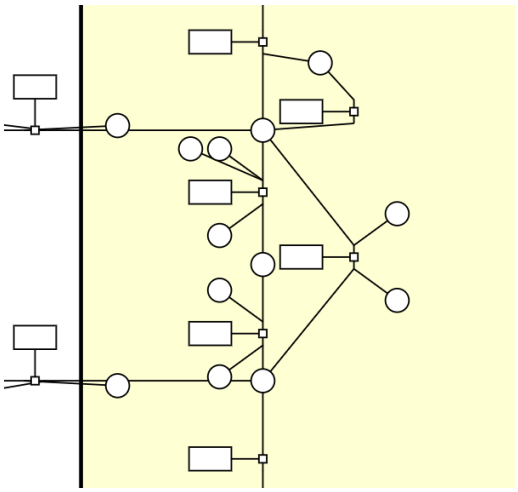
(a) We begin with the PCG near the centre of the diagram toward the right-hand side in Figure 8.33 (page 151). As in the actual MetaCrop layout of this network in Figure 8.32 (page 150), we might want the longer chain to be centred in the PCG.



(b) We select the pivot node (light blue).

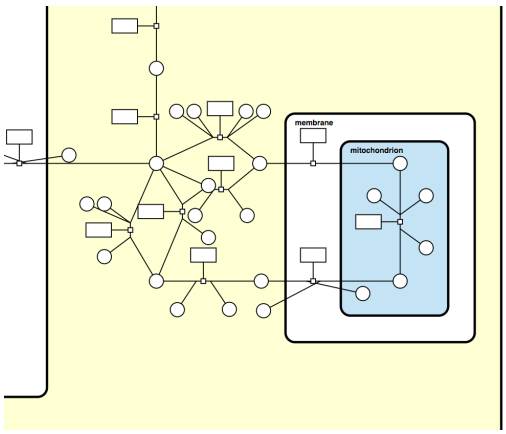


(c) The first process node in the long chain serves as the radial node. We grab it and drag it to lie directly south of the pivot.

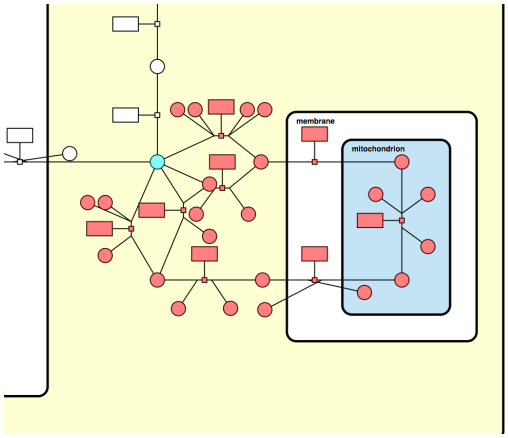


(d) On release the constraints are automatically updated and DESCEND is applied, yielding the desired new configuration. We may note one area for future improvement is the removal of edge-node overlaps, one of which is visible in this figure.

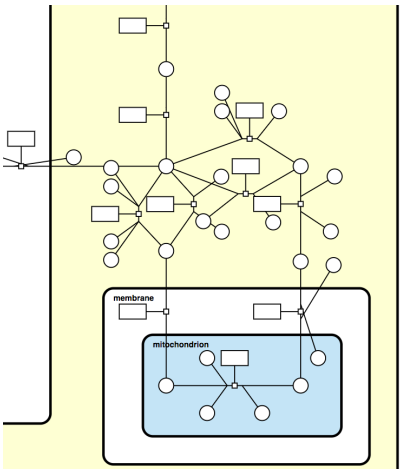
Figure 9.13: Transforming a parallel chain group



(a) We now consider the lower-right corner of Figure 8.33.

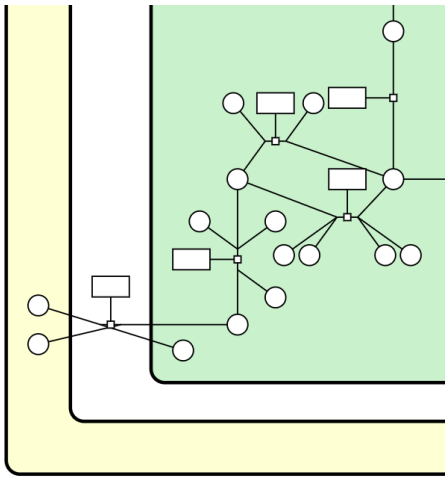


(b) The pivot node (blue) breaks the graph into two connected components. We select the component below and right of the pivot (red).

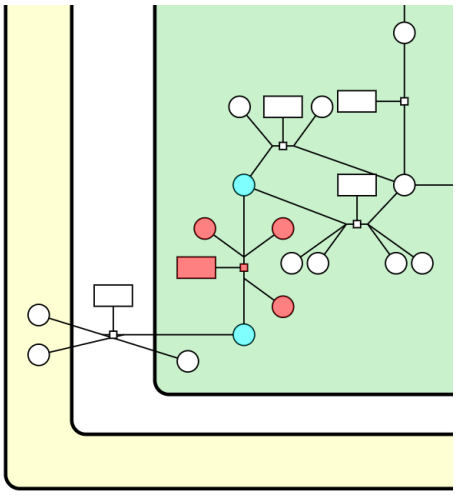


(c) With a single "diagonal flip" command, the chosen component is flipped. Internally all the necessary constraints are changed automatically and the stress of the layout is again decreased by DESCEND.

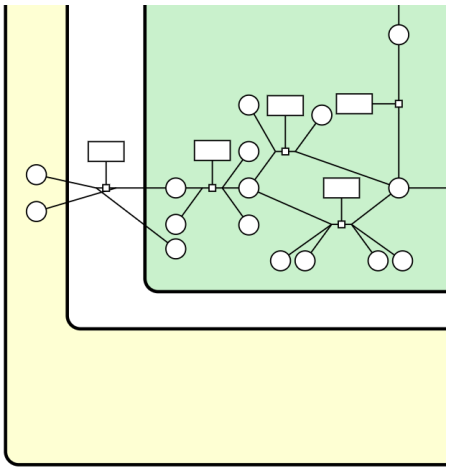
Figure 9.14: Diagonal flip



(a) We decide to straighten the pathway in the lower-left of Figure 8.33.

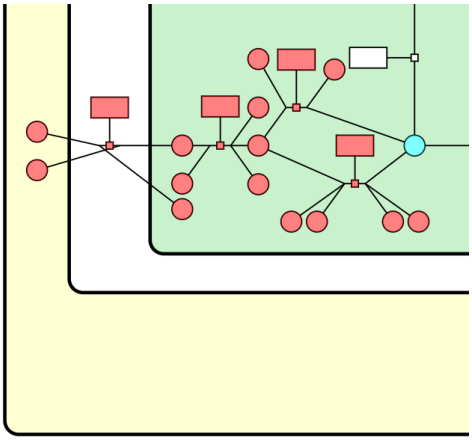


(b) This time we begin by selecting two pivot nodes (blue) and the component between them (red).

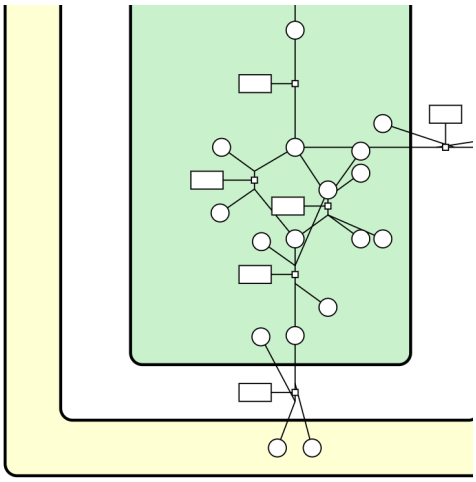


(c) A rotation command rotates the selected component ninety degrees clockwise.

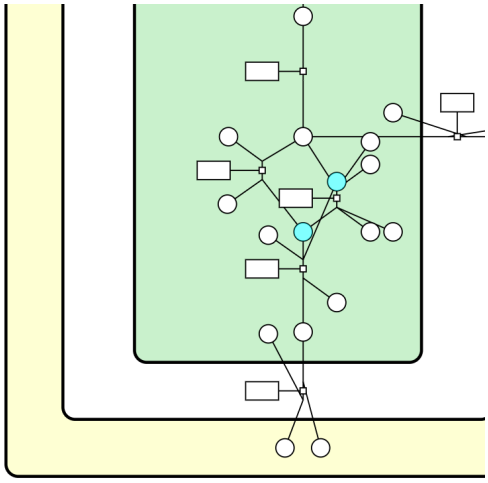
Figure 9.15: Rotation by component



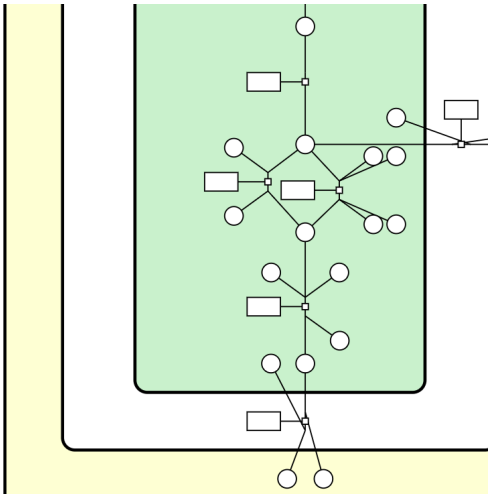
(a) We continue straightening the pathway in the lower-left of Figure 8.33, again selecting a pivot node and component.



(b) This time we rotate ninety degrees anticlockwise.



(c) Noticing that one of the satellites landed too far from the process node to which it belongs, we select this node and another one (blue). As opposed to the high-level transformations we have being using so far, we now make a low-level change to the relative constraint matrix, simply constraining the upper node to lie south of the lower one.



(d) The system again applies DESCEND in order to put our low-level change into effect, and now the layout is satisfactory.

Figure 9.16: Another rotation by component, plus a low-level separation

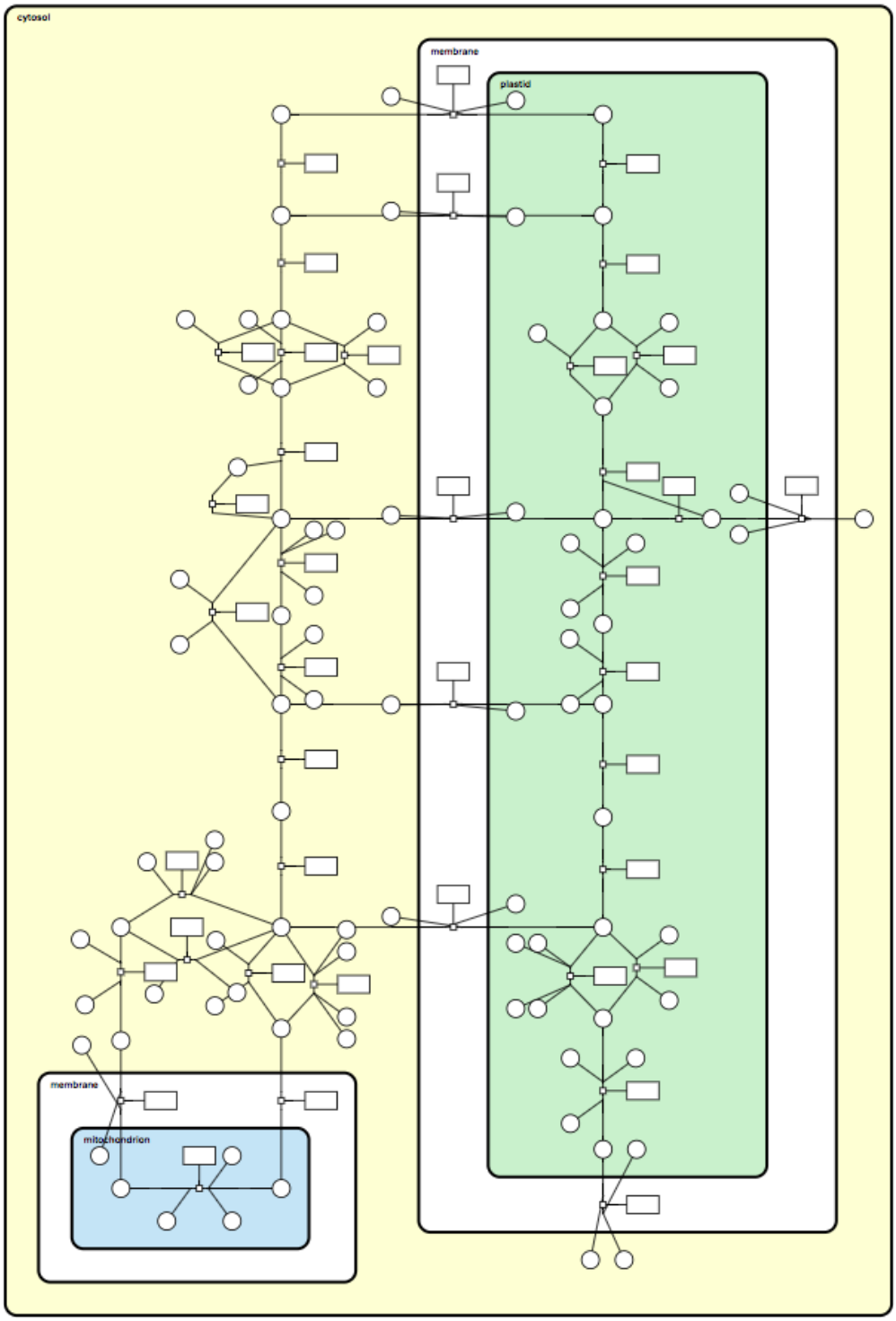


Figure 9.17: Applying one final horizontal flip to the entire network, we are left with a closer approximation to the hand-made MetaCrop layout in Figure 8.32 (page 150).

- If r is not a neighbour of p then the system determines a whole path of nodes to be rotated, possibly relying on user interaction.
- Based on the position where r is dropped relative to p, nodes are rearranged by editing the relative constraint matrix appropriately.

2. Componentwise Flips and Rotations
   - One or more nodes may be chosen as *cut nodes* c_i.
   - One or more of the connected components P_j resulting from deletion of the cut nodes are chosen.
   - The set of nodes $\mathscr{F} = \{c_i\} \cup \bigcup_j P_j$ is either flipped according to Table 9.1, or else rotated according to Table 9.2.
3. Chain Rerouting
   - A chain is selected, by selecting its terminals.
   - Nodes of the chain may be re-positioned one at a time, and the system automatically creates bend points as needed, according to the ORC arrangement system (see Table 7.2).
4. Parallel Chain Group Transformations
   - Rotation
     - One terminal is selected as pivot node.
     - The other terminal is dragged into a new compass direction from the pivot.
   - Chain Reordering
     - One terminal is selected as pivot node.
     - An internal node of a chain is dragged to a new angle relative to the pivot. If dropped in the centre it will be centre-aligned. If dropped between two other chains it will be re-positioned to lie between them.
5. Regular Polygonal Face Transformations
   - Angular and radial dragging of an RPF node relative to the centre of the RPF causes rotation and dilation, respectively.
   - When rotating, the RPF snaps to the nearest canonical rotation (see Figure 8.25, page 140).

With rotations and flips literally being types of linear transformation in vector geometry, the name "Phase-T" for "Transformation" seems apt. However, it must be remembered that the transformations described in this chapter are not merely geometric mappings; instead, they are achieved by altering the constraints in the relative constraint matrix (RCM) produced by the automatic phases, and then performing a DESCEND operation. This produces a low-stress layout with the new constraints, which enforce the desired new arrangement of nodes.

The initial list of interaction methods developed in this chapter is not believed to be complete. Just as we envisioned the catalogue of GSAs growing as our human-centred methodology is applied to new layout dialects in the future, there should be a corresponding growth in the catalogue of interaction methods to rearrange these. Moreover, even as the catalogue of high-level interaction methods grows we do not expect it to be "complete" in the sense of permitting arbitrary editing of the RCM. Therefore a good interaction system should also provide users with low-level control by allowing them to edit individual entries in the RCM directly.

Finally we reiterate that this work on interaction methods is only preliminary, and future work should include user studies to examine things like the usability of the methods proposed here, the best ways to resolve issues such as were discussed in Section 9.3, and any need for new interaction methods not yet considered.

Chapter 10

Conclusions

In the sections below we take some time to tie up a few loose ends. When examining the history of our subject in Chapter 2 we considered the ANDD system of Marks (Section 2.3) and the TSM system of Batini et al. (Section 2.4). The ANDD system sets the same goal as DiAlEcT's Phase-A of creating schematic arrangements or *perceptual organisations* of nodes, so one of the jobs left for this final chapter is to compare the approaches of DiAlEcT and ANDD. We do that in Section 10.2.

Meanwhile the TSM system is for creating orthogonal layouts and while we conducted a rigorous study to compare the yFiles implementation of TSM with our orthogonal layout algorithm HOLA in Chapter 6, that was before we defined DiAlEcT in Chapter 7. Therefore the present chapter is also a good place to compare the DiAlEcT and TSM frameworks, and we do that in Section 10.3. In support of that comparison we make some theoretical observations about stress and topology in Section 10.1. We conclude with thoughts on future work in Section 10.4.

10.1 Stress and Topology

In Chapter 2 we examined the varied origins of the stress function in both force-directed layout and multi-dimensional scaling. We also noted Nguyen, Eades, and Hong's observation that a stress-minimal layout is a faithful one. We can refine this idea a bit. The stress function defines a fitness landscape with many basins over the $2n$-dimensional space of all x- and y-coordinates when we have n nodes. The minimum points in one or more of these basins are global minima; others are local minima. In our use of CSML we have not striven to find global minima, but even if we did there could be multiple ones, so in all cases a stress-minimal position is *one* faithful interpretation or representation of the graph's structure, but other faithful representations are usually possible too.

Having noted this, perhaps we are now in a position to add one more piece to the puzzle, augmenting our understanding of the stress function's role in network layout. Namely, recalling that in Section 2.4.1 we noted the difference between the topology of a *graph* itself and the topology of a given *drawing* of that graph, perhaps we may now say that the stress function provides a connection between these two notions. That is, it is stress-minimisation that makes the topology of a drawing have something to do with the topology of the graph itself.

In a way this may sound obvious, once it is said. It may seem that this is just another expression of the faithfulness idea: We already observed that in a faithful layout geometric distances would reflect graph-theoretic ones, so "of course" planar topology should also reflect graph-theoretic topology, right? But to treat the issue this way is facile, for in order to truly understand *how* stress connects the two notions of topology we must rely on the

Heffter-Edmonds-Ringel Rotation Principle (recall Section 2.4.1), and this is a non-trivial result.

Let us unpack this one last time. The topology of a graph says what is connected to what, and by how long of a path. Minimising the stress function allows this information to be translated into a distribution of nodes in the plane. On the one hand this distribution also determines the drawing's topology via the Rotation Principle. On the other hand, the stress function divides the fitness landscape into basins, and altering the rotation system of the drawing tends to push it into a different stress basin (recall Section 5.2.2). Therefore any manipulation of the drawing that would change its topology would change its rotation system, which would likely move it into a new stress basin, which represents a new interpretation, or representation, of the topology of the graph. To recap: different "rubber sheet" topology of drawing leads to different rotation system, leads to different stress basin, leads to different representation of graph-theoretic topology.

What good are observations such as these? For one thing they help us to understand the comparison between DiAlEcT and TSM in Section 10.3. Apart from that, the hope is simply that they are in some way enlightening. For at the surface level it seems little more than a coincidence that two things associated with graphs should both be called "topology". On the surface, they use that word in very different ways. Graph-theoretic topology recalls directly Euler's sense of "what is connected to what" in the Bridges of Königsberg problem. On the other hand the topology of a drawing can be so called only after the development of the subject through nineteenth and twentieth-century mathematics, where the notion of "rubber sheet geometry" emerged. Through this historical development itself we can of course trace the connection between the two ways of using the word "topology", and yet the topological relation between graphs themselves and their drawings still seems tantalisingly out of reach. If stress minimisation provides a mechanism to help explain that relation, then this is something worth knowing. We may hope that future research will shed still more light on this puzzle.

10.2 Comparison of DiAlEcT with ANDD

The rule-based approach of the ANDD system was designed to arrange nodes much like the GSAs of DiAlEcT's Phase-A; however, recall from Section 2.3 that the system was implemented in Prolog and nodes were added to the diagram a few at a time as Prolog matched rules with those arrangements of nodes that had been created so far. The fact that all nodes instead have positions throughout the DiAlEcT layout process leads to important and fundamental differences with ANDD.

In DiAlEcT we start with Phase-D, bringing stress minimisation to bear from the beginning, and distributing nodes into natural starting positions, faithful to the graph-theoretic structure of the network. These positions continue to influence choices made in Phase-A, and each time we DESCEND we refresh this information. So for each node that has not yet been arranged we have some vital information: we know where that node would naturally land in a faithful layout of the graph, subject to the choices that have been made so far. We combine this information with the existing constraints each time we make a new choice.

An important advance of DiAlEcT is therefore the way it allows constraint choices to emerge as continually influenced by (a) a stress-minimal position for all nodes, and (b) the constraint choices that have been made so far. ANDD's rules too have "geometric variants" like DiAlEcT's GSAs, and at each stage an applicable variant may be chosen, but in ANDD there is no chance for as yet unplaced nodes to influence the choice, or to "react" to the choice and thereby maintain their stress-based influence on subsequent choices. Instead, only nodes already placed influence each choice.

We will find in Section 10.3 again that the existence and influence of positions for all nodes throughout the DiAlEcT process is an important feature differentiating DiAlEcT from earlier techniques.

10.3 Comparison of DiAlEcT with TSM

There are both intriguing similarities and important differences between DiAlEcT and TSM, and we consider these separately in the sections below.

10.3.1 Similarities

Among the similarities between DiAlEcT's automatic phases and TSM is that each can be characterised as a "refinement approach" to layout. We considered this aspect of TSM in Section 2.4.2. As for DiAlEcT, the Distribution phase achieves a first approximation to the final layout, then in the iterations of the Arrangement phase we continually refine the layout, making more and more decisions about the relative positions of nodes, and finally in the Emendation phase we add "finishing touches". So this is again a process of convergence toward a final layout by continual refinements.

There is also a strong analogy between the three automatic phases D-A-E of DiAlEcT and the three phases T-S-M of Topology-Shape-Metrics.

Phase-Decompose/Distribute versus Phase-Topology

Both distributing (by stress minimisation) and decomposing (peeling off trees as in HOLA or just one layer of leaves as in MCGL) have to do with topology. We have talked about the relation between stress minimisation and topology in Section 10.1. As for decomposing, we are simplifying the rotation system of each node by removing those neighbours whose position in the rotation system can be decided in a completely independent way.

Also in Phase-D we make a trade-off between stress minimisation and crossing reduction, and, as noted under *Application Guidelines* in Section 7.1, the topology-preserving stress minimisation technique of Dwyer et al. [DMW09b] may be employed. In such a case Phase-D would clearly be concerned with choosing a good topology.

Phase-Arrange versus Phase-Shape

The similarity between Phase-A and Phase-S seems obvious insofar as Phase-A is all about arranging GSAs into recognisable *shapes*. But that comparison is actually a bit shallow, since what these two phases compute is very different (constraints versus bend sequences).

Still, Phase-A's encoding of node arrangements in the relative constraint matrix means that absolute node positions are not being determined, only relative positions, and this is similar to the role of TSM's Phase-S which, by determining the sequence of bends in each connector establishes something about the relative positions of nodes without fixing their exact positions. Both Phase-A and Phase-S are concerned with *angles*: Phase-A in determining the compass direction from one node to another, Phase-S in choosing bend directions.

Phase-Expand/Emend versus Phase-Metrics

Both emending and expanding are, in part, about metrics. Emending is about metrics in dealing with even distribution (via neighbour-stress reduction). Expanding is obviously about metrics, since we are forcing nodes apart from one another via constraints in order to make room for reinsertion of any "leafy nodes" that may have been peeled off in Phase-D.

10.3.2 Differences

Despite the strong analogy between D-A-E and T-S-M there are important differences, both in DiAlEcT's ability to be flexible and make trade-offs, and, again, in the role of node positions throughout the DiAlEcT process.

Flexibility and Trade-Offs

Where TSM puts a strict monotonic order on aesthetics, DAE takes a more flexible and"human" approach by making trade-offs and exceptions.

To begin with, by allowing an initial stress minimisation to decide the basic distribution of nodes in the plane, we make a trade-off between stress minimisation and crossing minimisation. By sometimes deliberately creating bend points where it would be possible to have none, HOLA's Phase-A works very differently from the Phase-S of GIOTTO-Kandinsky, making a trade-off between bend minimisation and other aesthetic concerns like maximising symmetry or again minimising stress.

The guideline in the DiAlEcT system against altering the rotation system in Phase-A means that we *try* to maintain the topology determined by Phase-D but we *may* alter it under compelling circumstances. This is more flexible and "human" than the strict approach of TSM in which the topology is absolutely fixed after Phase-T.

The Influence of the Current Layout

Just as in the comparison of DiAlEcT with ANDD, the presence of a complete layout at every stage of the DiAlEcT process is a major difference with TSM. This may be a very important way in which the functioning of a DiAlEcT layout algorithm is similar to what human graphic designers do when creating a layout by hand, at least in a scenario like the one presented by the Orthowontist experiment of Chapter 4, where a messy layout is given and the task is to improve it.

In such a scenario, what do we naturally imagine a person doing? And what did we actually see happening in the layout processes examined in Section 9.1? Roughly the story seems to be something like this: You (i.e. the layout designer) begin untangling the network until some structure starts to emerge. Then you start creating shapes and arrangements and patterns to best exhibit the structure of the network as you perceive it. As you work—and this is the most important point—your choices are probably continually influenced by the positions at which the nodes currently sit.

It is this last point about node placement choices being influenced by the current layout that might just be the most human thing captured by DiAlEcT, and the biggest difference between this and earlier approaches like ANDD and TSM.

10.4 Outlook

A number of challenges for future research have been noted in the foregoing chapters, chiefly in Sections 6.4, 8.6, and 9.3. Questions include how best to manage density; how to handle Type II SBGN diagrams; what are the best interaction methods for Phase-T layout transformations.

Several graph substructure arrangements (GSAs) were described in Chapters 7 and 8, while several transformation methods were described in Chapter 9. These should be thought of as the beginnings of catalogues that should continue to grow as our approach is applied to more layout languages and dialects in the future. They should also be refined as future user studies teach us more about what people want in an ideal layout assistant.

In this thesis we set out with the view of node-link diagrams as artefacts produced by labs with special house styles, and where those who design these diagrams by hand would benefit from a skillful, computerised assistant. The computer should be able to produce a reasonable initial layout on its own, and should then understand corrections expressed in a high-level language. In pursuit of this goal we have developed a human-centred approach to network layout algorithm design.

This approach begins with a formative study to gather data about hand-made layouts. This can mean something as involved as an experiment where participants' editing procedures are recorded, like we did in Chapter 4, or it can be as simple as inspecting existing manually-laid-out diagrams produced by a given laboratory like we did in Chapter 8. The right approach depends on whether static data is enough, or the developer wants to learn from dynamic data as well.

In the second step we translate the findings of the initial study into two layout tools: one automatic and one interactive. The DiAlEcT framework tells us how to do this. This framework was described in Chapter 7 based on lessons learned from the techniques developed in Chapters 3 and 5. It describes how Phases-D (Distribute/Decompose), -A (Arrange), and -E (Emend/Expand) are to be designed in order to build the automatic layout algorithm, and how to design Phase-T (Transform) for the interactive editor. The latter allows the user to transform the initial layout produced by the automatic phases in both high-level and low-level terms. An application of the DiAlEcT framework was demonstrated in Chapters 8 and 9.

The third step of the methodology is to test the new layout tools in a normative user study. In this thesis we conducted such a study in Chapter 6 for the automatic orthogonal layout algorithm HOLA developed in Chapter 5. In future work, the MCGL algorithm of Chapter 8 and the interactive layout tools described in Chapter 9 should also be evaluated by user study.

By applying this new method we have designed and tested the HOLA algorithm for human-quality orthogonal network layout. This is the first time a network layout algorithm has been designed based on human behaviour, and then shown in rigorous testing to perform comparably to hand-made layout. Our hope is that future work will include successful application of this new approach to many more layout dialects.

Appendix A

Gradient-Projection for Snap Functions

The DESCEND operation in ADAPTAGRAMS computes a descent direction and step size in terms of the gradient and Hessian (matrix of mixed second partial derivatives) of the goal function S. Namely, if $g = \nabla S$ and $H = \nabla^2 S$ then the descent direction is $-g$ and the step size is

$$\frac{g^T g}{g^T H g}.$$

(See for example [NW99] p. 47.)

The terms in g and H corresponding to *P-stress* are given in [DMW09b]. Here we give the terms corresponding to the following three functions,

$$\begin{aligned} N &= \sum_{(u,v)\in E} q_\sigma(x_u - x_v) + q_\sigma(y_u - y_v) \\ G &= \sum_{u\in V} q_\sigma(x_u - a_u) + q_\sigma(y_u - b_u) \\ E &= \sum_{e\in E_V \cup E_H} \sum_{u\in V} q_\sigma\left((\sigma - d(u,e))^+\right), \end{aligned}$$

which are the node-snap, grid-snap, and edge-node repulsion terms, respectively. For $\sigma > 0$ we define

$$\gamma_\sigma(z) = \begin{cases} 2z/\sigma^2 & |z| \le \sigma \\ 0 & \text{otherwise} \end{cases}$$

and

$$\eta_\sigma(z) = \begin{cases} 2/\sigma^2 & |z| \le \sigma \\ 0 & \text{otherwise.} \end{cases}$$

For node-snap forces we have

$$\frac{\partial N}{\partial x_u} = \sum_{(u,v)\in E} \gamma_\sigma(x_u - x_v)$$

and

$$\frac{\partial^2 N}{\partial x_v \partial x_u} = \left\{ \begin{array}{cc} -\eta_\sigma(x_u - x_v) & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{array} \right\} \qquad \frac{\partial^2 N}{\partial x_u^2} = \sum_{(u,v)\in E} \eta_\sigma(x_u - x_v)$$

and similarly in the y-dimension.

For grid-snap forces we have

$$\frac{\partial G}{\partial x_u} = \gamma_\sigma(x_u - a_u)$$

and

$$\frac{\partial^2 G}{\partial x_v \partial x_u} = 0 \qquad \frac{\partial^2 G}{\partial x_u^2} = \eta_\sigma(x_u - a_u)$$

and similarly in the y-dimension. Recall that (a_u, b_u) is defined to be the closest grid point to (x_u, y_u).

For edge-node repulsion forces we have

$$\frac{\partial E}{\partial x_u} = \sum_{e \in E_V} \operatorname{sgn}(x_u - x_e)\gamma_\sigma\left((\sigma - d(u,e))^+\right)$$

where x_e is the x-coordinate of a vertically aligned edge e, and

$$\frac{\partial^2 E}{\partial x_v \partial x_u} = 0 \qquad \frac{\partial^2 E}{\partial x_u^2} = \sum_{e \in E_V} \eta_\sigma(\sigma - d(u,e))$$

and similarly in the y-dimension.

Appendix B

Proof of Edge Coincidence Test

We prove the **Edge Coincidence Test** from Chapter 3. We begin with definitions and notation.

Definition: A *constrained graph* is an ordered triple $G = (V, E, C)$ where V and E are the sets of nodes and edges in the graph, and C is a set of separation constraints on the x- and y-coordinates of the nodes.

Notation: The results of this section hold for both directed and undirected graphs. Since the directedness of edges is completely irrelevant to our results, we write an edge whose endpoints are u and v in unordered notation $\{u, v\}$.

Definition: An *edge constraint* for a graph $G = (V, E)$ is a separation constraint $z_u + g \leq z_v$ or $z_u + g = z_v$ such that $\{u, v\} \in E$, i.e. a separation constraint on the endpoints of an edge.

Definition: An *edge-constrained graph* is a constrained graph $G = (V, E, C)$ in which C contains only edge constraints.

Notation: When C is a set of constraints and S a set of equations and inequalities on coordinates of nodes, we will use the entailment relation $C \vdash S$ to indicate that each relation in S is entailed by the constraints in C.

Definition: A constrained graph $G = (V, E, C)$ is said to contain a *horizontal overlay* when there are edges $\{a, b\}, \{c, d\} \in E$ such that

$$C \vdash \{y_a = y_b = y_c = y_d, x_a < x_d, x_c < x_b\}.$$

In this case we write $(a, b) \rightrightarrows (c, d)$. Similarly, G is said to contain a *vertical overlay* when there are edges $\{e, f\}, \{g, h\} \in E$ such that

$$C \vdash \{x_e = x_f = x_g = x_h, y_e < y_h, y_g < y_f\},$$

in which case we write $(e, f) \upuparrows (g, h)$.

NB: While edges are written in undirected notation, the overlay notation *is ordered*. For example, $(a, b) \rightrightarrows (c, d)$ is different from $(b, a) \rightrightarrows (c, d)$.

Since the horizontal and vertical cases of the **ECT** are entirely similar, we prove only the horizontal case. We begin with a lemma.

Lemma: If an edge-constrained graph $G = (V, E, C)$ contains a horizontal overlay, then there exist three nodes $u, v, w \in V$ such that $\{u, v\}, \{u, w\} \in E$ and either $(u, v) \rightrightarrows (u, w)$ or $(v, u) \rightrightarrows (w, u)$.

Proof: By the definition of horizontal overlay there are edges $\{a, b\}, \{c, d\} \in E$ such that $C \vdash (a, b) \rightrightarrows (c, d)$. Let $H = (V, F)$ be the graph in which $\{e, f\} \in F$ if and only if

$\{e, f\} \in E$ and $C \vdash y_e = y_f$. Let K be the connected component of a in H. Since G is edge-constrained, we have $a, b, c, d \in K$. Note that all nodes in K share one and the same y-coordinate. We will find $u, v, w \in K$ satisfying the statement of the lemma.

To begin with, let deg_H denote degree in H, and suppose that there is any $u \in K$ with $deg_H(u) \geq 3$. Then by the pigeonhole principle u must either have two neighbours $v, w \in K$ on its left, or two on its right, and in either case we are done.

Suppose then that all $u \in K$ have $1 \leq deg_H(u) \leq 2$. Then K forms either a cycle or a chain. Consider first the case in which K forms a cycle, that is, every $u \in K$ has $deg_H(u) = 2$. Then if $u, v, w \in K$ could not be found to satisfy the lemma, then for all $v \in K$ we would have $x_u < x_v < x_w$, where u and w are the two neighbours of v. In this case we would have a cycle of less-than relations, making $x_u < x_u$ for each $u \in K$, which is impossible.

This leaves only the case in which K forms a chain. Again, if $u, v, w \in K$ could not be found to satisfy the lemma, then each $v \in K$ with $deg_H(v) = 2$ would have one of its neighbours on each side of it, so that K would contain no overlay at all, contrary to assumption. This proves the lemma.

Finally we restate the **ECT** for the case of horizontal overlays in terms of the definitions of this section, and prove it.

Edge Coincidence Test: Let $G = (V, E, C)$ be an edge-constrained graph with $\{u, v\} \in E$, having no constraints relating u and v, and containing no horizontal overlays. Let $S = \mathsf{SA}(u, v, \mathbb{E})$. Then $C \cup \{S\}$ entails a horizontal overlay if and only if there exists a node $w \in V$ such that $C \vdash y_w = y_u$ or $C \vdash y_w = y_v$, and satisfying one of the following two sets of conditions:

1. (a) $\{u, w\} \in E$, and
   (b) $C \vdash x_u < x_w$ or $C \vdash x_v < x_w$, or
2. (a) $\{w, v\} \in E$, and
   (b) $C \vdash x_w < x_u$ or $C \vdash x_w < x_v$.

Proof: It is clear that if a node w satisfying the stated conditions exists, then a horizontal overlay will be created when S is applied. Conversely, we now suppose that a horizontal overlay is created when S is applied, and prove that such a node w must exist.

Let $C' = C \cup \{S\}$. By the Lemma there exist three nodes $a, b, c \in V$ with $\{a, b\}, \{a, c\} \in E$ such that $C' \vdash (a, b) \rightrightarrows (a, c)$ or $C' \vdash (b, a) \rightrightarrows (c, a)$. But since neither of these overlays is entailed by C, we can conclude that one of the edges $\{a, b\}, \{a, c\}$ has to be $\{u, v\}$. We assume (renaming if necessary) that $\{a, b\} = \{u, v\}$, and show that taking $w = c$ satisfies the conditions of the theorem. Specifically, we will handle the case in which $a = u$. The case in which $a = v$ is similar.

In this case $C' \vdash (v, u) \rightrightarrows (c, u)$ cannot occur, since this would involve the entailment $C' \vdash x_v < x_u$, whereas we assumed that C states no relation on nodes u and v, while the only order relation entailed by S is $x_u < x_v$. Therefore we must have $C' \vdash (u, v) \rightrightarrows (u, c)$, which says that

$$C' \vdash \{y_u = y_v = y_c, x_u < x_v, x_u < x_c\}.$$

Since $y_u = y_v$ is the only equation entailed by S, we conclude that $C \vdash y_c = y_u$ or $C \vdash y_c = y_v$. By assumption, $\{u, c\} \in E$. And again, since the only inequality entailed by S relates x_u and x_v, it must be that $C \vdash x_u < x_c$ or $C \vdash x_v < x_c$. This completes the proof.

Appendix C

Orthowontist Text Responses

At the end of the online Orthowontist experiment (Chapter 4) a final page presented participants with a text field and the instructions, **"Please describe in your own words what your goals were when you were trying to improve the layout of a diagram."** The responses of the seventeen participants are presented below. One participant wrote an itemised list, and that is reproduced faithfully here.

1. Show clear relationships between objects and try not to overlap lines.
2. I was trying to find the starting point in order to figure out the relationships between the nodes so that I could sort and simplify the diagram.
3. Reducing the amount of curvy lines and "noise". Untangling unnecessary intersections. Making it as compact and clean as possible as charts are meant to be "at a glance" to assist with understanding a process, rather than being "attractive" at the expense of complicating understanding. You would also consider boxes and whether or not they were absolutely necessary, much like writing, especially business use the aim is to be clear and concise.
4. Edges had to be rounded, boxes centered when connected, and an equal distance between the different levels of importance. If an idea box lead to another, it had to be above and if it was equal to then to the side. The final image had to be balanced, not allowing the eye to correct balance while trying to figure out what was connected to what
5. No overlapping lines, line length shortening, layout on a grid
6. Consistency, straight joining lines with no corners where possible, keeping the shapes along the same logical flow straight
7. I am trying to
   - connect one edge from one side of Node as far as possible.
   - layout the less connect node on outside in a diagram
8. To try to untangle the lines, which were often criss-crossed and difficult to 'read', by moving the boxes into a more logical order.
9. Trying to increase left-to-right or clockwise reading in threads or loops, because that's the way Western readers are taught to read.

   Balance situations where two or more options came from one node, i.e. connect each new node on the same plane (not one above, one under, one to the right of, but all to the right on three different levels)

A small amount of aesthetic change, because I think curved corners are nicer than sharp ones

10. Simple, straight lines to indicate direct relationships between two items. Also, aligning nodes so they match up (with an imaginary ruler)

11. Mostly I was trying to make the links as clear as possible, and then achieve some kind of overall shape. Towards the end I began to focus more on the aesthetics rather than just the practicalities.

12. symmetry.

13. To keep the diagrams as simple as possible. Fewer over laps with lines and clear connections between boxes.

14. Get rid of as many bumps and overlaps as possible.

    Showing a more direct path.

15. Clean

16. Uncrossed lines, hierarchy of levels, right angles

17. trying to keep it simple and easy to read by trying to keep the lines straight whenever possible

Appendix D

Orthowontist Metrics

Among the hypotheses to be tested by the Orthowontist experiment (Chapter 4) was the idea that certain qualities or aspects of a layout would bear on how highly that layout would be ranked in order of preference. This included the classic aesthetic concerns of number of crossings and bends in connector routes, standard deviation in the length of connector segments, compactness of the drawing, and symmetry. We also investigated the stress of the final layout, its "gridiness" (a measure of how grid-like the layout was), and whether the trees of the network were placed in the external face. This appendix provides the formal details of these metrics.

D.1 Stress

Recall that the stress of a drawing of a graph $G = (V, E)$ is given by

$$\sum_{u,v \in V} \left(\frac{|u - v|}{d_{uv}} - \ell_0 \right)^2$$

where $|u - v|$ gives the geometric distance between the drawings of nodes u and v, while d_{uv} is the graph-theoretic distance between them, and ℓ_0 is the *ideal edge length* for the drawing. Therefore before we can compute the stress of a layout we must define the parameter ℓ_0 on which it depends.

In the case of human designed layouts there is a natural way to define ℓ_0: take the *average neighbour distance* (average distance between neighbouring nodes, i.e. nodes connected by an edge) in the final layout. To be clear, this means the straight-line distance between the centres of the nodes, not the lengths of the connector routes between them. See Figure D.1.

The rationale is that when a study participant clicked the "Done" button after each layout task in the Orthowontist study they were effectively saying, "I think this layout looks good," which in particular implies, "Neighbouring nodes are now about as far from each other as they ought to be." In other words, by showing us what a good layout

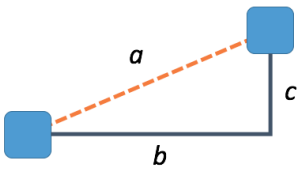


Figure D.1: The "neighbour distance" between these two nodes is the straight-line distance a between their centres, not the sum $b + c$ of the lengths of the segments making up the connector route.

Figure D.2: These two layouts of a three-node graph have the same average neighbour separation; however, the layout on the left has zero stress, while the layout on the right has higher stress.

looked like, participants were also showing us what they thought was an ideal neighbour separation. Thus, if for each node $v \in V$ the position of v in the final layout is given by $(v.x, v.y)$, then we define the ideal edge length ℓ_0 for stress computations as

$$\ell_0 = \frac{1}{|E|} \sum_{(u,v)\in E} \sqrt{(v.x - u.x)^2 + (v.y - u.y)^2}. \tag{D.1}$$

The reader may wonder whether it is fair to define the ideal edge length in this way, and whether this definition automatically means that every layout will have low stress. The answer is no. To see why not, we may begin by considering a simpler version of the stress function that sums stress terms only for neighbouring nodes (i.e. the *neighbour stress* function that was used in Chapter 5):

$$\sum_{(u,v)\in E} \left(|u - v| - \ell_0\right)^2. \tag{D.2}$$

We observe that this expression is equal to the *variance* in the neighbour separations in the layout. Meanwhile the actual stress function includes additional non-negative terms on top of the neighbour stress. Therefore with our definition of the ideal edge length ℓ_0 in (D.1) the stress of the layout is bounded below by the variance in the neighbour separations, and is by no means automatically low. See Figure D.2.

D.2 Symmetry

Various metrics for symmetry in node-link diagrams have been proposed. These metrics always incorporate the idea of pairing nodes and/or connector routes over axes of reflection, but the devil lies in the details and invariably some ad hoc choices are made. For example Purchase et al. used one definition in an initial study [PCJ96] but then after reconsideration formulated a different metric [PL96] for a later study [Pur97].

The revised symmetry metric of Purchase et al. seems good intuitively, and was shown to have some correlation with the usability of node-link diagrams [Pur97]. However, it did not take node size or ports into account. For our study we formulate a similar symmetry metric, but one that accounts for node sizes and ports.

To begin with, we want to consider axes of reflection over which node boxes and connector routes pair off symmetrically. The percentage of nodes and connectors participating in the single largest such symmetry may be taken as the symmetry value for the layout. However, this rough idea needs to be refined in several ways:

1. Since we are concerned with hand-made layouts we must accept approximate symmetry, so when nodes or edges "pair off" this will have to mean that when one is reflected across the axis of symmetry it lands "close enough" to its partner.

2. Since we are interested in grid-like orthogonal layouts it makes sense to consider only horizontal and vertical axes of symmetry. Furthermore we may demand that each potential axis pass either through the centre of one or more nodes, or through the midpoint between the centres of two aligned nodes. See Figure D.3. Restricting

to this finite set of possible axes also helps to make the problem computationally tractable.

3. We only want to consider non-trivial symmetries, so we will require that at least three nodes participate. Otherwise any pair of aligned nodes would constitute a symmetry, and we believe that this is not worth counting.

4. For each axis we will consider how many nodes and edges pair off symmetrically about it. However, we are looking for symmetries that "jump out" visually, so rather than considering random collections of nodes and edges we will consider only *subgraphs*, i.e. collections of nodes plus those edges having both their endpoints within that collection.

5. We want a normalised measure, so if n and m are the total numbers of nodes and edges in the graph, respectively, then for each axis A and subgraph H that has a reflection symmetry over A we will count the nodes and edges in H and divide by $n + m$. This assigns a symmetry value for H over A.

6. Again, since we want symmetries to pop out visually, we will exclude from consideration any subgraph H whose bounding box intersects any nodes not belonging to H. Such nodes create visual clutter that obscures the symmetry. A subgraph H whose bounding box does not intersect any nodes not belonging to H will be called "closed" with respect to the layout.

7. To each axis A we assign the largest normalised symmetry value over all closed subgraphs of at least three nodes. Then to the layout itself we assign the largest such value over all axes A.

8. But in fact we will again refine this idea to allow for "sub-symmetries," the idea being that if you can split the graph into two or more parts by deleting one or more edges, and find a different axis of symmetry for each part, then the total symmetry of the graph can be computed by combining the symmetries of these parts, minus a penalty for the deleted edges. See Figure D.4.

These ideas are easy enough to understand, but it takes a fair bit of formalism to make them precise. Readers satisfied with this rough description may feel free to skip the remainder of this section.

In the following definitions, we assume a graph $G = (V, E)$, along with width and height functions $w : V \to \mathbb{R}_{>0}$ and $h : V \to \mathbb{R}_{>0}$ for the nodes of G.

We also assume a *layout* $L = (c, r)$ of G, consisting of two maps, $c : V \to \mathbb{R}^2$ and $r : E \to (\mathbb{R}^2)^{<\omega}$, where $(\mathbb{R}^2)^{<\omega}$ denotes the set of all finite sequences of points in $\mathbb{R}^2$. The idea is that $c(v)$ represents the centre of node v, while $r(e)$ defines a piecewise-linear curve to represent edge e. The edge routes must connect the node centres; formally, for each edge $e = (u, v)$, if $r(e) = \langle p_0, p_1, \ldots, p_n \rangle$ then we must have $p_0 = c(u)$ and $p_n = c(v)$.

The functions c, r of the layout L give us the raw data to define the boxes and curves that make the graphical representations of the nodes and edges of the graph, but we need additional definitions to refer to these geometric figures, and we set about that now.

For each node $v \in V$ we define the *box* for v, denoted $\beta(v)$, to be the rectangle in the plane with centre at $c(v)$ and having width $w(v)$ and height $h(v)$.

Given two points $p, q \in \mathbb{R}^2$, we define $\sigma(p, q)$ to be the line segment connecting them.

For each edge $e \in E$ we define the *connector* for e, denoted $\kappa(e)$ as the union of all the segments $\sigma(p_{i-1}, p_i)$, $1 \leq i \leq n$, where $r(e) = \langle p_0, p_1, \ldots, p_n \rangle$.

It will be useful to be able to denote the geometric figure for both nodes and edges uniformly; therefore we define

$$\varphi : V \cup E \to \wp(\mathbb{R}^2)$$

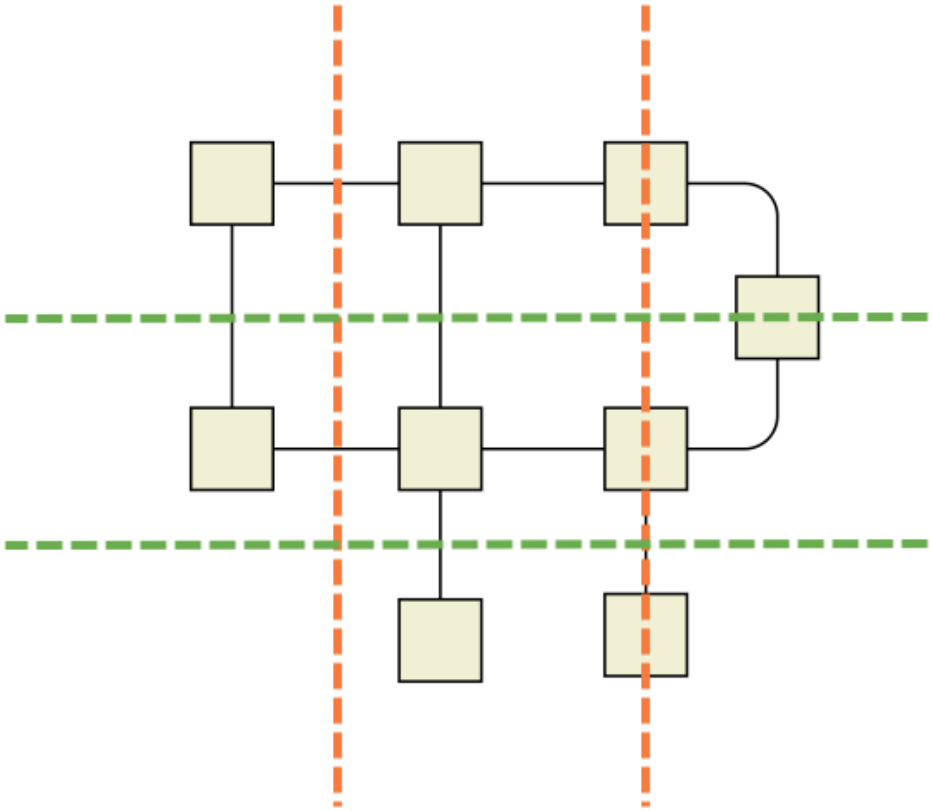


Figure D.3: For grid-like symmetries we consider only axes of symmetry that are horizontal or vertical, and that pass either through the centre of a node or through the midpoint between two adjacent aligned nodes. Four of this layout's eleven possible axes are shown.

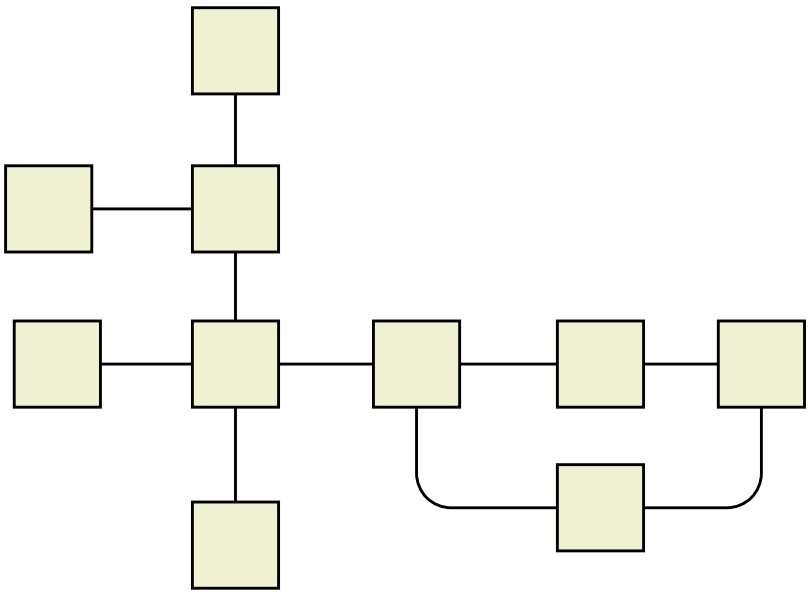


Figure D.4: As in this example, layouts may exhibit a lot of symmetry and yet lack a single axis representing all of it. In a case like this we believe it is important to consider breaking the layout into sublayouts, and examining the symmetry of each part separately. For example if the central edge in this figure is deleted then each of the two resulting components has perfect symmetry.

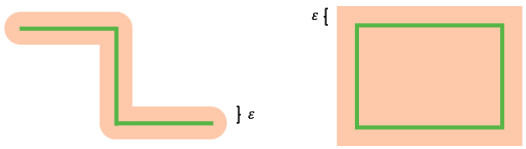


Figure D.5: ε-neighbourhoods around a connector route (left) and node box (right)

to be the unique map that equals β when restricted to V and equals κ when restricted to E. The codomain is the powerset of $\mathbb{R}^2$ since φ maps each node and edge to a subset of the plane.

We need to be able to talk about reflections. Therefore if F is any geometric figure in the plane, literally a subset $F \subseteq \mathbb{R}^2$, and if A is any axis, i.e. line, in the plane, then the reflection of F across A will be denoted $R_A(F)$.

Since we are interested in approximate symmetries we need to be able to talk about neighbourhoods around geometric figures. Therefore if $F \subseteq \mathbb{R}^2$ and $\varepsilon > 0 \in \mathbb{R}$ are given, we will denote by $N_\varepsilon(F)$ the *ε-neighbourhood around F*, i.e. the set of all points in $\mathbb{R}^2$ that lie within a distance of ε of any point in F. See Figure D.5.

Next we formalise the notion of a "pairing off" of geometric figures, up to a tolerance ε, when reflected over an axis A. If $H = (W, F)$ is any subgraph of $G = (V, E)$ (so $W \subseteq V$ and $F \subseteq W \times W$) then by the notation $\rho : H \to H$ we will mean a map $\rho : W \cup F \to W \cup F$ satisfying the condition that $\rho(W) \subseteq W$ and $\rho(F) \subseteq F$. If A is any axis and if a tolerance $\varepsilon > 0$ is given then a map $\rho : H \to H$ is said to be an *A, ε-reflection map* if it satisfies all the following conditions:

i. ρ is one-one and onto;

ii. ρ is its own inverse; i.e. $\rho(\rho(x)) = x$ for all $x \in H$; and

iii. for all $x \in H$,
$$R_A(\varphi(x)) \subseteq N_\varepsilon(\varphi(\rho(x))).$$

In words, the last condition states that, for each element x of H, be it node or edge, the reflection over axis A of the geometric figure representing x is contained in the ε-neighbourhood of the geometric figure representing the partner $\rho(x)$ in the intended symmetric pairing.

A subgraph H of G will be called *ε-symmetric over A* if there exists an A, ε-reflection map $\rho : H \to H$.

If $H = (W, F)$ is a subgraph of G its *bounding box* $\beta(H) \subseteq \mathbb{R}^2$ is defined to be the smallest rectangle in the plane containing $\beta(w)$ for all $w \in W$, and $\kappa(f)$ for all $f \in F$.

A subgraph $H = (W, F)$ of $G = (V, E)$ is said to be *closed* with respect to the layout L if for all $v \in V$, $\beta(v) \cap \beta(H) \neq \varnothing$ implies $v \in W$. In other words, H is closed when it includes every node whose box intersects the bounding box of H.

For a given axis A and tolerance ε, we can now define the set of all subgraphs of G that can contribute to the degree of symmetry we attribute to axis A. Namely, we define
$$\mathcal{S}(G, L, A, \varepsilon)$$
to be the set of all subgraphs H of G satisfying all the following conditions:

i. H contains at least three nodes;

ii. H is closed with respect to L;

iii. H is ε-symmetric over A.

For any subgraph $H = (W, F)$ of G we define the *mass* of H to be $m(H) = |W| + |F|$, i.e. the total number of nodes and edges in the subgraph.

For any axis A and tolerance ε we define the ε-symmetry of A to be

$$\operatorname{symm}_\varepsilon(A) = \max_{H \in \mathcal{S}(G,L,A,\varepsilon)} \frac{m(H)}{m(G)}.$$

Let $\operatorname{grax}(L)$ denote the set of all "grid-like axes" A for the layout L, i.e. the set of all axes that are either horizontal or vertical, and either pass through $c(v)$ for some $v \in V$ or through $(c(u) + c(v))/2$ for some nodes $u, v \in V$ that have equal x- or y-coordinate (i.e. are aligned in one dimension).

Finally we can define the ε-symmetry of the layout L:

$$\operatorname{symm}_\varepsilon(L) = \max_{A \in \operatorname{grax}(L)} \operatorname{symm}_\varepsilon(A).$$

Furthermore we propose that there is a principled tolerance ε to be used in these measurements, namely,

$$\varepsilon_0 = \frac{1}{4\,|V|} \sum_{v \in V} \left(w(v) + h(v)\right). \tag{D.3}$$

i.e. half the average extension of any node box in either dimension. When we speak of *symmetry* unqualified by any tolerance ε, we will always mean ε_0-symmetry. Thus

$$\operatorname{symm}(L) = \operatorname{symm}_{\varepsilon_0}(L).$$

We have just one last part of the definition of symmetry to work out, and that is the idea of splitting the graph G by deleting edges, determining the symmetry of the parts, and putting these together to get a measurement of the symmetry of the full graph G. This idea is motivated by examples like the layout in Figure D.4 (the top human layout for Graph 8 in the study), in which if the central edge is deleted, each of the two resulting subgraphs has complete symmetry. Recall that an edge is called a *bridge* if its deletion increases the number of connected components in the graph.

Furthermore, motivated by the same example, we believe it is important that we consider the deletion not just of any edge, but only of those edges routed in such a way that their deletion seems visually obvious. We are trying to detect symmetries that jump out visually, and in Figure D.4 we attribute to its isolation from other objects the central edge's appearance of being "prime for deletion".

We formalise this notion as follows. Given a graph $G = (V, E)$ and a layout $L = (c, r)$ of G, an edge $e \in E$ is called an *exposed bridge* if (a) it is a bridge, i.e. is such that its deletion increases the number of connected components in the graph, and (b) it is visually "exposed" in the sense that if $r(e) = \langle p_0, p_1, \ldots, p_n \rangle$ then at least one of the segments $\sigma_i = \sigma(p_{i-1}, p_i)$ is such that the perpendicular line passing through the midpoint of σ_i does not intersect any other node or segment in the layout.

Our suspicion is that an exposed bridge will have a far greater tendency to induce visual perceptual chunking than an unexposed one, and this seems consistent with gestalt principles [Koh50], although to verify this specific claim would require further user studies.

For a graph $G = (V, E)$, let $\operatorname{exbr}(G) \subseteq E$ denote the set of exposed bridges of G. For any set of edges $F \subseteq E$, we denote by $G \setminus F$ the subgraph of G resulting from deletion of all edges in F. We write $\mathscr{C}(G)$ for the set of connected components of G. When L is a layout of G and H is a subgraph of G we denote by $L|_H$ the layout L restricted to the

subgraph H. Then we can define the *fractional symmetry* of a layout L of G as

$$\text{fsymm}(L) = \max_{F \subseteq \text{exbr}(G)} \frac{\sum_{H \in \mathscr{C}(G \setminus F)} m(H)\, \text{symm}(L|_H)}{m(G)}.$$

Note that since in particular we have $\varnothing \subseteq \text{exbr}(G)$ it follows that $\text{fsymm}(L) \geq \text{symm}(L)$. Therefore, since our goal is always to attribute the maximum possible symmetry to any layout, we will only be concerned with fractional symmetry.

D.3 Compactness

We computed compactness as the ratio of the area occupied by the nodes to the total area of the graph, the latter being given by the bounding box of all nodes and bend points.

D.4 "Gridiness"

As in Chapter 3, we think of a layout as being "grid-like" when many of the nodes participate in alignments. Therefore we measure the grid-like quality of a given layout—its "gridiness"—by reporting the proportion of nodes that participate in at least one alignment of three or more nodes. We require at least three nodes in order to discount trivial cases.

Since we are concerned with hand-made layouts we must accept approximate alignments, up to some error tolerance ε. For this we used one quarter of the average node size, or half the tolerance defined in (D.3).

Vita

Publications arising from this thesis include:

Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow (2016). "HOLA: Human-like orthogonal network layout." IEEE transactions on visualization and computer graphics 22, no. 1: 349-358.

Ulf Rüegg, Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow (2014). "Stress-minimizing orthogonal layout of data flow diagrams with ports." In International Symposium on Graph Drawing, pp. 319-330. Springer Berlin Heidelberg.

Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow (2013). "Incremental grid-like layout using soft and hard constraints." In International Symposium on Graph Drawing, pp. 448-459. Springer International Publishing.

Permanent Address: Caulfield School of Information Technology
Monash University
Australia

This thesis was typeset with LaTeX 2$_\varepsilon$[1] by the author.

[1] LaTeX 2$_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Glenn Maughan and modified by Dean Thompson and David Squire of Monash University.

References

[AMA07] Daniel Archambault, Tamara Munzner, and David Auber. TopoLayout: Multilevel graph layout by topological features. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):305–317, 2007.

[AvHK06] James Abello, Frank van Ham, and Neeraj Krishnan. ASK-GraphView: A large scale graph visualization system. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):669–676, 2006.

[BC01] Ralf Brockenauer and Sabine Cornelsen. Drawing clusters and hierarchies. In *Drawing graphs*, pages 193–227. Springer, 2001.

[BEKW02] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, and Dorothea Wagner. Sketch-driven orthogonal graph drawing. In *Graph Drawing*, pages 1–11. Springer, 2002.

[Ber83] Jacques Bertin. *Semiology of graphics: diagrams, networks, maps*. University of Wisconsin press, 1983.

[BGHM07] Aaron Barsky, Jennifer L Gardy, Robert EW Hancock, and Tamara Munzner. Cerebral: a cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics*, 23(8):1040–1042, 2007.

[BLW76] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1976.

[BNT86] Carlo Batini, Enrico Nardelli, and Roberto Tamassia. A layout algorithm for data flow diagrams. *Software Engineering, IEEE Transactions on*, (4):538–546, 1986.

[BO79] Jon L Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *Computers, IEEE Transactions on*, 100(9):643–647, 1979.

[Coh97] Jonathan D Cohen. Drawing graphs to convey proximity: an incremental arrangement method. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 4(3):197–229, 1997.

[Com] First Annual SBGN Layout Competition. http://www.sbgn.org/Competition/Competition_2010/.

[DBETT99] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall, 1999.

[DBGL95] Giuseppe Di Battista, Ashim Garg, and Giuseppe Liotta. An experimental comparison of three graph drawing algorithms. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 306–315. ACM, 1995.

[DFM93] Ed Dengler, Mark Friedell, and Joe Marks. Constraint-driven diagram layout. In *Visual Languages, 1993., Proceedings 1993 IEEE Symposium on*, pages 330–335. IEEE, 1993.

[DH96] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996.

[DK05] Tim Dwyer and Yehuda Koren. Dig-cola: directed graph layout through constrained energy minimization. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 65–72. IEEE, 2005.

[DKM05] Tim Dwyer, Yehuda Koren, and Kim Marriott. Stress majorization with orthogonal ordering constraints. In *International Symposium on Graph Drawing*, pages 141–152. Springer, 2005.

[DKM06] Tim Dwyer, Yehuda Koren, and Kim Marriott. IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):821–828, 2006.

[DLF+09] Tim Dwyer, Bongshin Lee, Danyel Fisher, Kori Inkpen Quinn, Petra Isenberg, George Robertson, and Chris North. A comparison of user-generated and automatic graph layouts. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):961–968, 2009.

[DLS84] Jan De Leeuw and Ineke Stoop. Upper bounds for kruskal's stress. *Psychometrika*, 49(3):391–402, 1984.

[DMS06] Tim Dwyer, Kim Marriott, and Peter J Stuckey. Fast node overlap removal. In *Graph Drawing*, pages 153–164. Springer, 2006.

[DMW09a] Tim Dwyer, Kim Marriott, and Michael Wybrow. Dunnart: A constraint-based network diagram authoring tool. In *Graph Drawing*, pages 420–431. Springer, 2009.

[DMW09b] Tim Dwyer, Kim Marriott, and Michael Wybrow. Topology preserving constrained graph layout. In *Graph Drawing*, pages 230–241. Springer Berlin Heidelberg, 2009.

[Ead84] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42(11):149–160, 1984.

[EdMN95] P. Eades and C.F.X. de Mendonça N. Vertex splitting and tension-free layout. In *Graph Drawing*, pages 202–211. Springer, 1995.

[EH12] Peter Eades and Seok-Hee Hong. How to draw a graph, revisited. In *Expanding the Frontiers of Visual Analytics and Visualization*, pages 111–126. Springer, 2012.

[Eul41] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[Eul58] Leonhard Euler. Elementa doctrinae solidorum. *Novi Commentarii academiae scientiarum Petropolitanae*, 4:109–140, 1758.

[FCW67] Clifford J Fisk, David L Caskey, and Leslie E West. ACCEL: Automated circuit card etching layout. *Proceedings of the IEEE*, 55(11):1971–1982, 1967.

[FI65] Clifford J Fisk and D.D. Isett. ACCEL automated circuit card etching layout. In *Proceedings of the SHARE design automation project*, pages 9–1. ACM, 1965.

[FK95] Ulrich Fößmeier and Michael Kaufmann. Drawing high degree graphs with low bend numbers. In *International Symposium on Graph Drawing*, pages 254–266. Springer, 1995.

[FK97] Ulrich Fößmeier and Michael Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In *International Symposium on Graph Drawing*, pages 134–145. Springer, 1997.

[Fle13] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[FR95] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[Gal16] Joseph Gallian. *Contemporary abstract algebra*. Cengage Learning, 2016.

[GKN04] Emden R Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, pages 239–250. Springer, 2004.

[GM03] Carsten Gutwenger and Petra Mutzel. An experimental study of crossing minimization heuristics. In *International Symposium on Graph Drawing*, pages 13–24. Springer, 2003.

[Har] Frank Harary. *Graph theory. 1969*. Addison-Wesley, Reading, MA.

[HBF08] Nathalie Henry, Anastasia Bezerianos, and Jean-Daniel Fekete. Improving the readability of clustered social networks using node duplication. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1317–1324, 2008.

[HF06] Nathalie Henry and Jean-Daniel Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE transactions on visualization and computer graphics*, 12(5):677–684, 2006.

[HFM07] Nathalie Henry, Jean-Daniel Fekete, and Michael J McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE transactions on visualization and computer graphics*, 13(6):1302–1309, 2007.

[HHE07] Weidong Huang, Seok-Hee Hong, and Peter Eades. Effects of sociogram drawing conventions and edge crossings in social network visualization. *J. Graph Algorithms Appl.*, 11(2):397–429, 2007.

[Him95] Michael Himsolt. Comparing and evaluating layout algorithms within GraphEd. *Journal of Visual Languages and Computing*, 6(3):255–273, 1995.

[HM96] Weiqing He and Kim Marriott. Constrained graph layout. In *International Symposium on Graph Drawing*, pages 217–232. Springer, 1996.

[Hol06] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.

[Hua07] Weidong Huang. Using eye tracking to investigate graph layout effects. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on*, pages 97–100. IEEE, 2007.

[KA06] Kurt W Kohn and Mirit I Aladjem. Circuit diagrams for biological networks. *Molecular systems biology*, 2(1), 2006.

[Kam89] Tomihisa Kamada. *Visualizing abstract objects and relations*, volume 5. World Scientific, 1989.

[KDMW13] Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow. Incremental grid-like layout using soft and hard constraints. In *Graph Drawing*, pages 448–459. Springer, 2013.

[KDMW16] Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow. HOLA: Human-like orthogonal network layout. *Visualization and Computer Graphics, IEEE Transactions on*, 22(1):349–358, 2016.

[KFMO05] Hiroaki Kitano, Akira Funahashi, Yukiko Matsuoka, and Kanae Oda. Using process diagrams for the graphical representation of biological networks. *Nature biotechnology*, 23(8):961–966, 2005.

[KH03] Yehuda Koren and David Harel. Axis-by-axis stress minimization. In *International Symposium on Graph Drawing*, pages 450–459. Springer, 2003.

[KK89] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[KMS94] Corey Kosak, Joe Marks, and Stuart Shieber. Automating the layout of network diagrams with specified visual organization. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(3):440–454, 1994.

[KNJ+07] Kaname Kojima, Masao Nagasaki, Euna Jeong, Mitsuru Kato, and Satoru Miyano. An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC Bioinformatics*, 8(1):76, 2007.

[Knu63] Donald E. Knuth. Computer drawn flowcharts. *Communications of the ACM*, 6, 1963.

[Koh50] Wolfgang Kohler. *Physical Gestalten*, pages 17–54. The Humanities Press, 1950.

[Kru64] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[KS80] Joseph B Kruskal and Judith B Seery. Designing network diagrams. In *Proc. First General Conf. on Social Graphics*, pages 22–50, 1980.

[LHM+09] N. Le Novère, M. Hucka, H. Mi, S. Moodie, F. Schreiber, A. Sorokin, E. Demir, K. Wegner, M. Aladjem, S. M. Wimalaratne, F. T. Bergman, R. Gauges, P. Ghazal, K. Hideya, L. Li, Y. Matsuoka, A. Villéger, S. E. Boyd, L. Calzone, M. Courtot, U. Dogrusoz, T. Freeman, A. Funahashi, S. Ghosh, A. Jouraku, S. Kim, F. Kolpakov, A. Luna, S. Sahle, E. Schmidt,

S. Watterson, G. Wu, I. Goryanin, D. B. Kell, C. Sander, H. Sauro, J. L. Snoep, K. Kohn, and H. Kitano. The Systems Biology Graphical Notation. *Nature Biotechnology*, 27:735–741, 2009.

[LK05] Weijiang Li and Hiroyuki Kurata. A grid layout algorithm for automatic drawing of biochemical networks. *Bioinformatics*, 21(9):2036–2042, 2005.

[LNW03] Edward A. Lee, Stephen Neuendorffer, and Michael J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers (JCSC)*, 12(3):231–260, 2003.

[MA88] Joseph Manning and Mikhail J. Atallah. Fast detection and display of symmetry in trees. *Congr. Numer.*, 64:159–169, 1988.

[Mar91a] Joe Marks. A formal specification scheme for network diagrams that facilitates automated design. *Journal of Visual Languages & Computing*, 2(4):395–414, 1991.

[Mar91b] Joseph William Marks. Automating the design of network diagrams. 1991.

[Mar92] George Markowsky. Misconceptions about the golden ratio. *The College Mathematics Journal*, 23(1):2–19, 1992.

[MBK97] Cathleen McGrath, Jim Blythe, and David Krackhardt. The effect of spatial arrangement on judgments and errors in interpreting graphs. *Social Networks*, 19:223–242, 1997.

[Met] MetaCrop. http://metacrop.ipk-gatersleben.de.

[MPWG12] Kim Marriott, Helen Purchase, Michael Wybrow, and Cagatay Goncu. Memorability of visual features in network diagrams. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2477–2485, 2012.

[MT01] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*, volume 10. JHU Press, 2001.

[Mun00] James R. Munkres. *Topology*. Prentice Hall, Inc, 2000.

[Mun14] Tamara Munzner. *Visualization Analysis and Design*. CRC Press, 2014.

[NEH13] Quan Nguyen, Peter Eades, and Seok-Hee Hong. On the faithfulness of graph visualizations. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pages 209–216. IEEE, 2013.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999.

[NW11a] Martin Nöllenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *Visualization and Computer Graphics, IEEE Transactions on*, 17(5):626–641, 2011.

[NW11b] Martin Nöllenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *Visualization and Computer Graphics, IEEE Transactions on*, 17(5):626–641, 2011.

[PAC02] Helen C Purchase, Jo-Anne Allder, and David A Carrington. Graph layout aesthetics in UML diagrams: user preferences. *J. Graph Algorithms Appl.*, 6(3):255–279, 2002.

[PCA02] Helen C. Purchase, David Carrington, and Jo-Anne Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

[PCJ96] Helen C Purchase, Robert F Cohen, and Murray James. Validating graph drawing aesthetics. In *Graph Drawing*, pages 435–446. Springer, 1996.

[PCJ97] Helen C. Purchase, Robert F. Cohen, and Murray I James. An experimental study of the basis for graph drawing algorithms. *Journal of Experimental Algorithmics (JEA)*, 2:4, 1997.

[PL96] Helen Purchase and David Leonard. *Graph drawing aesthetic metrics*. Key Centre for Software Technology, Department of Computer Science, University of Queensland, 1996.

[PPP12] Helen C Purchase, Christopher Pilcher, and Beryl Plimmer. Graph drawing aesthetics–created by users, not algorithms. *Visualization and Computer Graphics, IEEE Transactions on*, 18(1):81–92, 2012.

[Pur97] Helen Purchase. Which aesthetic has the greatest effect on human understanding? In *Graph Drawing*, pages 248–261. Springer, 1997.

[Pur98] Helen C Purchase. Performance of layout algorithms: Comprehension, not computation. *Journal of Visual Languages & Computing*, 9(6):647–657, 1998.

[RKD+14] Ulf Rüegg, Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow. Stress-minimizing orthogonal layout of data flow diagrams with ports. In *Graph Drawing*, pages 319–330. Springer, 2014.

[RMS97] Kathy Ryall, Joe Marks, and Stuart Shieber. An interactive constraint-based system for drawing graphs. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 97–104. ACM, 1997.

[Sam69] John W Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 18(5):401–409, 1969.

[SCC+12] Falk Schreiber, Christian Colmsee, Tobias Czauderna, Eva Grafahrend-Belau, Anja Hartmann, Astrid Junker, Björn H. Junker, Matthias Klapperstück, Uwe Scholz, and Stephan Weise. Metacrop 2.0: managing and exploring information about crop plant metabolism. *Nucleic Acids Research*, 40, 2012.

[SDMW09] Falk Schreiber, Tim Dwyer, Kim Marriott, and Michael Wybrow. A generic algorithm for layout of biological networks. *BMC bioinformatics*, 10(1):1, 2009.

[Sei83] Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[SFVHM09] Miro Spönemann, Hauke Fuhrmann, Reinhard Von Hanxleden, and Petra Mutzel. Port constraints in hierarchical layout of data flow diagrams. In *International Symposium on Graph Drawing*, pages 135–146. Springer, 2009.

[SM95] Kozo Sugiyama and Kazuo Misue. Graph drawing by the magnetic spring model. *Journal of Visual Languages and Computing*, 6(3):217–231, 1995.

[SRMOW11] Jonathan Stott, Peter Rodgers, Juan Carlos Martinez-Ovando, and Stephen G Walker. Automatic metro map layout using multicriteria optimization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(1):101–114, 2011.

[SSvH14] Christoph Daniel Schulze, Miro Spönemann, and Reinhard von Hanxleden. Drawing layered graphs with port constraints. *Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout*, 25(2):89–106, 2014.

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.

[Syl78] James Joseph Sylvester. Chemistry and algebra. *Nature*, 17:284, 1878.

[Tam87] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[TDBB88] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, 1988.

[Tor52] Warren S Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[Tut60] William Thomas Tutte. Convex representations of graphs. *Proceedings London Mathematical Society*, 10(38):304–320, 1960.

[Tut63] William Thomas Tutte. How to draw a graph. *Proc. London Math. Soc*, 13(3):743–768, 1963.

[vHR08] Frank van Ham and Bernice E. Rogowitz. Perceptual organization in user-generated graph layouts. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1333–1339, 2008.

[VW82] Christopher J Van Wyk. A high-level language for specifying pictures. *ACM Transactions on Graphics (TOG)*, 1(2):163–182, 1982.

[WC11] Yu-Shuen Wang and Ming-Te Chi. Focus+context metro maps. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2528–2535, 2011.

[WEK04] Roland Wiese, Markus Eiglsperger, and Michael Kaufmann. yFiles – visualization and automatic layout of graphs. In *Graph Drawing Software*, pages 173–191. Springer, 2004.

[WMMS08] Michael Wybrow, Kim Marriott, Linda McIver, and Peter J. Stuckey. Comparing usability of one-way and multi-way constraints for diagram editing. *Computer-Human Interaction, ACM Transactions on*, 14(4):1–38, 2008.

[WMS10] Michael Wybrow, Kim Marriott, and Peter J Stuckey. Orthogonal connector routing. In *Graph Drawing*, pages 219–231. Springer, 2010.

[WPCM02] Colin Ware, Helen Purchase, Linda Colpoys, and Matthew McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.