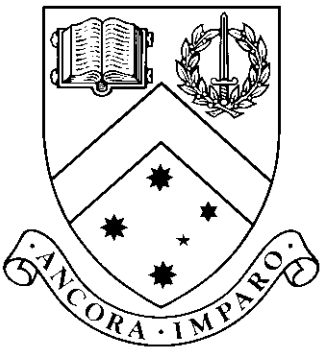
Using semi-automatic layout to improve the usability of diagramming software

by

Michael Wybrow, BCompSc (Hons)



Thesis

Submitted by Michael Wybrow

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Prof. Kim Marriott

Associate Supervisor: Prof. Peter J. Stuckey

Associate Supervisor: Dr Linda McIver

Associate Supervisor: Dr Tim Dwyer

Clayton School of Information Technology
Monash University

June, 2008

For Chelsea.

Contents

Using semi-automatic layout to improve the usability of diagramming software

Michael Wybrow, BCompSc (Hons)
Michael.Wybrow@infotech.monash.edu.au
Monash University, 2008

Supervisor: Prof. Kim Marriott
Kim.Marriott@infotech.monash.edu.au
Associate Supervisor: Prof. Peter J. Stuckey
pjs@csse.unimelb.edu.au
Associate Supervisor: Dr Linda McIver
linda.mciver@gmail.com
Associate Supervisor: Dr Tim Dwyer
Tim.Dwyer@infotech.monash.edu.au

Abstract

Diagrams are a useful way to efficiently convey abstract information. They are invaluable in fields such as software engineering, social network analysis, and bioinformatics. Layout is an important aspect of diagram creation, impacting heavily on readability. Good diagram layout is difficult because it encompasses both a user's aesthetic preferences as well as drawing conventions for particular styles of diagrams. Diagram authoring software provides some layout tools but these usually perform a once-off change. As such, the author of the diagram handles the majority of diagram layout manually.

This thesis presents persistent layout tools for user-specified placement, connector routing and automatic network layout. These tools are implemented using constraint solvers that maintain spatial relationships throughout further editing. The tools have been designed to be highly usable, with focus on their behaviour, interface and user interaction.

Constraint-based placement tools are evaluated at several stages of design via usability experiments as well as informal feedback from users. The constraint-based approach is found to be more usable than existing layout tools, offering the user valuable layout assistance without depriving them of flexibility.

A fast incremental connector routing technique is demonstrated. It maintains optimal connector routes, providing more predictable behaviour than existing approaches. We show its versatility via extensions to produce routings fulfilling various desired aesthetic criteria.

A new model for interactive network layout is presented, where the user and the system collaborate to produce better layouts. The user has a large amount of control over the layout through the use of structural styles, placement tools and stylistic constraints; the responsibility for all of which are delegated to the system to maintain.

All of the presented techniques are fully implemented in Dunnart, a truly usable constraint-based diagram editor. Dunnart features user-specifiable placement tools, incremental connector routing, and continuous network layout, all of which can be directed by the user, to help produce better diagram layouts. Dunnart provides a compelling demonstration of the possible applications arising from the research described in this thesis.

Using semi-automatic layout to improve the usability of diagramming software

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.

Michael Wybrow
June 16, 2008

Acknowledgments

I would like to express my gratitude to my excellent PhD supervisors: Prof. Kim Marriott, Prof. Peter Stuckey, Dr Linda McIver and Dr Tim Dwyer. Their enthusiasm, advice, guidance and friendship over the term of my research is highly appreciated.

A special thanks to Kim Marriott who, in addition to being an excellent supervisor, offered me a scholarship I would not otherwise have received, and ultimately enabled me to take up postgraduate study. Thanks Kim, you always said you would get me to do a PhD. I'm still not exactly sure how that happened, but I'm glad it did!

Thanks to all the present and past members of the "constraint solving" group at Monash University. I have truly appreciated the social and friendly nature of the group. I have especially enjoyed the all the lunches and dinners we've shared together. Particular thanks go to the members of the Adaptive Diagrams Research Group for the useful comments and suggestions on my research at various times.

Particular thanks go to Nathan Hurst, who regularly showed excitement at my research and led me in profitable directions. Also, for providing some incredibly interesting problems as distractions. To Peter Moulder, who was my go-to person while troubleshooting difficult technical problems. Whenever I had tried everything and was convinced I had stumbled upon a compiler bug, he would then quickly determine the source of the error—within my code! To Cameron McCormack, who frequently looked at my software prototypes and asked useful questions. To Peter Sbarski for often providing welcome distractions, often in the form of interesting or humorous web content. Finally, thanks to Tim Dwyer for being a great mentor, friend and collaborator.

Thank-you to the various people I have shared an office with during my candidature. Specifically, Dr Tony Jansen and Dr Sarah Boyd, my first officemates, who welcomed me and shared their wisdom and experience on all things PhD-related. Also to Reza Rafeh, Pritika Sanghi, Chris Mears, Dhananjay Thiruvady and Cagatay Goncu, for providing a pleasant working environment with plenty of laughter.

Thank-you to Bulia Byak, MenTaLguY and Peter Moulder for their assistance integrating `libavoid` into Inkscape. Thanks also for feedback from Inkscape's friendly developer community. I've enjoyed working as a member of the Inkscape team and look forward to hacking on it further in the future.

Although it is unlikely any of them will ever read this document, I'd like to mention the following Australian musicians whose incredible music from the last few years has been the soundtrack to my PhD research: Sarah Blasko, Clare Bowditch and the Feeding Set, The Cat Empire, Frankie Wants Out, Gersey, The Guild League, Lior, Missy Higgins, Ben Lee, The Lucksmiths, Josh Pyke and Matt Roberts.

Special thanks to my circle of close friends, many of whom I have known since kindergarten. All have been very supportive and understanding, especially while I was perpetually "writing up". My absence from social commitments during this busy time has only strengthened my appreciation for these friendships.

Finally, huge thanks go to my wife Chelsea and my family. I can't possibly begin to express my gratitude for all the love and support they've given me.

Michael Wybrow

Monash University
June 2008

Chapter 1

Introduction

Un bon croquis vaut mieux qu'un long discours
[A good sketch is better than a long speech]

— Napoléon Bonaparte

The value of diagrams and illustrations for effectively conveying information and as pedagogical aids has long been recognised. Such graphics are constantly increasing in popularity and usage. They pervade written documents, instructional materials and advertising. In recent years there has been a dramatic and measurable increase in the image-to-word ratio in documents of all types (Horn, 1999b).

Is a picture worth a thousand words? Thorough experimentation and evaluation of human learning has led to the acceptance of the multimedia principle; students learn better from words and pictures than from words alone (Fletcher and Tobias, 2001). For example, the use of "stand-alone" diagrams—containing all elements necessary for complete understanding—has been shown to be significantly more efficient than a mixture of diagrams and explanatory text as a communication method for human learning (Chandler and Sweller, 1991; Horton, 1991; Mayer, 2001b).

Many groups (psychologists, educators, publishers, advertisers, graphic designers, and engineers, to name a few) have an interest in conveying information visually. Such industries have their own theories and conduct research on information visualisation, appending their own labels. Graphic designers know it simply as design. In publishing, it is called *information graphics*; in business it's *presentation graphics*; and in research literature it's known as *scientific visualisation* or *information visualisation*. Software engineers refer to it as *interface design*, and architects speak in terms of *signage* and *wayfinding*. Horn (1999a) brings these together as Information Design, described as the art and science of preparing information to be used by human beings with efficiency and effectiveness.

While many of the groups comprising the field of Information Design have similar or compatible guidelines for the arrangement of information, there are no generally accepted rules to determine good layout for diagrams. Horn states that Information Design varies greatly in style and quality, limiting its usefulness for those concerned with precision and clarity of information.

Good diagrammatic layout depends on many factors, some of which can be drawn from general stylistic advice. Aesthetician Edward Tufte, author of several books on information graphics, warns us of *chartjunk*—unnecessary or confusing visual elements in charts and graphs (Tufte, 1983). More recently, experiments by education researchers have led to a similar conclusion. The *coherence principle* for multimedia learning states that students learn better when extraneous words, pictures, and sounds are excluded rather than included (Mayer, 2001a).

Experiments have shown that element placement in a diagram has important effects on reader perception and understanding (Winn, 1990). This research offers suggestions for good layout, such as clustering components in diagrams to keep relevant information physically close. This advice is echoed by the *spatial contiguity principle* of multimedia learning (Mayer, 2001a), which states that "students learn better when corresponding words and pictures are presented near rather than far from each other on the page or screen".

Other empirical research examines the aesthetic properties affecting user comprehension of various diagram types. Typically, these criteria interact with each other so there are trade-offs between them. Previous work has identified important layout features for comprehension of certain styles of diagrams, such as Unified Modelling Language (UML) diagrams (Purchase, Allder and Carrington, 2002) and node-link diagrams (Ware, Purchase, Colpoys and McGill, 2002), see Section 1.2.

Comprehension of diagrams is depends a lot on their layout. Hence, good layout is an important goal during diagram authoring. Some information graphics (such as maps and plots) can be automatically generated without human authorship; others are manually created, often using diagram editing software. Such software must offer powerful yet flexible placement and layout tools to facilitate the user's creation of a clear diagram.

Unfortunately, placement and layout tools in most existing diagram editors have only a once-off effect. Diagram authors are forced use these tools repeatedly to manually maintain placement and layout relationships to suit the desired drawing style. There is conjecture that such tools would be better implemented as constraints that are permanently maintained by the diagram editor, thus saving the author significant effort. This claim has not been formally evaluated, nor has it been demonstrated how all relevant types of layout tools would fit into this model.

This thesis demonstrates the value of persistent constraint-based layout tools through the results of several usability studies. It explores how they should best be implemented, concentrating on their behaviour and user interface, and iteratively improving them through a user-centered design approach. It suggests techniques for automatic object-avoiding connector routing, so that the user can set certain routing preferences and then the editor maintains optimal connector routes throughout further editing. It also presents a new continuous network layout model where the diagram editor constantly improves the diagram during normal construction and manipulation. The users can set structural styles, stylistic constraints and specify placement relationships, all of which will be persistently maintained by the diagram editor, though under the full control of the user.

Section 1.1 examines various classes of information graphics. Section 1.2 describes diagrams and looks at their layout. Section 1.3 discusses the approach and software available for authoring diagrams and other information graphics. Section 1.4 describes the motivations behind the work and lists the research contributions. Section 1.5 describes the research methodology. Finally, Section 1.6 presents an overview of the thesis structure.

1.1 Taxonomy of information graphics

In the *Semiology of Graphics* Bertin (1983) divides graphic representations into four groups: diagrams, networks, maps, and symbols. He classifies these in terms of information's *components*, *divisions* of components and *correspondences* between divisions of components. For example, each axis in a standard barchart would represent a component. That axis is then broken down into divisions of the component it represents, such as a time scale or groupings of the data. The bars in the chart directly show correspondences between divisions of components.

Bertin says that when correspondences are between all the divisions of one component and all the divisions of another, as in the case of a barchart, then we have a *diagram*. When the correspondences are among all divisions of the same component, then we have a *network*. This matches the standard recognised definition of a network diagram. The divisions are represented as points or shapes, and correspondences are represented as lines drawn between these. Bertin recognises that the difficulty for the designer when drawing networks is to determine the "representation of the component that will produce the simplest structure (the fewest intersections)". When the divisions are arranged in a geographic order then the network is a geographic *map*. *Symbols* occur when correspondence is not between elements of the graphic but is exterior, that is, between an element of the graphic and the reader. Road signs, which convey information assisted by recognition, association and/or learned knowledge of meaning are such a symbol.

More recently, in his book *Information Graphics: A Comprehensive Illustrated Reference*, Harris (1999) separates information graphics (also referred to as charts) into five categories; diagrams, graphs, maps, tables, and other charts. Diagrams, he says, are generally non-quantitative and are primarily constructed of geometric shapes, connected by lines or arrows. They employ text, which may fall both inside and outside of shapes. Graphs (or plots) are a graphical representation of the relationship between two or more groups of information, and usually have one or more axes, at least one of which shows a quantitative scale. Maps are a type of chart that focus on the display of information in conjunction with spatial location data. A table (or matrix) is a chart with information arranged in rows and columns in some meaningful way. Harris admits that not all charts can be fit into these categories and states that some charts must be placed in a fifth "other" category. A floor plan is an example of such a chart. He also states that there is overlap and that a single chart can be placed in multiple categories depending on the criteria employed.

While acknowledging the usefulness of Harris' classification for information graphics, we find it problematic to use the term "graph" to describe numerical plots, due to the

secondary use of the word within the field of mathematics to describe a set of points (or "nodes") connected by lines (or "edges"). This secondary usage is a more appropriate definition for us, so within this thesis the term graph will describe the kind of network diagram shown in Figure 1.2. We will revert to using "plot" to describe the original class of charts from Harris' classification.

We additionally feel that two other categories—technical drawings and illustrations—exhibit distinct differences to other information graphics and are deserving of their own classes. Therefore, for the purposes of this thesis, information graphics are divided into six classes: tables, plots, maps, technical drawings, illustrations, and diagrams. Our breakdown of these classes is illustrated in Figure 1.1. We examine these classes below in further detail since some of the terminology is easily confused. This classification is by no means definitive and as with Harris' version there is overlap and ambiguity between the classes. It is however a useful grouping for our purposes.

- **Tables**: In tables, data is arranged into rows and columns in a meaningful way to allow the reader to quickly look up individual entries. Multiplication tables ("times tables"), calendars, and the Periodic Table are all organised in such a manner. Figure 1.1(f) shows an example reference table displaying Japanese hiragana characters with stroke order.

- **Plots**: Plots are a precise mapping of quantitative data to spatial representations. Often used as a form of reporting, since they can better reveal patterns in the data, plots offer a more comprehensible alternative to a table of numbers. Example plots are pie charts, histograms and scatterplots. Figure 1.1(c) is a pie chart showing a breakdown of military deaths from World War 1 by country.

- **Maps**: Maps are a form of information graphic that display data correlated with geographic information. Some examples are contour maps, weather maps, and navigational charts. Figure 1.1(b) shows a map of Australia with the divisions of its states and territories.

- **Technical drawings**: Technical drawings tend to be precise mechanical or architectural plans. They show physical components to scale, along with other annotations. Examples include engine assemblies, house floor plans, architectural drawings and printed circuit board layouts. Figure 1.1(e) is a technical drawing showing the profile of an ISO metric screw thread.

- **Illustrations**: Illustrations encompass a large variety of creative drawings. Often produced by graphic designers or artists, these may be created as advertisements, street signs, company logos or just as decorative art. These graphics are without stylistic rules and parameters; they range from childish freehand to slick montages of text and graphics. Figure 1.1(d) shows an illustration of the Linux kernel mascot "Tux the Penguin", drawn by Larry Ewing.

- **Diagrams**: Diagrams are usually a simplified, abstract pictorial representation of some data or process. While appearing diverse, they are essentially comprised of

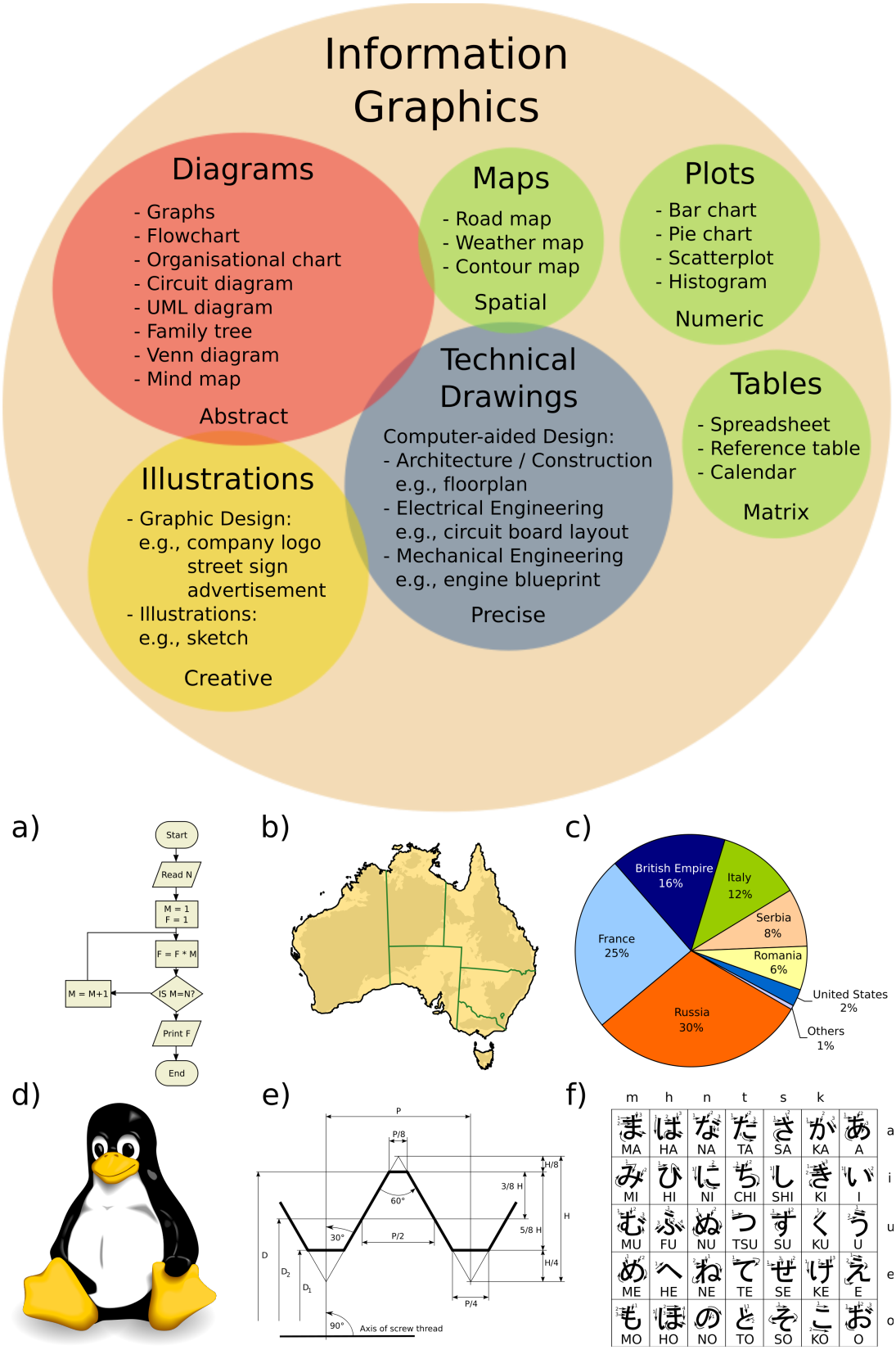


Figure 1.1: Classes of information graphics: a) diagrams, b) maps, c) plots, d) illustrations, e) technical drawings and f) tables.

the same basic primitives: text, geometric shapes (or "nodes") with lines or arrows (also referred to as "edges" or "connectors") drawn between them. Some examples of diagrams are flowcharts, organisational charts, circuit diagrams, UML diagrams, graphs (node-link diagrams), trees, mind maps, Venn diagrams, and family trees. Figure 1.1(a) shows a small flowchart giving an algorithm to compute $N!$ (N factorial), Figure 1.2 a simple directed graph, and Figure 1.3 a protein topology diagram for the protein α_1-antitrypsin.

There is certainly crossover and blurring between the groups. As stylised representations of train or tram networks, metro maps may be classified as maps or diagrams, depending on whether they include accurate geographic locations for stations. Many metro maps discard geographic information in favour of simpler layout and easy interpretation. Similarly, a diagram that depicts an electrical circuit could be considered a technical drawing (or even a map) once it includes information for placement of components and wiring routes upon a printed circuit board.

It is worth noting that information graphics are not restricted to static images. The field of Information Visualisation has developed with the evolution of computers, enabling information graphics that are generated dynamically, or with which the user may interact. Card, Mackinlay and Shneiderman (1999) present a thorough introduction to Information Visualization, which they define as "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition".

The primary concern of this thesis will be the authoring and layout of diagrams. Some diagrams make liberal use of colours and graphics, which makes it unclear when they cease to be diagrams and become illustrations. We will focus specifically on network diagrams, an important and common type of diagram. Network diagrams are composed of shapes or points with some kind of relationship between them expressed with connecting lines, arrows or similar notation.

We will work with network diagrams containing up to a hundred nodes, essentially diagrams of sizes that likely to be manually authored in diagram editors. We mainly consider network diagrams occurring in the biological sciences and software engineering domains. We feel these are important and compelling examples since experts in these fields make heavy use of such diagrams for their work.

1.2 Layout of diagrams

As stated earlier, diagrams are an abstract representation. As such, their structure (the shapes and the connections between them) is important and has meaning—it is usually specified by the data or information being presented. On the other hand, the layout of the diagram (the arrangement of elements on the page) is not usually specified and can be decided by the author to maximise readability or emphasise important information. Some diagrams have particular drawing conventions (e.g. organisational charts are drawn as a tree with levels), but otherwise there can be considerable freedom to place elements for purely aesthetic reasons.

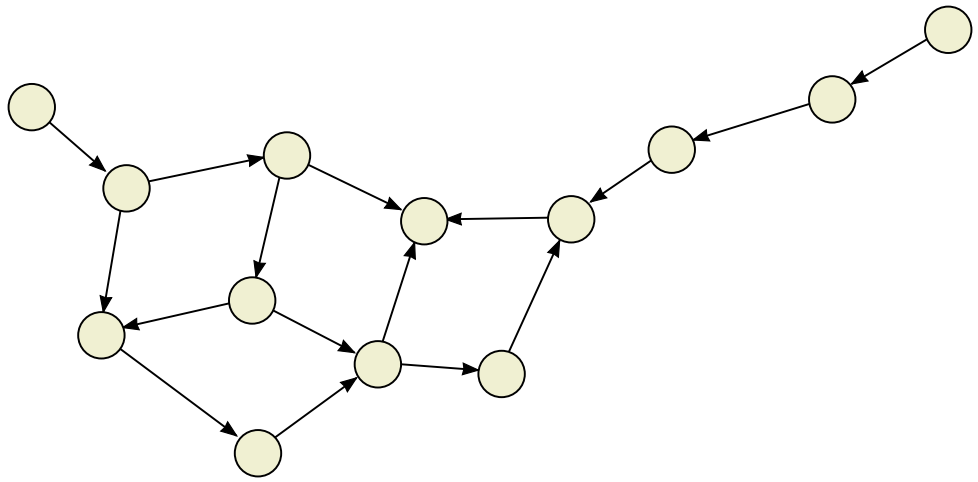


Figure 1.2: Simple directed graph (node-link diagram).

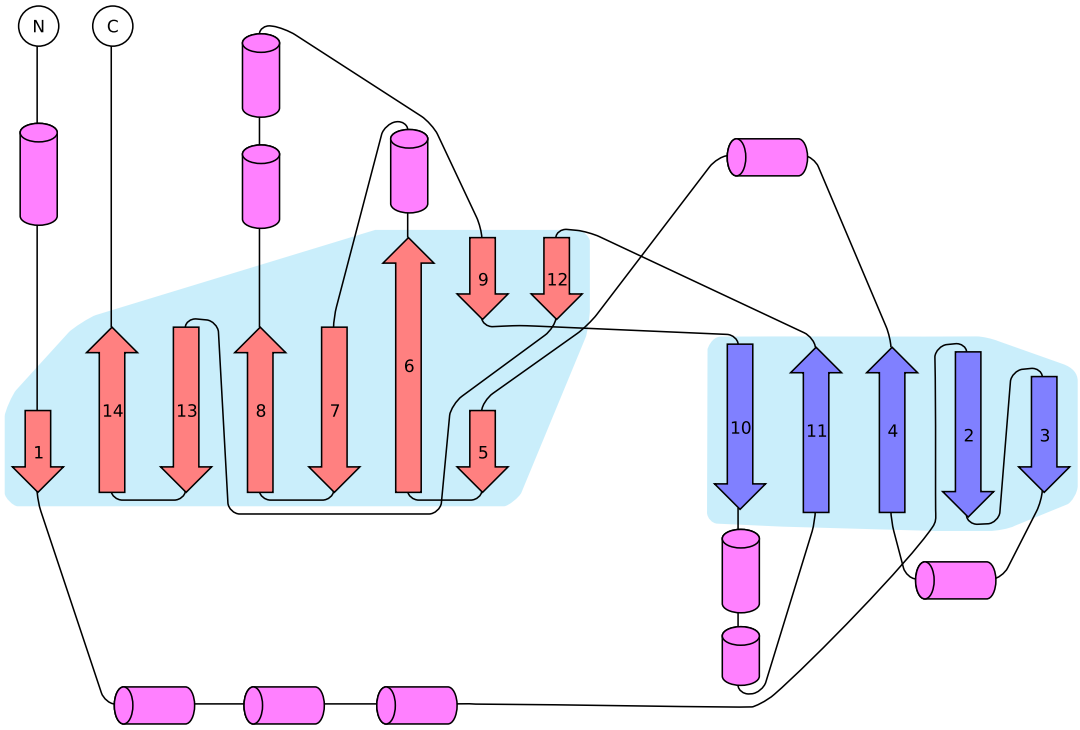


Figure 1.3: Protein topology diagram for the protein α_1-antitrypsin.

Certain aesthetic rules are considered to be desirable for drawing comprehensible diagrams. Though mostly unsubstantiated by empirical research, their value is largely viewed as axiomatic. Some of these properties have been demonstrated to be of particular importance. Purchase et al. (2002) show that for UML diagrams the most important criteria are: few connector crossings, high orthogonality (i.e. arrangement in horizontal and vertical lines), as well as short connector lengths, fewer connector segments and increased continuity between connector segments (reduced bendiness). While the results are based on experiments with UML diagrams, the findings likely apply to a much larger set of diagram types, most of which have very similar features to UML.

Ware et al. (2002) have shown path continuity to be of high value for general node-link diagrams. They find that roughly 38° of connector continuity (a connector deviating from a straight line by 38°) is equivalent in cognitive cost to a single crossing. Both bends and crossings negatively impact a person's ability to comprehend connector paths, although they can't simply be the eliminated since reduction of one will often lead to increases of the other. Here the direct trade-offs between different aesthetics are clearly evident.

Diagram pragmatics are also worthy of consideration. As in natural language, a diagram's layout might imply meaning beyond the pure semantic information of the diagram. An example of natural language could be "John got in his car and went to the shops". In this case the implicature (implied meaning beyond what is literally stated) is that John got in his car *then* went to the shops. Marks and Reiter (1990) and Oberlander (1996) have explored how this idea applies to diagrams. In terms of layout they find implicature mostly occurs when some objects are laid out irregularly when all others are laid out in a regular pattern and vice versa. The user observes this layout difference and attaches some special meaning to a shape or shapes that really have none. Another example is given by Petre (1995) who shows a circuit diagram with convoluted wiring, specifically a particular connector with unnecessary bends and segments left from previous editing. The existence of diagram pragmatics is an argument for some form of automatic placement tools—these could automatically correct convoluted connector paths or help preserve regularity in diagrams. By the same token, it is equally important that the user has control over layout tools and can manipulate them, or altogether override them when they would otherwise result in diagrams with unintended implicature.

Figure 1.4 shows a common educational diagram used to explain the phases of the moon as viewed from Earth. The diagram shows the moon at various stages of its cycle, both from "above" the solar system and as viewed from Earth. Notice that this drawing includes many objects aligned with each other, leading to a large amount of symmetry in the diagram. Some of these alignments are obvious: the Earth and the Sun are centre aligned. Some are more subtle: the label for each phase of the Moon is centre aligned with the Moon image directly above or below it. Some of the objects are distributed equally to provide symmetric spacing. If we neglected these relative placement relationships and just placed each object approximately, the diagram would look messy and this could adversely affect its comprehension.

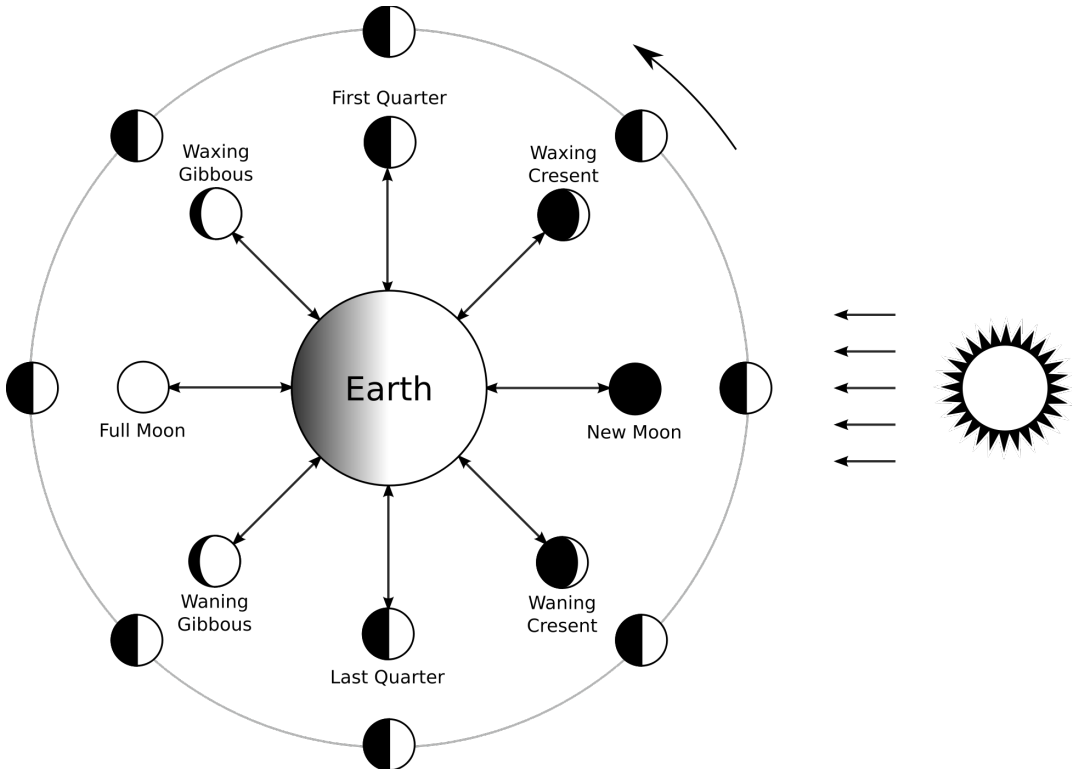


Figure 1.4: A common educational diagram showing the phases of the moon, as viewed from Earth. To be neatly arranged, diagrams such as this benefit from extensive use of alignment and distribution tools throughout their construction.

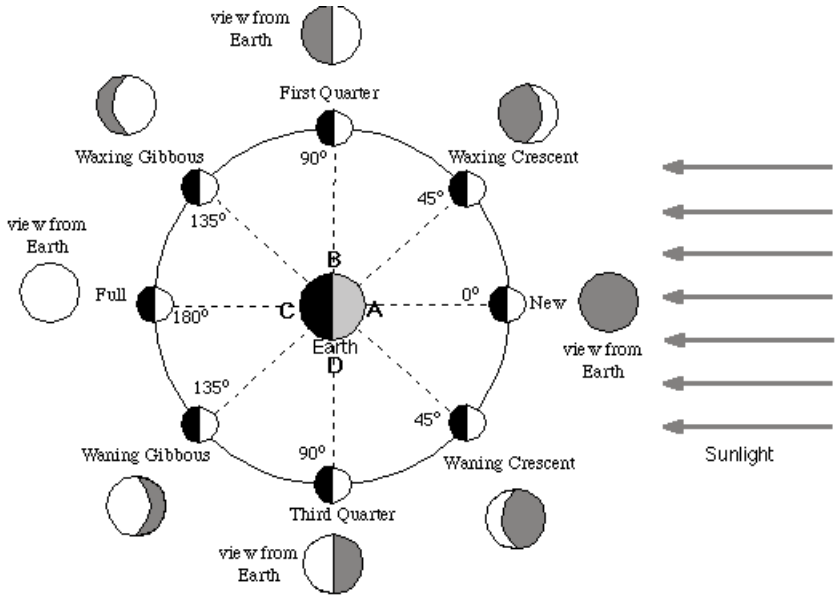


Figure 1.5: A manually drawn "phases of the moon" diagram. This diagram was created by Nick Strobel for his Astronomy Notes website (`http://www.astronomynotes.com/`). Notice several objects are slightly displaced from their ideal positions.

Interestingly, a quick Google search for "phases of the moon" images reveals a large number of computer-drawn, manually arranged versions of this diagram.[1] Such an example is included as Figure 1.5. This image, and many others, have the majority of objects aligned, but often one or two objects are out of place: a text label is not centred or a Moon image is slightly out of position. This suggests the authors desired the placement relationships to be there but had some difficulty enforcing them all.

Good diagram layout involves choice and balancing of aesthetic trade-offs. Diagram authors are required to determine the desired properties of the diagram's layout, and then to enforce these as geometric relationships between objects within the diagram. Diagram authoring software can aid this process by offering specialist layout tools.

1.3 Authoring software

Tables and plots can be automatically generated from their source data by spreadsheet or statistical software. The structure of their layout is largely prescribed by the style of the table or plot being generated. Such software will often limit the manual changes that users can make to generated tables or plots, only allowing cosmetic changes such as assigning colours to values, and adjusting spacing and labels. As a result, these software systems don't usually offer any sort of layout tools to the user.

By nature of representing geographical data, maps have a limited amount of layout flexibility. Options mostly come from ways to visualise the data being superimposed atop the underlying geographical information. Thus, other than for automatic placement of labels, maps require only minimal use of layout tools in their construction. Certain maps, such as metro maps, sacrifice geographic information in favour of a drawing style. As discussed earlier, we would class these as diagrams and expect they would be arranged in diagram authoring software, using the initial geographic positions of stations as a starting point.

Technical drawings are usually authored by engineers or draughtsmen using specialised Computer-Aided Design (CAD) software. Where it is vital that elements are exact, squarely aligned, and precisely attached, it is unsurprising that CAD software makes heavy use of *constraint-based tools*.

A constraint specifies a relationship among element attributes that should be maintained. In a graphical application these will often be geometric relationships, for example, to constrain two shapes to be left-aligned with each other. Traditionally, draughtsmen would work with rulers, compasses, various triangles and splines (flexible strips of plastic, metal or wood, pinned at several points, and used to draw curves). In software, all of these devices can be modelled using constraints.

Constraint-based layout tools are often implemented using *dynamic guides* that show up only during editing and only at relevant positions. Dynamic guides are temporary indicators that are displayed on the canvas marking significant characteristics of objects during interaction, as shown in Figure 1.6. Combined with snapping functionality (where

[1] `http://images.google.com/images?q=phases+of+the+moon`

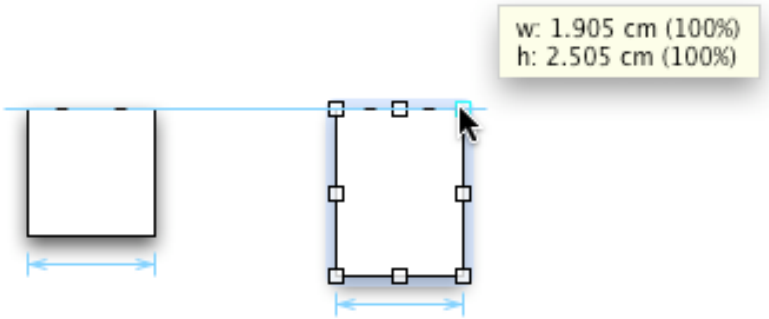


Figure 1.6: While the user resizes a shape, dynamic guides are displayed to indicate when the shape position or dimensions are significant. Here, dynamic guides indicate that the two shapes are equal width and have their top edge aligned.

dragged objects jump to significant nearby positions), they allow the user to place objects easily relative to nodes, intersections, midpoints, quadrants, tangents, perpendiculars, centres, lines and the edges of objects.

In addition, Parametric CAD systems like Pro/ENGINEER[2] and Solidworks[3] use constraints to propagate changes to object parameters throughout the system, updating everything based upon these values. Within CAD systems, constraints can enable manipulation of components within physical models to check that they have the expected range of movement or clearance (Stahovich, 2001). In this way, mechanical devices can be assembled and simulated within the CAD system, allowing problems in the design of components to be identified prior to the manufacture of physical prototypes.

The creators of computer-authored illustrations rely on specialist illustration software with tools for creating a variety of shapes as well as once-off placement adjustment, grouping and cloning (duplication of an object such that when a master object is modified the clones are altered in the same fashion). Placement and cloning features are forms of constraint-based tools appropriate for composition of illustrations. Some illustration software also offers dynamic guides for constrained placement of objects, just as CAD systems do.

In the case of software specifically designed for authoring the more abstract types of information graphics that we have classed as diagrams, most support special connector objects that can be attached to shapes at each end and which automatically stay attached throughout further editing. Connectors are automatically moved when shapes are moved, saving the user from maintaining these relationships themselves. Some diagram editors offer dynamic guides for manual placement, much like illustration and CAD software. Nearly all diagram editors offer once-off placement tools for alignment and distribution of shapes. These automate a movement action that the user would otherwise have to do manually. The more advanced commercial editors support semi-persistent placement relationships with constraints. These take the form of guideline objects to which shapes

[2]Pro/ENGINEER, Parametric Technology Corporation, `http://www.ptc.com/products/`
[3]SolidWorks, SolidWorks Corporation, `http://www.solidworks.com/`

can be "glued". The shapes are moved automatically by the system when the guideline is moved, allowing the placement relationship to be preserved after creation.

The odd placement errors seen in the phases of the moon example (Figure 1.5) are indicative of diagrams created by software that provides only once-off alignment and distribution tools. These once-off tools are perfect when the user need only apply them once, but this requires the user to construct the diagram in a very precise manner such that they complete each component of the diagram, then use the tools to align or distribute it to its final position. This situation rarely occurs in practice—the user is more likely to repeatedly rearrange the same objects with the once-off tools as they make space for new diagrammatic elements, or change decisions about how best to arrange elements. It may be that there is an optimal series of steps to create the final diagram, but inexperienced users will rarely use the optimal series, even when provided with the target diagram (see Chapter 4). This is doubly true for the user who is creating a diagram without a clear idea of the layout of the final product. Much like drafting an essay or a doctoral dissertation, it is likely that very little of the final product remains unaltered from early drafts.

Some diagram editors offer automatic once-off layout tools for hierarchical, tree or organic force-directed layout. The force-directed layout technique models connectors with spring-like forces and nodes as physical weights. Additional repulsive forces are added between each pair of nodes. In this way, connected nodes are drawn together with almost uniform connector lengths, and the repulsive forces cause the graph to spread out. Figure 1.2 is an example of the force-directed style of layout.

The research field of *graph drawing*—described in more detail in Chapter 6—is concerned with exploring algorithmic layout techniques for arranging graphs (Di Battista, Eades, Tamassia and Tollis, 1999). Some relevant work has been in the area of dynamic graph layout, which aims to create incremental layout methods specifically for use in interactive contexts where preserving the user's "mental map" of the changing graph is an important consideration. See Branke (2001) for an overview. Graph drawing strives to produce layouts with aesthetic properties that make them more readable. This makes integration of these techniques into interactive diagram authoring software particularly appealing.

While some of the static graph drawing approaches have filtered into the layout tools of popular diagram editors, the same has not been true for the interactive equivalents, even where implementations of these techniques—such as Dynagraph (Ellson, Gansner, Koutsofios, North and Woodhull, 2003)—exist. Part of the problem is that these dynamic graph layout approaches do not lend themselves well for use in an interactive environment where the user can manipulate all objects and may place constraints upon the movements of certain objects. For this purpose, constrained graph layout (He and Marriott, 1998) has been proposed as a viable solution.

Most diagrams can be created with general purpose diagram editing software, though there are also specific diagram editors available for authoring individual diagram types, such as UML editors or software for mind map or circuit diagram creation.

As we shall see in Section 2.4, existing diagramming tools do not provide the facilities for automatic enforcement of desired diagram aesthetics. In fact, the freedom of placement afforded to diagrams makes optimal diagram layout a combinatorial problem and automatic methods alone cannot provide perfect solutions. Diagram software needs special placement and layout tools to aid the user in setting up and maintaining desired geometric relationships within their diagrams.

The subject of this thesis is integrating semi-automatic placement and layout tools into diagram editors so as to improve their usability and usefulness.

1.4 Contributions

The contributions of this research, outlined in this section, fall into a couple of different categories. First, several interesting results arising from our usability studies into constraint-based placement tools. Second, algorithmic contributions related to incremental connector routing and interactive network layout. Third, software contributions from a connector routing library, and importantly, a usable constraint-based diagram editor.

Placement tools

While existing diagram editors partially cater for layout constraints in diagrams, they leave much to be desired. Many offer only once-off placement tools, meaning that in standard editing the user will have to use them repeatedly on the same shape. This is because once-off placement tools position objects based upon the positions of other objects *at a given instant*. Those that do offer a persistent form of placement relationships (alignment and distribution) are implemented with one-way constraints. Akin to formulae in a spreadsheet application, they allow shape positions to be set equal to the position of a guideline plus an offset, "gluing" the shape to the guide. Moving the guideline causes attached shapes to move, but the "glue" metaphor ends there; a shape can only be stuck to a single shape in each dimension, and moving the shape overwrites its position formula causing it to become "unglued". We hypothesise that this is a major usability issue with existing tools.

Truly persistent placement tools are possible. There is a large amount of research into multi-way constraint solving technologies that can be used to implement them, and solver software is available. However, until now there has been no empirical investigation into their use for this purpose in diagram editors, and it was unknown whether they offer any improvement.

In Chapters 3 and 4 we conduct two formal experiments, comparing the usability of one-way and multi-way constraint-based alignment and distribution tools in diagram editors. This was done to evaluate the usefulness of this new class of multi-way tools and identify any issues they create. The results provide strong support for the hypothesis that multi-way constraint-based placement tools are more usable than existing one-way constraint-based placement tools.

The second experiment leads to several interesting usability-related results. It shows that participants using versions of the tools with immediate feedback (where constrained

objects are updated continuously during manipulation) perform more slowly than those with delayed feedback. While not statistically significant, the slowdown is nonetheless interesting, pointing to a potential issue with the use of direct manipulation in constraint-based diagramming environments. The experiment also confirms previous anecdotal observations on the importance of clutter and constraint comprehension in constraint-based systems.

After further refinements to the design of the tools, the results of a third experiment (Chapter 4) show fading of inactive constraint indicators to be beneficial in reducing the widely reported issue of clutter. This study also investigates the usefulness of an *information mode*, which can be used by the user to visually query how objects are attached to each other via constraint-based relationships.

Connector routing

Many of the connector routing solutions in popular diagramming software automatically redraw connectors, keeping them visibly attached to shapes as the shapes are moved. However, these editors do not always automatically reroute connectors when a better route is available, or when shapes are dropped over existing routes. Their routing algorithms often fail to consider obstacles in the diagram, so shapes and connectors frequently overlap.

Smart object-avoiding connectors can be thought of as another sort of constraint that should be persistently maintained during interaction in diagram editors. Like placement constraints, the user should have some control over their behaviour and be able to override the connector routing where necessary. The routing should take into account desired aesthetic properties for connectors, as noted by experiments into diagram comprehension. This kind of connector behaviour has not been previously available.

In Chapter 5 we present an algorithm for incrementally computing minimal length object-avoiding poly-line connector routings. This technique is fast enough for use in an interactive diagram editor with direct manipulation. The routing algorithm has a cost function that can be modified to account for diagram aesthetics such as number of connector segments, or angle between segments. The algorithm is extended to allow for intelligent routing in the presence of clusters, and post processing steps to reduce the number of crossings and meaningfully display common paths between connectors.

An implementation of the connector routing algorithm is available as an open source software library named `libavoid`.[4]

Automatic network layout

Automatic network layout is another aspect of diagramming that stands to benefit from being persistently maintained. Currently, automatic network layout tools are not provided in interactive environments, or when they are, they are once-off tools being rerun every time an action is made that alters the structure of the diagram. A simple form of persistent layout could check for and remove overlap between shapes. A more complex form might involve a complex graph layout algorithm constantly beautifying the layout while observing

[4] libavoid, Michael Wybrow, `http://adaptagrams.sourceforge.net/`

all placement constraints placed on shapes within the diagram. Obviously the user would require a way to configure or override the layout process, and may not want it running continuously. Interactive constraint-based layout for diagram editors has not been tried.

Chapter 6 presents the development and implementation of a new interaction model for integrating automatic network layout into diagramming tools. In this model the layout engine continuously adjusts the layout in response to author interaction while preserving placement constraints. The author guides the layout by moving diagram components and can alter the layout style by adding or removing placement constraints. Visual feedback is offered when constraints cannot be satisfied.

Dunnart: A usable constraint-based diagramming tool

A major contribution of this research is Dunnart,[5] a truly usable constraint-based diagram editor. As discussed throughout Chapters 4–6, Dunnart contains complete implementations of all the constraint-based placement tools, connector routing, and automatic network layout techniques presented in this thesis. Dunnart is a complete and usable diagram editor, having been iteratively improved based on findings from each of our usability experiments. It was used to produce the majority of diagram examples in this thesis, and can be adapted easily to work with diagrams from specific application domains as illustrated by the real-life case studies in Section 6.8. Dunnart consists of 51,000 lines of `C++` code and is a sizeable piece of software.[6] Dunnart is publicly available for download and the source code will soon be made available under an open source software license.

1.5 Research methodology

We make use of well known usability frameworks and principles for user interface design when evaluating issues with constraint-based placement tools in diagram editors. Formal comparisons of the tools are achieved through quantitative usability studies, each using a between-groups design, with standard parametric statistical analysis of the results. Additional qualitative data is collected via query techniques such as post-test debriefing and surveys.

Design and prototyping of the new multi-way constraint-based placement tools is achieved through a user-centered design process. The underlying algorithms already exist, but we create an interface and define behaviour for the tools. We accomplish this via an iterative design procedure that focuses on the requirements of the user and in which redesign is guided by evaluation of each current iteration.

In the case of connector routing, there are existing approaches for static routing, but algorithmic work is required to create a technique suited to an interactive environment—both runnable incrementally and fast enough for use in an interactive application. We compare our new incremental connector routing algorithm with existing static routing

[5] Dunnart, Michael Wybrow, `http://www.csse.monash.edu.au/~mwybrow/dunnart/`

[6] This figure excludes external libraries which are used for graphics operations, thread management, image loading and constraint solving. It also excludes `libavoid`, which is an additional 7,600 lines of `C++` code.

techniques. Our extensions to the initial technique are developed in response to anecdotal feedback from the use of the algorithm in two different applications. Part of this design is guided by empirical research into diagram aesthetics.

The integration of automatic network layout into an interactive diagram editor is again approached from a user-centered design perspective. The aim is to create an environment in which the layout runs continuously, beautifying the diagram, while the user is able to guide or override it by adding or removing placement constraints or by dragging objects around. An important aspect of this work is to make sure the automatic network layout is integrated with the previous layout tools and connector routing.

Since all the ideas in this thesis are fully implemented in a working constraint-based diagram editor, additional qualitative evaluation comes from various users and/or observers of this software. While their experiences are anecdotal, the fact they are based on real use of the system make them valuable when the new interaction model presented is so radically different from existing diagram editors. That is, the editor's many fundamental differences are not easily tested through standard usability evaluations since such experiments generally need to examine very specific questions. For this reason, such informal feedback is included where appropriate.

1.6 Thesis overview

This introductory chapter has provided some basic background and motivation for the research by highlighting the value of diagrams: discussing how they are authored, and stressing the importance of good layout.

Chapter 2 provides background on usability and user interface design principles. It details the history of constraint-based graphical editors as presented in the literature, concluding with an evaluation of constraint related features in current popular software, both commercial and open source.

The research results are presented in Chapters 3–6, and are arranged in chronological order. Since much of the software design follows an iterative design approach, this process is presented in sequence—design and implementation sections are interleaved with discussion of each user study. Each chapter represents research into different areas of focus. These areas lead directly into one another and are cumulative.

Chapter 3 describes a usability study comparing the one-way constraint-based placement tools present in several popular editors with equivalent tools based on multi-way constraints seen only in research systems.

Chapter 4 presents the new experimental diagram editor (Dunnart) featuring redesigned placement tools. It presents a follow-up usability study designed to verify the results of the earlier study and test the value of immediate feedback for constraint-based tools. Problems uncovered by the study are addressed in design changes to the tools. These changes are similarly evaluated through a subsequent usability test.

Chapter 5 presents a technique for fast, incremental poly-line connector routing suitable for use in interactive applications. It compares the efficiency of the method with an

existing routing implementation as well as showing how it can be extended to account for desired diagram aesthetics and other diagram features such as clusters.

Chapter 6 describes how the layout and routing tools from previous chapters can be brought together with constraint-based graph layout techniques to provide persistent interactive graph or tree layout within diagram editors. It details a new interaction model that allows automatic semi-supervised layout and layout beautification. In this model a layout engine runs continuously, but the user is able to override this engine to improve the layout or add various kinds of constraints between objects.

Finally, Chapter 7 presents the conclusions, along with future directions for the research.

Chapter 2

Background

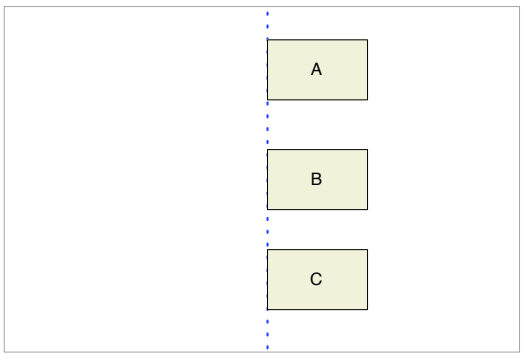
2.1 Introduction

When editing diagrams and other graphical documents it is often useful to be able to specify geometric relationships between the elements, for instance "left-align these three objects" or "equally space the selected objects between the outer two". A once-off movement to fulfil this relationship can be done by simply adjusting the positions of shapes. However, one would often like this relationship to be preserved during subsequent editing. Tools that set up such persistent geometric relationships are generally implemented using constraints.

A constraint specifies a relationship among element attributes that should be maintained. For instance, vertical alignment of three boxes A, B and C can be specified by

$$A.x = L.x$$
$$B.x = L.x$$
$$C.x = L.x$$

where L is an *alignment guideline*, see figure at right.



Over the last four decades there has been considerable effort in developing efficient constraint solving techniques for interactive graphical applications, see surveys in Hower and Graf (1996) and Badros (2000).

One-way (or data-flow) constraints are the simplest, most widely used approach (Vander Zanden, Halterman, Myers, McDaniel, Miller, Szekely, Giuse and Kosbie, 2001). They form the basis for a variety of commercial products including widget layout engines, spreadsheets and diagram editors. A one-way constraint is exactly like a formula in a spreadsheet cell. It has the form $x = f_x(y_1, ..., y_n)$ where the formula f_x details how to compute the value of variable x from the variables $y_1, ..., y_n$. Whenever the value of any of the y_i's changes, the value of x is recomputed, ensuring that the constraint remains satisfied. Thus in the above example they will ensure that if the alignment guideline is

moved then the boxes will follow it. One-way constraints are simple to implement and can be solved extremely efficiently. They are also very versatile since f_x can be any function.

The main limitation of one-way constraints is that constraint solving is directional and hence cyclic dependencies are not allowed, that is, an attribute cannot be defined in terms of itself. In the above example, if box B is moved the other boxes and alignment guideline will not follow it since the constraints only compute values for $A.x$, $B.x$ and $C.x$, and only as a result of changes to the value of $L.x$. A change to B effectively overwrites the formula that caused its position to depend on L. The formula for $B.x$ will also be overwritten if another constraint is applied to $B.x$, such as requiring B's centre to be vertically aligned with some other objects. Thus, with one-way constraints it is generally not possible for multiple dependent constraints to apply to the same object.

As a result of these limitations, so-called multi-way constraint solving techniques have been developed. These approaches fall into four main classes: local propagation-based (e.g. (Sannella, Maloney, Freeman-Benson and Borning, 1993; Freeman-Benson, Maloney and Borning, 1990; Vander Zanden, 1996)); linear arithmetic solver-based (e.g. (Gleicher, 1993; Borning, Marriott, Stuckey and Xiao, 1997; Marriott and Chok, 2002)); geometric solver-based (e.g. (Fudos, 1995; Kramer, 1992)); and general non-linear optimization methods such as Newton-Raphson iteration (e.g. (Nelson, 1985; Heydon and Nelson, 1994)). In multi-way constraints, all variables can potentially be output variables—any variable can be calculated from the values of the other variables. With a multi-way constraint solver, if B is moved then the other boxes and alignment guideline will follow.[1]

The use of both one-way and multi-way constraints for diagramming has been examined in many prototype systems, as we shall see later in the chapter. Although constraints have been the basis for some parametric CAD applications, they have enjoyed only limited use in diagramming software. There are a few reasons that these methods have not yet transferred into widely-used diagramming tools. First, there is complexity in implementing constraint solvers that are fast and flexible enough for use in interactive applications. Second, there is no clear consensus on the form that constraint-based tools should take—research prototypes differ greatly in their behaviour and the interface presented to the user. An important concern is that there has been almost no formal evaluation involving the usability of the prototype systems or the general worth of constraint-based tools for diagramming.

In this chapter we review the use of constraint-based tools in diagram editors—both in research systems and in popular commercial and free software. Section 2.2 first outlines some Human-Computer Interaction and usability concepts that provide the basis for discussion and evaluation of existing systems. Constraint-based graphics and diagramming systems arising from research are examined in Section 2.3, with discussion focusing on the usability of these tools. Section 2.4 explores the features of constraint-based tools within popular open-source and commercial diagram editors.

[1] The guideline is not strictly necessary when using multi-way constraints: the positions of the shapes could just be constrained to be equal to each other.

2.2 Human-Computer Interaction and software usability

Human-Computer Interaction (HCI) is the study of interactive computer-based systems, with particular emphasis on the analysis of their use for the purpose of designing more usable software. HCI arises from the field of ergonomics, which in turn is concerned with improving the design of objects, processes and environments used by humans. HCI is a multidisciplinary area encompassing computer science, psychology, design and several other fields.

In this section we outline some principles for user interface design and describe the *Cognitive Dimensions* framework. Together, these provide a good basis and vocabulary for discussing and evaluating the usability of constraint-based tools in diagram editors.

2.2.1 Principles of user interface design

Dix, Finlay, Abowd and Beale (2003) have presented a useful set of general principles that can be applied to the design of interactive systems to increase usability. They show these principles fit into three main categories. We describe the categories and relevant principles below in terms of our domain.

Learnability

These principles determine how easily new users can begin productively using the system.

- **Predictability**: The system should always behave predictably in response to any particular user action. It should be deterministic, always resulting in the same output for a given set of input. Chapters 3–4 examine the problems with user understanding of multi-way placement tools; essentially an issue of predictability. The connector routing techniques described in Chapter 5 are deterministic, and thus fairly predictability.

- **Synthesizability**: Deals with whether the user can build up a reliable mental model of the system such that they can determine how the system will react for some given input. Throughout the thesis, the synthesizability principle motivates investigation into appropriate interfaces and on-screen indicators for constraint-based tools.

- **Familiarity**: The tools and behaviour of the system should map as closely as possible to the real world and previous tools so as to utilise users' prior experience. Familiarity is utilised in Chapter 6 to avoid providing unnecessary indicators for non-overlapping objects and topology preservation, which act like a (familiar) physical model.

- **Generalizability**: Knowledge of one situation should be extendible to similar situations. For example, the user should be able to align a diamond in the same way they align a rectangle.

- **Consistency**: Similar situations and relationships within the system should behave in a similar fashion to those the user is already familiar with. For example, right-clicking on a diagram object should bring up a context menu for that object, as occurs in most GUI software applications.

Flexibility

These principles refer to the available avenues for information exchange between the system and the user.

- **Dialog initiative**: We wish to maximise the user's ability to pre-empt the system. This is not of so much concern to us since communication in most graphical editors is almost entirely user pre-emptive.

- **Task migratability**: The user should be able to hand over execution of a particular task to the system. This is in fact exactly the service offered by the geometric placement tools in Chapter 4, the automatic connector layout in Chapter 5 and the continuous layout in Chapter 6.

- **Multi-threading**: The user should be able to perform multiple tasks at once. For example, they shouldn't be limited by modal dialogs. This is of concern in Chapter 6, where the automatic layout should not limit the user's interaction choices while it is running.

- **Substitutivity**: The system should allow the user to arbitrarily substitute equivalent input values for each other. For example, we deal with substitutivity in Chapter 4 where our multi-way placement tools can be used to intelligently distribute shapes, guidelines, or a combination of both.

- **Customizability**: This encompasses both Adaptability, where the user should be able to alter forms of input and output, and Adaptivity, where the system should automatically change the interface.

Robustness

These principles cover the support provided to the user for completion and assessment of goals—basically, how the system helps the user accomplish their aims.

- **Observability**: The on-screen representation of the page should be such that the user can evaluate the internal state of the diagram without changing it. Observability is an issue we deal with throughout the thesis as we discuss and improve on-screen indicators for constraint relationships.

- **Recoverability**: The user should be able to undo an action that was realised to have been in error. Undo and redo operations are provided for placement tools in Chapter 4 and automatic layout in Chapter 6.

- **Responsiveness**: The system should react to user actions as quickly possible. Chapter 5 deals with the time taken to compute connector routes; this is an issue of responsiveness. In Chapter 6, responsiveness drives the decision to implement the automatic layout via a threaded model.

- **Task conformance**: The system should offer all tools needed to complete any particular task the user may want to achieve, and in a way the user understands.

Adherence to these principles is a likely indicator that tools will be more usable and hence more useful. We use them to evaluate existing, widely-used placement tools, and design a new set of placement tools based on multi-way constraints.

2.2.2 Cognitive Dimensions

A useful high-level vocabulary for evaluating usability is provided by Green's *Cognitive Dimensions of Notations* framework (Green, 1989; Green, 1991). For interactive systems, the framework aims to enable evaluation of the system in terms of the impact that the system's design will have on its users. The framework is activity-based and focuses on the design choices and trade-offs between potential design options.

Subsequent usability discussion in this thesis will make use of the following descriptions of the dimensions, based on those presented by Green and Petre (1996) and Blackwell, Britton, Cox, Green, Gurr, Kadoda, Kutar, Loomes, Nehaniv, Petre, Roast, Roe, Wong and Young (2001).

- **Viscosity: resistance to change.**
  If a particular goal requires the user to perform many actions, then the system is said to be viscous. Ideally we want the system to have a low viscosity, making it easier for the user to accomplish their goals. For example, the system should allow the user to select multiple shapes and move them together, rather than forcing them to move them individually.

- **Visibility: ability to view components easily.**
  Visibility relates to how easily information is available to the user and whether cognitive work is required to obtain it. We wish to strive for high visibility for information. The user should be able to see required information immediately or at least be able to see a way to quickly reveal that information.

- **Premature Commitment: constraints on the order of doing things.**
  Premature commitment refers to systems that force the user to make decisions before they have adequate information to base the decision on. Within a diagramming application, premature commitment could involve properties for objects that must be chosen at the time of their creation and can not be easily modified later. For example, a connector that must be attached to a chosen edge of a shape before the final placement of shapes has been decided by the user.

- **Hidden Dependencies: important links between entities are not visible.**
  Hidden dependencies occur when the system has dependencies between objects that are not fully visible. A classic example is a spreadsheet, where the user can look at the formula for a cell to determine from which cells it gets its value, but they are unable to determine which other cells take their value from it without potentially examining the formula of every cell. Constraint-based tools have similar dependent behaviour and can suffer from hidden dependencies if this is not taken into account.

- **Role-Expressiveness: the purpose of an entity is readily inferred.**
  Role-expressiveness denotes how easily the user can determine the purpose of interface elements. In terms of diagramming tools, the name by which they are referred, the icon used to represent them and their on-screen appearance all have an effect on the tool's role-expressiveness.

- **Error-Proneness: notation invites mistakes; system gives little protection.**
  User errors can be slips by a user who knows what they are doing but accidentally acts incorrectly, or mistakes resulting from complexity of the system. Error-proneness is used to describe how much the system induces careless mistakes from the user, or how little it does to protect them from such errors.

- **Abstraction: types and availability of abstraction mechanisms.**
  Abstractions describe a grouping of objects, redefined under a common label, that can be interacted with. Styles in a word-processor are a simple example of abstractions, as are some of the constraint-based geometric tools described later in this chapter. They both provide an easier interface for a user to interact with many low-level properties or relationships. Abstractions tend to be beneficial and help to reduce viscosity, although systems with a lot of abstractions may be difficult for the user to learn.

- **Secondary Notation: extra information other than in formal syntax.**
  Systems should give the user a method for recording information other than formally defined data. Good examples of this are fields for notes in database applications, or comments in programming languages. In a diagram editor this may be accomplished by allowing the user to associate arbitrary text with shapes and connectors via labels or some other means.

- **Closeness of Mapping: closeness of representation to domain.**
  Ideally a system's objects should closely relate in behaviour (and possibly appearance) to the real-world objects the system models. For diagram editors, this dimension will apply to any tools that use real-world metaphors to describe their behaviour to the user.

- **Consistency: similar semantics are expressed in similar syntactic forms.**
  Consistency is important since users will often infer behaviour of objects in the interface from other similar objects they are familiar with. Usability is adversely affected if such objects resemble each other but behave completely differently.

- **Diffuseness: verbosity of language.**
  This dimension describes the situation where some objects take up more screen space than is necessary. This reduces the user's available work area. Conversely, there can also be issues with terseness if not enough space is dedicated to certain objects.

- **Hard Mental Operations: high demand on cognitive resources.**
  A representation for objects in the system can cause issues for users if they are overly complex and make excessive demands on the user's working memory. A strategy to combat hard mental operations, when they are unavoidable, is to provide the user with a form of note-taking to lessen the amount that must be remembered.

- **Provisionality: degree of commitment to actions or marks.**
  Provisionality refers to the ability for the user to make provisional actions, as a way of finding out what might happen. This can sometimes be useful in systems that force premature commitment on the user. In a diagram editor, provisionality may be offered by allowing any action to be cancelled—while in progress—by hitting the escape key. If this is consistent throughout the interface, then every action may be provisional.

- **Progressive Evaluation: work-to-date can be checked at any time.**
  The system should allow users to stop at frequent intervals to evaluate their progress so far. This is important for novices, and while not strictly necessary for experts, it is often preferred. In diagramming applications, actions that change the state of the diagram—such as dragging shapes—should show the changes in progress, rather than just showing these changes only once the action is complete.

2.3 Research history of constraints in diagram editors

The application of constraints for use with diagram editing was apparent from even the earliest forms of these tools. As a result there is a long and varied history of exploration into the use of constraints in diagram editors.

2.3.1 Sketchpad

By far the earliest interactive constraint-based drawing system was Sutherland's Sketchpad (Sutherland, 1963). Sketchpad represented the first attempt to allow humans and computers to "converse rapidly through the medium of line drawings" rather than via text-based communication. Amazingly ahead of its time, Sketchpad featured a graphical view of the drawing being constructed, and used a light pen as its primary form of user input. The user could use this light pen like a stylus on a modern-day tablet PC to drag a point within the drawing to a new location. The system would recompute the positions of objects as quickly as possible to maximise feedback to the user—important for both responsiveness and observability of the system.

Sketchpad allowed the user to set atomic constraints, such as specifying that lines be parallel or perpendicular, or that a point lies on a line or circle. These points would

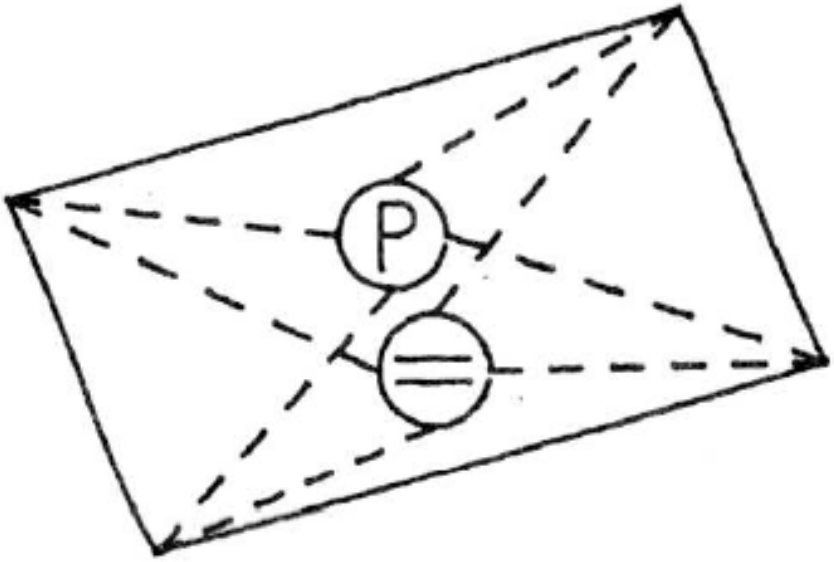


Figure 2.1: A Sketchpad drawing of a parallelogram. The P and $=$ symbols denote that the left and right edges of the shape are both parallel and of equal length.

then be constrained through further editing. Sutherland recognised the importance of an on-screen representation for constraints and Sketchpad had *abstractions* to represent constraint relationships. Sketchpad's abstractions are shown as a capital letter, drawn in a circle and placed at the approximate centre of all the points they relate to, with a dotted line out to these points. Figure 2.1 shows a drawing of a parallelogram in Sketchpad with a couple of abstractions shown. Sutherland acknowledges that such a representation may be problematic since the abstractions can overlap each other and may distract the user from the drawing. For this reason there is an option in Sketchpad to have the abstractions visible or hidden. Also, since all abstractions look similar (other than the letter that denotes their type), they have low role-expressiveness and consistency, inviting the user to make mistakes. Sutherland suggests it would be desirable to have the user create their own custom representation for relationships.

Sketchpad pioneered many ideas influencing the design of the interactive graphical systems that have followed it. Sketchpad's model of interaction was an early attempt at a direct manipulation interface, although the term "direct manipulation" was not coined till almost twenty years later (Shneiderman, 1983). Aside from the recognition of the value of constraints for this kind of diagram authoring, Sketchpad included the concept of a sub-picture, the predecessor to grouping tools in modern diagram editors. Sketchpad had a notion of master and slave objects as part of its sub-pictures, where slave objects could be reused and updated as a result of changes to the master. This is a feature common in modern illustration and design software. Sutherland also recognised the ability to place real-world constraints, like physical stresses or production cost, into technical drawings, a feature present in modern-day CAD applications.

```
put first: rect {
     sw = 0;
     ht = wd = 1;
}
put next: rect {
     nw = first.se;
     ht = wd = first.ht;
}
put last: rect {
     sw = first.ne;
     se = next.ne;
     ht = wd;
}
```

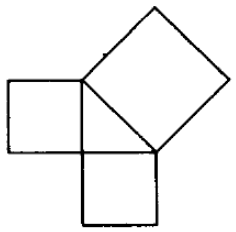


Figure 2.2: An output drawing of three squares from IDEAL along with a subset of the equations used to generate it. The two small squares are constrained to have equal size and share a corner. The third square is constrained to have two corners in common with the other two squares. Each box is made square by the constraint that its width and height are equal ($ht = wd$).

2.3.2 METAFONT, IDEAL and COOL: Text-based systems

METAFONT, IDEAL and COOL were three systems that employed a textual declarative constraint language to constrain the position of objects in a diagram.

Knuth's (1979) METAFONT is a language used for specifying vector fonts dependent on some set of defining properties of the font, such as aspect ratio, font slant, stroke width, and serif size. METAFONT can solve nonlinear equations so long as the user specifies them in an order that makes them linear after substitution. A system like this will always be error-prone, since the writing of formulae in this fashion involves premature commitment and hidden dependencies. Knuth himself points out that one of the key hurdles to its adoption was the requirement for font designers to be mathematicians to be able to specify their font in METAFONT format.

In Van Wyk's (1982) IDEAL, drawings are constructed by defining box primitives that have significant points (the bounding box coordinates) and by defining drawing operations in terms of these significant points. These significant points can also be specified in terms of algebraic equations. IDEAL can solve "slightly non-linear" systems of equations, that is if the equations can be processed in a particular order and after substitution appear linear. Figure 2.2 shows a simple IDEAL drawing, along with the equations used to draw it. In the case of unsatisfiable equations, IDEAL aborts and prints out an error message. Van Wyk acknowledges that the "equation solver's error handling is occasionally a problem; users would like to see the whole minimal set of mutually inconsistent equations, rather than just one element of that set".

Kamada's (1989) COOL is a constraint-based object layout system where diagrams are textually specified by listing the graphical objects and relations that describe either drawing relationships, such as connectors, or geometric relationships, like alignment or ordering.

All positioning is handled by COOL, which deals with the problem of over-constrained systems by allowing the user to specify constraints as "rigid" (satisfied exactly) or "pliable" (solved approximately). COOL includes some graph layout functionality. This is achieved by computing a spring layout for the diagram and then feeding the resultant position data into COOL as linear constraints on x and y positions. COOL solves these constraints along with the constraints arising from geometric relations. Kamada admits that "it may be difficult to use geometric relations besides graph layout constraints, because the users can hardly forecast the resultant layout generated by the graph layout system." Other than this point, Kamada does not discuss the usability of COOL, but like METAFONT and IDEAL it requires users to have a predetermined sense of their desired final layout (and a good understanding of the underlying constraints) when specifying objects and the relations between them. In the case where the system is not easily described on a first attempt it would often require multiple re-specification of the constraints and re-execution of the tool.

2.3.3 Chimera, Pegasus and Penguins: Constraint inference

Several systems have explored the idea of interactively inferring geometric constraints: Chimera (Kurlander and Feiner, 1993) with persistent constraints; and Pegasus (Igarashi, Matsuoka, Kawachiya and Tanaka, 1997) with once-off position adjustment. The Penguins framework (Chok and Marriott, 1998) provided persistent constraints with syntax-based inference.

The Chimera editor (Kurlander and Feiner, 1993) takes snapshots of an initial scene as well as one or more additional snapshots, and then identifies constraints that are satisfied in all snapshots and applies these to the objects in the scene. This avoids the issues in having the user manually specifying the geometric constraints. The user can test what constraints are present in the scene by manipulating objects with the mouse. The idea is that if something behaves unexpectedly, constraints can be turned off, the object manipulated freely and another snapshot taken which will eradicate the undesired constraint in question.

Chimera captures both absolute and relative geometric constraints. It can infer: fixed vertex location; coincident vertices; distance between two vertices; relative distance between two pairs of vertices; distance between parallel lines; relative distance between two pairs of parallel lines; slope between two vertices; relative slope between two pairs of two vertices; angle defined by three vertices; and equality between two angles, each defined by three vertices.

Chimera's authors recognise that debugging, editing and refining constraints are difficult tasks. For this reason Chimera has a graphical browser that can visualise constraints, and filters to show only a single kind of constraint. Chimera may be appropriate for heavily constrained scenes since it starts with all possible constraints enforced and is subtractive. Other situations, where there are few constraints, are better handled by explicitly specifying the required constraints. Where the user has to remove a lot of constraints, the system will exhibit high viscosity and hidden dependencies.

The recoverability of Chimera's constraint inference system is also an interesting question. Once a snapshot causes a constraint to be removed, how does the user get that constraint back if it was only accidentally removed? We assume that the user can delete a snapshot from the system, but then this also involves the difficult task of determining the snapshot at fault. Chimera's authors do not discuss this situation.

Pegasus (Igarashi et al., 1997) is a drawing system that reads pen input, performs beautification of strokes, and then infers higher level constraints. When there are multiple possibilities it shows them to the user and they choose the desired one by tapping the pen. The authors note there are issues with this interface where there are several possibilities available—an issue of viscosity. Pegasus appears useful for technical drawings where features like symmetry and parallel lines are commonplace.

The Penguins framework (Chok and Marriott, 1998) takes constraint inference one step further, offering a framework for developing *intelligent diagram* editors for individual classes of diagrams. It allows generation of diagram editors that are aware of semantics and can check that a diagram is syntactically correct. Such editors parse the diagram as it is constructed and can create and maintain geometric constraints representing semantic relationships within the diagram

2.3.4 Juno and Juno-2: Double-view editing

Juno (Nelson, 1985) and its successor Juno-2 (Heydon and Nelson, 1994) offered *double-view editing* where the user could textually modify constraints, as well as graphically via direct manipulation.

Juno offers *geometric specification*: geometric constraints that become numerical constraints on the positions of points. It allows non-linear constraints, so supports parallel and congruent lines, which correspond to quadratic equations. It uses Newton-Raphson iteration and produces predictable output so long as the points have reasonable starting positions—the solver needs hints for the positions of unknown points. These constraints are presented as high-level tools. For example, Juno offers a horizontal and vertical t-square for alignment, a compass for equalising distances, parallel bars for equalising directions and a snowman for freezing points. A Juno drawing and the actions required for its construction are shown in Figure 2.3. Juno doesn't support inequalities, like greater-than constraints that can be used for modelling left-of behaviour. Constraints are not visualised graphically, so they must be removed or modified by editing the textual code of the diagram. While the high-level metaphors that Juno use for describing constraints have a close behavioural mapping to real-world drafting tools, the method of creating or modifying these relationships does not.

Juno allows folding of multiple drawing operations into a procedure defined in terms of a set of control points, thereby creating a useful abstraction. The user does this by drawing a shape and then clicking the fold button and choosing the points which are to be parameters. The authors note that it would be nice to have compound constraints for situations such as creating a right angle.

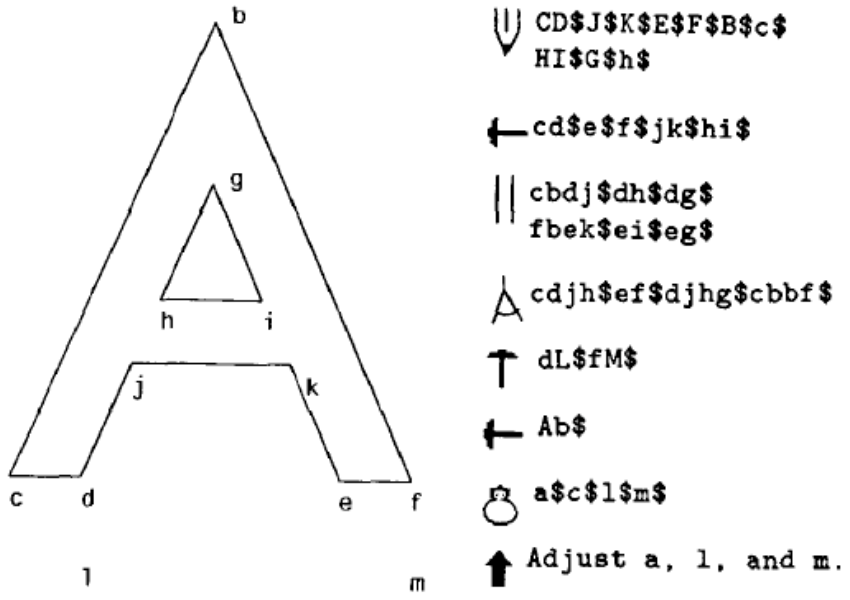


Figure 2.3: A drawing of the letter *A* constructed in Juno, along with a list of the user actions required to construct it.

One issue with constraint-based systems is that under-constrained problems can result in solutions far from what the user may have intended—that is, the constraint solver can modify variables by a large amount and still have the constraints satisfied. To avoid this problem, Juno uses current variable values from the graphical view as hints for constraint solving.

Juno features a drag tool that updates the drawing in real time. The location of the mouse is used as the hint for the point being dragged. The user can merge two points by dragging one over another; this replaces the original point in all expressions with the target point. The user can also select groups of points for dragging, though it is only fast enough to solve small examples.

Juno-2 led to the following observations about direct manipulation and feedback in constraint-based systems (Heydon and Nelson, 1994):

> "The ability to drag a point while the system continuously solves the constraints and updates the drawing is invaluable, both because it provides a way of making fine aesthetic adjustments, and because it allows motion along a continuous trajectory through the constraint solution space. The latter benefit is especially noticeable in cases where the solution space contains discontinuities. Using the Adjust tool, it is quite easy to inadvertently flip into another branch of the solution space, but to do so using the drag tool requires a conscious, rapid motion of the mouse."

Since most diagrams have more than two degrees of freedom, dragging could involve either translating or scaling. If particular results are desired, then certain points need to

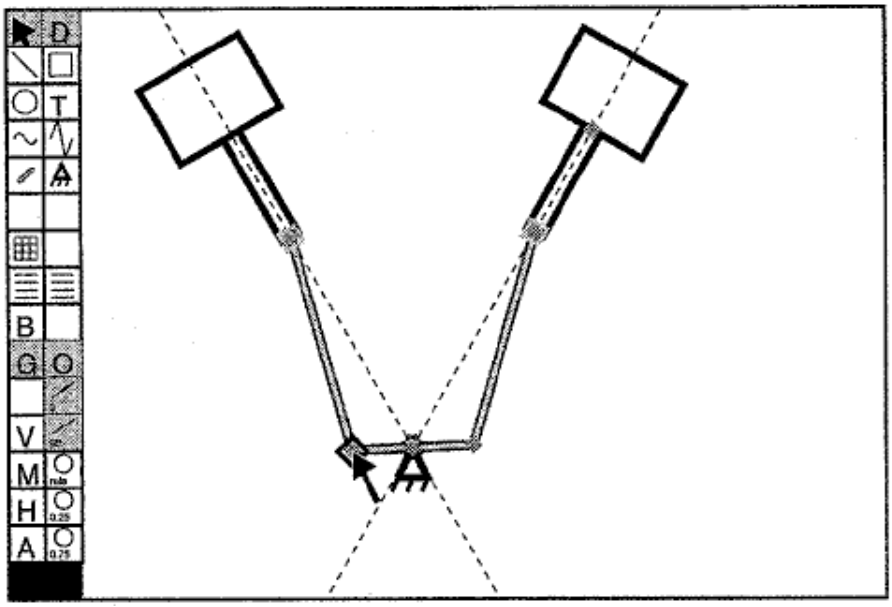


Figure 2.4: Briar's interface, showing a drawing of pistons in an engine. When the user moves the selected point, the pistons are constrained to move as a result.

be frozen. For example, in Figure 2.3, the user has set up the drawing so that there a three frozen "handles"—'a' (missing from the original figure, should be directly to the right of 'b'), 'l' and 'm'—that the user can move manually but the solver cannot. They have also frozen the point 'c'. Careful examination of the constraints will reveal that dragging 'a' adjusts the height of the letter, 'l' adjusts the stroke width and 'm' adjusts the width of the letter. It is obvious that such a system will be error prone to users, due to hidden dependencies and issues of consistency in point movement behaviour. Such a system also involves hard mental operations when setting up the constraints in a logical and useful fashion, as well as for users other than the original author when they must make sense of the behaviour of the diagram during interaction. Juno and Juno-2's authors state that use of constraints in Juno "remove the tedium of graphical programming, but require careful thinking" and that the process can be "daunting to novice users".

2.3.5 Briar

Briar (Gleicher, 1992; Gleicher and Witkin, 1994) allowed creation of constraints by letting the user drag objects into new relationships and then prompting the user with the option of making those relationships persistent. They explain that previous specify-then-solve constraint-based systems suffer from "solving constraint-satisfaction problems from arbitrary starting points, presenting state jumps to users, and coping with conflicts". These issues occur primarily because the system must alter the state of the model to satisfy new constraints. Briar's approach, which is termed *augmented snap-dragging*, avoids these problems since constraints will already be satisfied before they are added.[2]

[2]Snap-dragging (Bier and Stone, 1986) is a technique which uses a gravity metaphor where, as the user drags a shape, it will snap and connect to significant objects such as guidelines.

Briar has what the authors refer to as *differential constraints*, effectively the ability for constrained models to be dragged. They say the continuous motion and ability for users to experiment with models conveys most of the information about constraints, and is a useful tool for understanding and debugging constraints. Figure 2.4 shows a constrained diagram of pistons in an engine. The user can drag the selected point and see the pistons move as a result. Briar has a free mode where the user can move things freely without any constraints. Violated constraints are then removed when the constraint solving is turned back on. Users are also able to "rip" objects off of constraints by dragging with a modifier key held down.

For setting up constraints other than contact between objects, Briar has alignment objects that are not part of the diagram itself, but exist as objects to which other shapes can be constrained. These alignment objects include slope lines, angle lines, and distance lines. Juno's authors acknowledge that such alignment objects could clutter the drawing and cause issues. They say "such problems can be partially combated by selective display and better use of alternate visual cues such as dimming".

2.3.6 Broom alignment metaphor

Raisamo and Räihä (1996) present a completely different interface for direct manipulation alignment tools. Their idea is essentially an "alignment stick" that the user can use like a ruler to push a group of shapes, thus aligning them. The length of the stick can be adjusted at any time and individual objects can be "locked" (via an option on their context menu) so that the alignment stick ignores them. The alignment stick can have any rotation and shapes are given alignment points that can be arbitrarily positioned using a special tool. This allows the stick to push against any particular point on the shape.

Separately, Robbins, Kantor and Redmiles (1999) present a similar alignment tool interface based on a push broom metaphor. Their broom tool is a combination of alignment and distribution as a single direct manipulation tool. Clicking and then dragging on the canvas activates a broom in the direction of the dragging. This can then be used to push shapes into alignment. Pulling back the broom causes shapes to revert back to their previous position. Pressing space bar distributes shapes equally along the head of the broom. They perform a simple study, concluding that the broom provides a more natural and direct interaction style than standard alignment tools—they claim that the broom results in significantly less mouse movement and dragging. This would result in the system having low viscosity.

2.3.7 GLIDE

GLIDE (Ryall, Marks and Shieber, 1997) provides an interface targeted towards collaborative diagram creation between the computer and user. Ryall et al. propose to achieve this "with the computer performing local optimizations and the user responsible for global control". In their own words:

> "The user, who controls the design process at a global level (the choice of VOFs and the gross placement of nodes in the diagram), can easily guide the

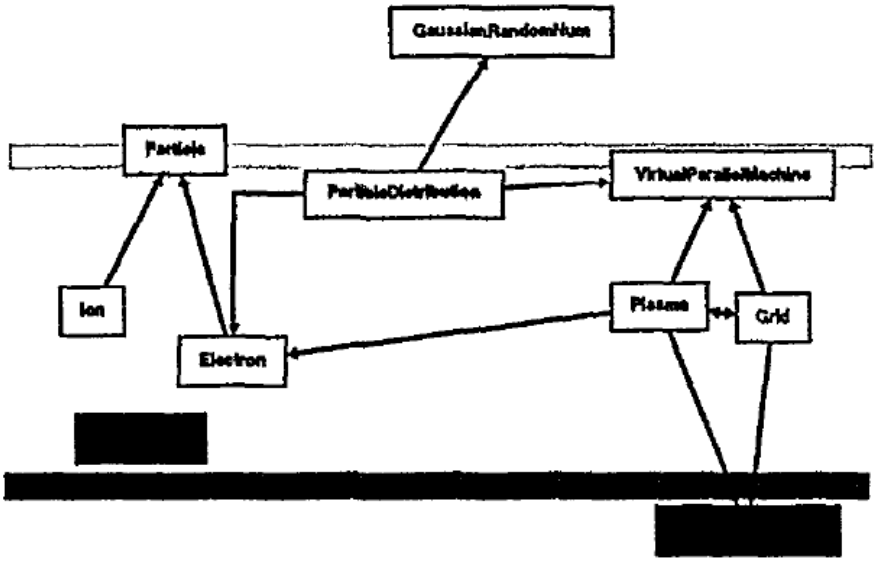


Figure 2.5: A diagram in GLIDE, showing the user adding an alignment Visual Organisation Feature.

> computer to find more satisfactory solutions and acceptable layouts by moving nodes or adjusting VOFs."

Their constraints are presented as high-level Visual Organisation Features (VOFs), and theirs is the first system to allow interactive specification and manipulation of these.

GLIDE offers the following VOFs: alignment, even spacing, sequence (keeping nodes in a particular order in the vertical or horizontal plane), clustering (grouping nodes as much as possible), zone (non-overlapping of sets of grouped nodes), symmetry, and hub shapes. These VOFs are modelled as springs, so may be satisfied to varying degrees. They have an on-screen representation of VOFs that allow them to be manipulated and removed via direct manipulation. Figure 2.5 shows the user adding a horizontal alignment VOF to a diagram in GLIDE.

GLIDE offers the user a lot of control over the layout, but a concerning aspect of its behaviour is that VOFs are not guaranteed to be satisfied. This means two objects that have been constrained to be "aligned" may not be properly aligned if there are other forces acting on them. This leads to problems with the consistency in the layout behaviour. Also, the user may be subjected to hidden dependencies and hard mental operations if they try to determine why the layout is behaving the way it is, and are forced to understand the underlying spring forces used for its implementation.

In nearly all of these papers, there are comments to the effect that constraints are useful but it is difficult to determine a good user interface that allows the user to easily understand and control the constraints.

(a) (b)

Figure 2.6: Effect on layout due to once-off left-alignment of shapes A and C with shape B. Shapes A and C are moved into line with shape B.

2.4 Diagram editors in practice

In this section we examine the constraint-related features present in some popular commercial and open-source diagram editors. Specifically we consider the features of Visio,[3] Dia,[4] ConceptDraw,[5] OmniGraffle,[6] SmartDraw,[7] yEd Graph Editor,[8] and the Tom Sawyer Layout Assistant for Visio.[9] We focus here on the behaviour and interface of these features, discussing these in terms of their usability.

2.4.1 Once-off placement tools

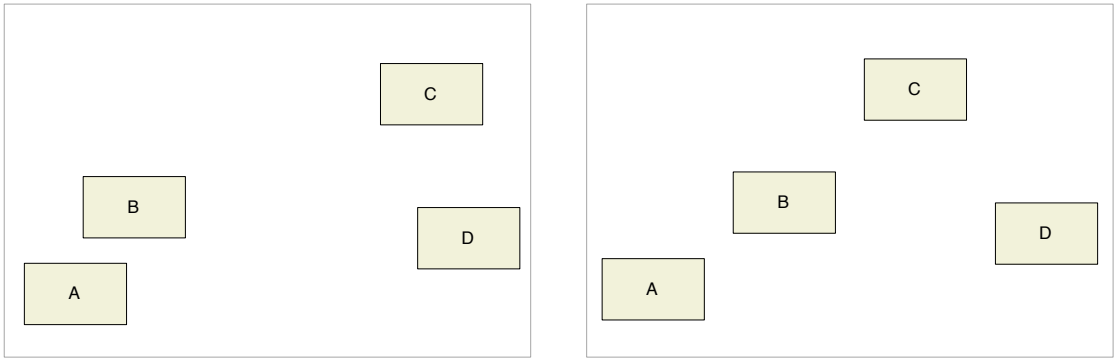
Most diagram editors provide once-off alignment and distribution tools that adjust the positions of the selected objects. Some examples of software that offer these tools are Visio, ConceptDraw, OmniGraffle, SmartDraw, Dia and yEd.

When shapes have been aligned through once-off alignment their physical layout on the page will have changed, but their behaviour or properties are otherwise unaffected. In most systems, alignments work by adjusting the positions of all the shapes in the selection to align with a lead object. Figure 2.6 shows changes to a diagram as a result of left-aligning all shapes with shape B.

Distribution leaves the two outermost objects where they are and distributes the remaining objects in the selection equally between them. The user has control over the specific type of distribution used. Once again, no lasting relationship is created. An example of distribution is shown in Figure 2.7 where all shapes have been horizontally distributed by their centre.

The key limitation of once-off placement tools is that they have no memory. Thus, if the user aligns some objects and then distributes one of these objects by creating a

[3]Visio Professional 2007, Microsoft Corporation, `http://office.microsoft.com/visio/`
[4]Dia 0.96.1, Dia developers, `http://live.gnome.org/Dia`
[5]ConceptDraw 7.2.0, Computer Systems Odessa, `http://www.conceptdraw.com/cd7/`
[6]OmniGraffle Pro 4.2.1, The Omni Group, `http://www.omnigroup.com/omnigraffle/`
[7]SmartDraw 2007, SmartDraw.com, `http://www.smartdraw.com/`
[8]yEd Graph Editor 3.0.0.3, yWorks, `http://www.yworks.com/products/yed/`
[9]Tom Sawyer Layout Assistant 5.5 Pro, Tom Sawyer Software, `http://www.tomsawyer.com/lav/`



(a) (b)

Figure 2.7: Effect on layout due to horizontal distribution of shapes A, B, C and D by their centre. The outermost shapes, A and D, remain in place, while the remaining shapes are moved so they are spaced equally between the outer two.

new distribution relationship, that object will be moved to satisfy the distribution and the user may need to realign the original objects again.[10] This results in a viscous system where many placement actions have to be used to accomplish layout tasks. Also, because the user can achieve the desired layout with a minimum number of steps if they apply the placement actions in a particular order, the system can be seen to involve premature commitment. Maintaining a placement relationship becomes a secondary task that the user must continually address while working on their main task. Ideally, as the task migratability principle suggests, we should be able to hand off this job to the diagram editor.

2.4.2 Manual placement with dynamic guides

An alternative approach to once-off placement tools are *dynamic guides* or *smart guides*. These feature commonly in drawing software like Adobe Illustrator[11] and CorelDRAW,[12] but are also present in some diagramming software, for example, OmniGraffle.

In OmniGraffle, smart guides are semi-transparent guidelines and accompanying distance values that show up during editing. As the user drags or resizes objects, snapping and smart guides give the user an indication of when the object reaches a significant position or size. Smart guides tell the user when the selected shape is spaced equally between two others, has the same dimensions as another shape, or is currently in line with some object.

Similarly, dynamic guides in CorelDRAW and Illustrator show up only during editing and only at relevant positions. They are displayed as infinite length lines that intersect significant points on objects, as discussed in Chapter 1.

[10] The "group shapes" feature of most diagram editors could be seen as a method of preserving alignment. However, persistent alignment would allow us to move a single shape vertically along a vertical guideline without the other shapes in the alignment being moved, a grouped alignment does not allow this. Also, shapes can usually still be individually dragged within a group (without moving the other shapes in the group), which allows them to become unaligned, albeit though manual intervention.

[11] Illustrator CS3, Adobe Systems Incorporated, `http://www.adobe.com/products/illustrator/`

[12] CorelDRAW Graphics Suite X3, Corel Corporation, `http://www.corel.com/coreldraw/`

(a) (b)

Figure 2.8: Effect on layout due to one-way left-alignment of shapes A and C with shape B. In contrast to Figure 2.6, a guideline is created and all three shapes become aligned with and attached to it.

Visio has some support for dynamic guidelines but these are not enabled by default. Moreover, its support is somewhat limited; it doesn't offer such a range of snapping options as in the drawing programs.

Obviously, when dynamic guides "suggest" the correct desired relationship they are faster to use and more powerful than manually specifying relationships through dialog boxes. On the other hand, they also introduce their own issues. Firstly, snapping to the dynamic guides needs to be enabled to make them useful. In this case objects often snap to relationships the user isn't interested in, preventing access to particular relationships or hindering precise placement of objects. Secondly, dynamic guides can be distracting when there are many objects on the page and additional dynamic guidelines that will flash in and out of visibility when objects are dragged across them. With large numbers of objects, these problems are exacerbated and snapping often becomes error-prone.

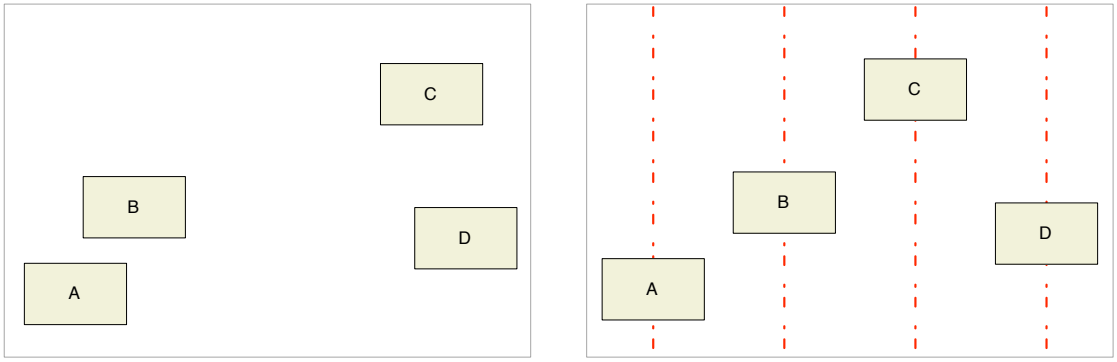
As with the once-off placement tools, users may have to repeatedly perform similar actions to maintain relationships between objects after subsequent moves. They therefore suffer problems with viscosity and premature commitment.

2.4.3 Semi-persistent placement with one-way constraints

In addition to once-off placement tools, Visio and ConceptDraw provide a persistent form of alignment through the use of guidelines. Guidelines are purely placement aids; they act like normal manipulatable objects on the page but are not part of the final diagram, that is, will not be visible on printed versions of the diagram. Visio also provides a guideline-based persistent distribution tool.

One-way constraint-based alignment tools work by creating a guideline connected to the lead object in the selection. The tools then adjust the positions of all the other selected objects to bring them in line and glue them to the guideline, as shown in Figure 2.8. Often snap-dragging is provided as a means of attaching shapes to existing guidelines.

Once shapes have been glued to a guideline, they can be moved by moving the guideline. Unfortunately, one-way constraints only allow us to specify that the shape is constrained



(a) (b)

Figure 2.9: Effect on layout due to one-way horizontal distribution of all shapes by their centre. Guidelines are created for all shapes, and the shapes are attached to these. The guidelines (and shapes along with them) are spaced equally between the outermost two. This differs from Figure 2.7 where the shapes were simply moved to be equally distributed.

to align with the guideline. They do not specify that the guideline is constrained to align with the shape. As a result, moving shapes directly (rather than via guidelines) will always overwrite the position formulae used to describe the constraint-based relationship. This unglues them from guidelines.

Like alignment, Visio's persistent distribution tools are implemented using guidelines. The distribution tool creates a guideline for each selected shape and glues the shape to it. The tool then takes the two outermost guidelines and distributes the other guidelines (and attached shapes) equally between these, as shown in Figure 2.9.

As a result of using the tool, we end up with a persistent relationship that can be further manipulated. The outermost guideline on each end of the distribution can be dragged, effectively resizing the entire distribution. The other guidelines in the distribution cannot be dragged, because they are dependent on the positions of the outermost guidelines. Visio changes the colour of distributed guidelines but otherwise gives no visible indication of the distribution relationship. This means there is no easy way to remove the relationship other than to delete the guidelines. Here the user cannot easily infer the role of distributed guidelines.

The alignment and distribution tools themselves are required to move shapes to set up relationships. This breaks shapes from their prior alignment or distribution relationships.

We can see that from the user's perspective one-way constraint-based placement tools have a serious drawback: alignment and distribution relationships can break due to manipulation of objects involved in the relationship or because more than one constraint is applied to the same object. Not only do these relationships contain hidden dependencies, but these dependencies can change quietly in surprising ways, thus inviting errors. This is a problem inherent in one-way constraints—each constraint has a fixed direction and an attribute can only have a single formula associated with it. Chapters 3–4 partially solves these problems by suggesting the use of multi-way constraint-based placement tools.

2.4.4 Persistent placement with multi-way constraints

Since multi-way constraints can have any input or output, they can be used to create truly persistent placement tools where the created relationship is a constraint that is maintained through all further editing until it is explicitly removed, even in the presence of other constraints.

As we saw in Section 2.3, tools based on multi-way constraints have been widely explored in research fields, but have not made the transition into any of the mainstream diagramming software. Chapter 4 explores how this step should best be accomplished.

2.4.5 Automatic connector routing

As with multi-way constraints, there has been much research into techniques for connector routing. We discuss this research later, in Section 5.2. Although routing algorithms have been widely explored for the specific fields of robot path planning and PCB layout, such techniques have not found use within popular diagramming tools for connector routing, as we will demonstrate below. In keeping with our survey of placement tools, we will focus on the behaviour and interface for connector routing tools, from the view of the user.

Most diagram editors and graph construction tools provide some form of automatic connector routing. They typically provide orthogonal (i.e. routes using only horizontal and vertical line segments) and some type of poly-line or curved connectors. Usually the editor provides an initial automatic route when the connector is created and again each time the connector endpoints (or attached shapes) are moved. The automatic routing is usually chosen by an ad hoc heuristic.

Both Microsoft Visio and ConceptDraw provide object-avoiding orthogonal connector routing. In these two applications the routes are updated only after object movement has been completed, rather than as the action is happening.

ConceptDraw offers orthogonal object-avoiding connectors that are updated live as attached objects are dragged, though not if an object is moved or dropped onto an existing connector's path. ConceptDraw's method for routing does not use a predictable heuristic and often creates surprising paths, and these routes do not always avoid crossings with shapes, even when the crossing is unnecessary. Sometimes these routes even overlap the shapes to which the connector is attached. This is especially noticeable when dragging shapes through a large range of possible positions. It also seems that the routing gets stuck in certain configurations, sometimes moving a shape downwards causes a connector to oscillate between a couple of different configurations. ConceptDraw's routing ignores the predictability principle of user interface design. ConceptDraw allows connectors to be connected to the centre of shapes and has an option for connectors to auto-choose their port on the shape.

Visio offers orthogonal connectors, as well as curved connectors that roughly follow orthogonal routes. Visio's connectors are updated when the attached shapes are moved or when objects are placed over the connector paths, but only following either of these events. Again, connector routing in Visio does not use a predictable heuristic, such as minimizing distance or number of segments. Visio updates its connectors dynamically as objects are

resized or rotated, but if there are too many objects for dynamic updating to be responsive it reverts to calculating paths only when the operation finishes. Also, Visio doesn't revert back to better routes if a shape is moved out of a connector's path until that connector is rerouted for another reason. Its curved connectors are purely orthogonal connectors, drawn as curves—so if a shape is close to a corner of the connector, the curved connector will overlap the shape. If there isn't a fairly simple route,[13] then it reverts to a default (shortest) path ignoring all shape-connector crossings. Visio's connectors can "reroute freely" where they always use the "best" route available, or "reroute on cross-over" where they only switch routes when a route becomes invalidated by a shape being placed over a connector. Visio doesn't take connector crossings into account when routing. However, it does have an option to draw connectors so they visibly arch over other connectors with a small semicircle where they cross.

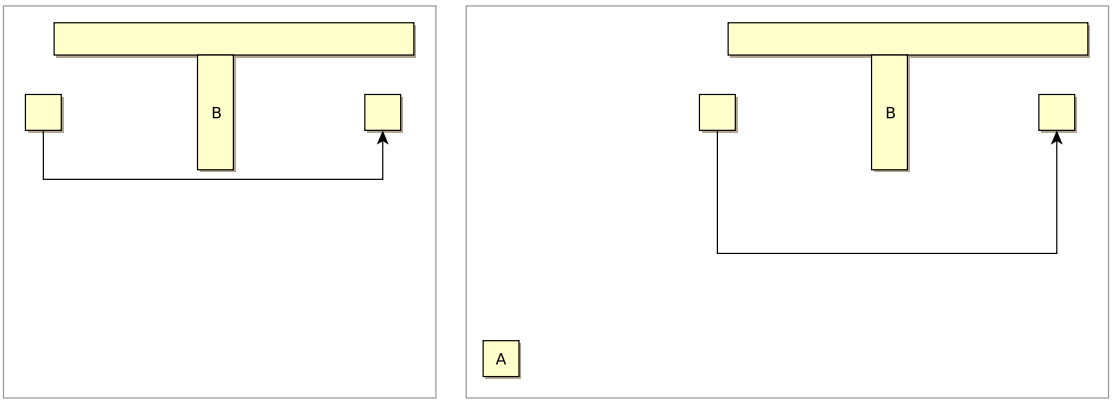
The graph layout library yFiles[14] and its accompanying demonstration editor yEd offer both orthogonal and "organic" edge routing. Both of these are layout options that can be applied to a diagram or selection but are not maintained throughout further editing, so after any changes to the graph the router must be rerun manually to determine new routes. After routing, as shapes are moved and resized, only the last segments of each attached connector (i.e. the segment at each end connected to the shape) are automatically redrawn. The rest of the route remains unchanged.

Organic edge routing in yEd results in curved poly-line edges generated from a force-directed layout where nodes repel edges. The side-effect of this is: if the existing layout before running the router has overlap between edges and nodes, the router tends to move the nodes rather than routing the edges around them. This is because the organic routing is coupled with a layout algorithm. The documentation also states "this algorithm will only work correctly if there already is enough room between each pair of nodes in the whole graph, that is, there should be at least three times the minimal distance room between two nodes." This behaviour is certainly undesirable in an interactive diagramming application where the user may have specifically chosen positions for objects and does not want them moved in the process of routing connectors.

The orthogonal routing in yEd only routes edges around existing shape positions. The user can choose the particular side of a shape that they would like each edge routed to, or this decision can be made automatically by the router. Orthogonal routes are computed to avoid nodes within the graph. The user is able to alter the weighting between the router's use of *space driven* or *centre driven* search. The former gives paths that are closer to the centre point of the straight line between the edges two endpoints. It also tends to lead to edges being routed more tightly around obstacles whereas the latter results in edges being routed through the centre of free space. The user is also able to specify that edges are routed on a grid, and may add a routing penalty to reduce unnecessary crossings. yEd also does a small local crossing reduction by reordering all edges routed into the same side

[13] It is not easy to determine the metric that Visio uses to determine when to give up doing object-avoiding routing: it could be based on the route having more than a certain number of segments, or being longer than a particular length.

[14] yFiles, yWorks, `http://www.yworks.com/products/yfiles/`



(a) (b)

Figure 2.10: Surprising routing in yEd: (a) Without shape A, the connector routes tightly around obstacle B. (b) With shape A present, the connector takes a wider route.

of a node. In general, the orthogonal routing is quite good but not always predictable, especially with centre driven search where the order of edge processing becomes evident as each routed edge alters the free space available for the next. Also, placement of seemingly unrelated and unconnected nodes in the graph can often alter routes created for other edges, even while using space driven search as shown in Figure 2.10. Unless the user understands the underlying routing algorithms, there is hidden information that governs the behaviour of the routing. This breaks the principles of predictability, synthesizability and consistency.

OmniGraffle and Dia provide connector routing when attached objects are moved, though these routes do not observe objects in the diagram and may overlap them. They both allow the user to attach connectors to the shape or to specific connection ports, though this does not result in any special routing to accommodate the selected port other than some simple improvements to orthogonal connectors. Both allow additionally user-specified bend-points, and OmniGraffle possesses an option to draw connectors so they visibly "hop" other connectors where there are crossings.

We know of no editor which ensures that connectors are optimally routed in any meaningful sense and updates them in a consistent manner appropriate for interactive applications. Chapter 5 describes a technique and implementation to achieve this goal.

2.4.6 Clusters

It is common in graphs and many types of diagrams to group several objects together within a parent object as a cluster. Often we would like to be able to display the cluster in a collapsed form without the child objects visible or in an expanded form with the children visible. The diagram editor should be able to recompute the boundary of the cluster as the positions of its children are altered.

yEd has rectangular cluster groups, that can be collapsed and expanded. When there are connectors running from an object within a cluster to an object outside and the cluster is collapsed, the connectors will appear to come directly from the cluster object. Moving

objects in the cluster resizes the cluster (rectangular). Holding down shift, objects can be added or removed from the cluster.

ConceptDraw MINDMAP allows branches of the mindmap to be collapsed or expanded via clicking a small icon positioned next to topics and subtopics.

Most other editors, such as Visio, ConceptDraw, Dia, and OmniGraffle, do not support clusters. They all allow multiple objects to be grouped so that they may be manipulated as a single object. ConceptDraw supports assigning objects to a layer (an overlay) that may be hidden. Some functionality of clusters can be approximated through these features, but true support for clusters is lacking.

2.4.7 Automatic network layout

Again, with automatic network layout, there is a large amount of literature discussing techniques for graph layout, both static and dynamic. This work is examined later, in Section 6.2. Although there is much research into automatic graph drawing methods, these techniques have not found widespread use within popular diagramming tools. Where they have, they are used like single-effect placement tools; they just change the layout of the graph once and then need to be repeatedly reused. We will demonstrate this below, while examining the features of the automatic network layout tools that are available in widely-used diagramming software.

Visio has a once-off "Lay Out Shapes" option that can beautify a selection or current page with either a Radial, Circular, Flowchart/Tree, Compact Tree or Hierarchy layout. There are various options that can also be selected by the user to control interaction with grid lines and connector styles, and direction for layout and spacing. Layout is based on the structure of the graph as determined by connectors, but then in the resulting layout connector paths are not taken into account and this often leads to connectors overlapping shapes. Strangely, Visio's layout style and connector style choices are completely independent, and this can be problematic where certain routing styles clash with particular layout styles.

Visio has additional support for specialist diagram types with their own layout options. When a new organisational chart or brainstorming diagram is created the user is given access to an additional context-specific toolbar and set of tools for layout, style alteration of the entire diagram, and some tools for exploration, such as collapsing and expanding hierarchies. It does some correctness checking and only allows particular actions to be completed when the right objects are selected. Its layout is a once-off operation, with a re-layout button. Collapsing hierarchies is limited to hiding and showing subordinates (branches) in the organisational chart (tree), for example.

The Tom Sawyer Layout Assistant is an add-on for Visio 2000–2003 that adds an extra menu and toolbar with additional diagram layout options. Each of these layout styles (circular, hierarchical, orthogonal, symmetric and tree layout) is a once-off layout adjustment activated via a menu option or toolbar button. When one is selected, the add-on reads shape positions and connector information and then updates the diagram with the new layout. There is a global option called "Incremental Layout", which attempts to maintain

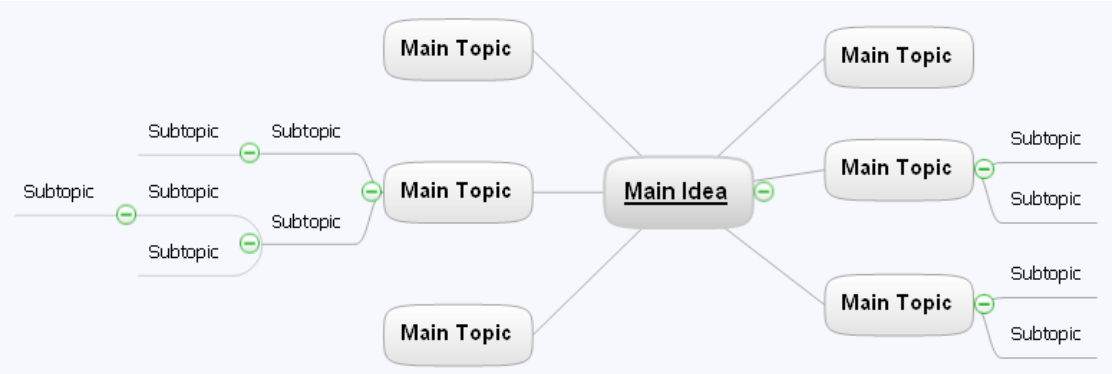


Figure 2.11: Radial layout of a mindmap in ConceptDraw MINDMAP. Branches can be collapses or expanded by clicking the small plus or minus icon next to nodes.

the relative position of diagram objects over successive iterations of the layout algorithms. They state that this helps preserve the user's mental map if minor modifications are made to the diagram and the layout is then rerun. This preference can also be used to preserve relative geographic positions if coordinates are set for diagram objects. It will also likely make the produced layout more predictable to the user.

Unfortunately, the assistant's layout tools have to be manually rerun after any changes to the diagram. Moreover, there is no way to have the Layout Assistant arrange only a portion of the diagram, meaning the user cannot manually position some shapes and have the add-on position the rest. Since this is an action that the user may desire which cannot be achieved, it breaks the principle of task conformance.

The assistant performs connector routing as part of the layout process. To allow this, Visio's automatic routing needs to be disabled. Sadly, when this is done, any manual re-routing or any new connector creation the user performs results in obviously incorrect connector paths where the connector is not even drawn to the correct endpoint. This behaviour is unfortunate since it breaks most of the principles of learnability, causing the user to lose trust in the system and become frustrated. After manually triggering a new layout, the modified/new connector is noticed and an appropriate route is set for it. The type of routing is specific to each type of layout, but the routes avoid crossing shapes. All layout styles have some options that adjust their parameters and allow added flexibility, for example, direction for hierarchical layout.

ConceptDraw has no automatic network layout tools. ConceptDraw MINDMAP,[15] a specialised version of ConceptDraw for creating mindmaps, has some basic layout tools. It has once-off radial layout as well as hierarchical layout in the four major compass directions. Radial layout is essentially just left and right hierarchical layout; see Figure 2.11 for an example of this layout. The layout needs to be manually rerun after the user repositions any nodes, though ConceptDraw MINDMAP automatically moves nodes around to avoid overlap if the user drops a node partially over another node.

[15]ConceptDraw MINDMAP 5.1.2.0, Computer Systems Odessa, `http://www.conceptdraw.com/mindmap/`

OmniGraffle has both hierarchical and force-directed layout of the whole diagram or selection. Hierarchical layout can be done in the four primary compass directions, and has options to preserve object ordering or minimise crossings. The force-directed layout has options to alter the line tension, line length, and shape repulsion. The graph layout is a once-off operation run by clicking a "Lay Out Now" button, or can be set to trigger when the structure of the diagram changes, or after any connection change.

yEd has hierarchical, force-directed ("organic"), orthogonal and circular layout. By default its layout starts from random node positions each time, meaning the layout can result in vastly different node positions each time it is run. Both the hierarchical and organic layout have options that are used to respect initial starting positions. Whenever objects are added to the diagram or the structure of the graph changes, the layout needs to be rerun. Since yEd is an interactive demonstration of the features of the underlying yFiles libraries, there are many options controlling the finer behaviour of algorithms, a great proportion of which are beyond the average user's comprehension. While the layouts that it produces can be very good, there is no way to constrain objects (to not be moved in future layouts or to stay aligned) or interact with the layout other than via parameters of the layout algorithm. This means the user who is unhappy with the layout must change some input option to the algorithm, rerun it, and hope they make the layout better, rather than worse.

While the popular diagram authoring software we have surveyed offer various types of automatic network layout, all of them conduct just a single layout. Just as with the once-off placement tools, this results in a highly viscous system where the user will have to rerun the layout algorithm after making any changes to the diagram. There are no major diagram editors that persistently preserve layout properties during diagram creation and manipulation. This continuous adjustment of the layout is offered by some interactive graph browsing software, but these tend to "take over" responsibility for the diagram layout and deprive the user of creative control that is important in an authoring context. Chapter 6 presents an approach to provide continuous network layout for diagram editors.

2.4.8 Non-overlap of shapes

None of the commercial diagram editors enforce non-overlap of shapes, except as part of their automatic network layout algorithms.

In Visio, if a dropped shape would overlap with other shapes, these other shapes are moved aside. It seems to move these shapes to the closest edge of the dropped shape, but it is not always predictable. Further, it seems to matter what type the shape is—it ignores some shapes.

2.4.9 Immediate feedback

Immediate feedback is essentially the concept of having direct manipulation in the presence of constraint-based relationships. These relationships might be an alignment or distribution, a certain type of object-avoiding connector, or some other constraint entirely.

Regardless, the user should be able to interact with the system in real-time, have the constraints satisfied and see the results immediately.

Some editors, like Visio, provide "live dynamics" in some simple situations, but for most cases the result of the action is not visible until the action has been completed, for example, shape outlines are shown while dragging with the mouse and the shape is actually moved when the mouse is released.

ConceptDraw shows live feedback while guidelines are being dragged around, attached objects are moving and connectors attached to those objects are rerouted, as long as their "auto route connectors" options are enabled.

Use of immediate feedback is explored for all of the constraint-based tools described in this thesis, and is an important aspect of their usability.

2.5 Conclusions

We have seen that while constraint-based graphics tools have been fairly widely explored in research fields, this technology has not yet made the jump into popular diagramming software. Available constraint-solving technologies and techniques should provide users with more powerful and useful layout tools. However, this is still largely unverified—a likely contributor to low adoption of these tools.

In the case of placement tools, it would appear that existing tools suffer from several usability problems. Tools based instead on multi-way constraints would seem to avoid those problems, but require careful evaluation of their own. We believe that multi-way constraints should replace one-way constraints for placement tools in diagram editors.

To evaluate this claim properly requires several steps. Firstly, some preliminary usability evaluation of existing one-way and proposed multi-way constraint-based tools is necessary. This will compare the two classes of tools and determine whether tools based on multi-way constraints offer any added benefit to the user. We expect this will be the case. Assuming it is, placement tools based on multi-way constraints can be developed via an iterative user-centred design process. This will involve evaluation of their interface and behaviour via usability experiments to identify and correct possible user difficulties in the use of the tools. This will ultimately yield a usable and useful set of persistent placement tools for diagram editors or identify why this is not possible.

Similarly, for other diagram constraints such as connector routing and automatic network layout, there is again much research experience that has not been utilised for integration into existing open-source and commercial diagram editors. With connector routing, some work is required to determine how existing line routing techniques can be adapted to be usable in highly interactive scenarios where features such as diagram aesthetics need to be taken into account, and certain traditional assumptions (like non-overlapping obstacles) cannot be enforced. Again, usability must be a primary concern and focus, and the underlying algorithms hidden from the user by predictable and understandable behaviour.

For automatic network layout the question is how existing layout techniques can be better integrated into interactive diagramming software, so that they become a unified part of the editor, rather than an extension that merely takes the current structure of

the diagram and returns a completely new layout for the page. The goal is to make layout requirements (such as ideal lengths of connectors, or downward pointing connectors) persistent constraints that are enforced usefully through further editing, while the user still has the power to override or alter when desired. The unexplored questions are related to the desired interaction model, user interface, behaviour and feedback of such tools so that they are usable and useful.

In each of these cases—placement tools, connector routing and automatic network layout—the goal is to relieve the user of tasks they would otherwise have to perform repeatedly to maintain the style or appearance of the diagram. This work can be offloaded onto the system, with the user still able to exert direction and control over the automated processes—whether this be specifying placement relationships, routing preferences or layout styles.

The following chapters will explore all these ideas in detail.

Chapter 3

Alignment and distribution

3.1 Introduction

Despite the large amount of research in the area of constraints and graphical editors, most mainstream, commercially available diagram editors only provide tools that perform once-off placement, and those editors that do provide constraint-based placement tools, such as Visio and ConceptDraw, use one-way constraints rather than multi-way constraints.

We believe there are three main reasons why tools based on other, potentially more powerful constraint solving techniques such as multi-way constraints, have not made their way into commercial graphical editors other than CAD applications. First, one-way constraint solvers are simple to write and extremely efficient. An efficient multi-way constraint solver is more complex to write and may require considerable numerical programming expertise. However, with the development of efficient algorithms for solving multi-way constraints and open source implementations of these algorithms this reason has become less important. Second, there is little or no evidence of their value. Graphical editors are unlikely to provide multi-way constraint-based tools without compelling evidence that they are going to be useful to users. Third, previous multi-way constraint-based prototype systems have proposed or suggested user interfaces for constraints, but these have varied greatly and often have perceived problems, or have never been exposed to real user testing.

There have been almost no usability studies which investigate the value of the various constraint-based systems that have been presented. In particular, there has been no investigation of the general claims that multi-way constraint-based tools are better than one-way constraint-based tools, for example, see (Hill, 1993; Sannella et al., 1993). For this reason, we conducted an experiment comparing the usability of one-way and multi-way constraint-based alignment and distribution tools.

In our study we designed and implemented a set of multi-way placement tools as an add-on for Microsoft Visio 2002. Visio already provides tools which allow users to set up persistent alignment and distribution relationships that are implemented using one-way constraints. An extensions framework is provided, which allowed us to plug a multi-way constraint solver into Visio. Using this platform we conducted a usability study to compare the usefulness of once-off tools, one-way constraint-based tools and multi-way constraint-based tools.

(a) (b)

Figure 3.1: Effect on layout due to one-way left-alignment of shapes A and C with shape B. A guideline is created in left-alignment with shape B, and shapes A and C are moved to align with the guideline. All three shapes become attached to the guideline.

This study showed some statistical significance and general trends favouring multi-way tools above one-way constraint-based tools and once-off placement tools. It particularly showed severe usability issues with the one-way placement tools. On further examination, we felt that some of these issues might be a result of the user interface provided by Visio rather than intrinsic limitations of one-way constraints.

Section 3.2 provides motivation for the research by introducing and discussing the limitations of existing once-off and one-way constraint-based alignment and distribution tools. Section 3.3 describes the design of the multi-way constraint-based tools, implemented as an add-on for Visio. Section 3.4 discusses the experimental design, procedure and results of the usability study.

3.2 Background

The shortfalls and limitations of once-off and one-way constraint-based placement tools in existing diagram editors were outlined in Section 2.4. This section explains the motivation for the study by describing the usability issues of one-way constraint-based tools in more detail.

3.2.1 One-way alignment and distribution

In addition to once-off placement tools, Visio and ConceptDraw provide a persistent form of alignment through the use of guidelines. Purely placement aids, guidelines act like normal manipulable objects on the page but are not part of the final diagram and they will not be visible on printed versions of the diagram. Visio also provides a guideline-based persistent distribution tool.

One-way constraint-based alignment tools work by creating a guideline connected to the lead object in the selection. The tools then adjust the positions of all the other selected objects to bring them in line and glue them to the guideline, as shown in Figure 3.1.

Often "snap-dragging" is provided as a means of attaching shapes to existing guidelines. Snap-dragging (Bier and Stone, 1986) is a technique which uses a gravity metaphor where, as the user drags a shape, it will snap and connect to significant objects such as guidelines.

Once shapes have been glued to a guideline, they can be moved by moving the guideline. Unfortunately, one-way constraints only allow us to specify that the shape is constrained to align with the guideline. They do not specify that the guideline is constrained to align with the shape. As a result, moving shapes directly (rather than via guidelines) will always overwrite their position formulae which are used to describe the constraint-based relationships. For example, the shape s might be centre aligned to guideline g using the position formula $s.x = g.x$. If it is subsequently moved to an x position of 40, its position formula will now be $s.x = 40$ and the shape will be unglued from the guidelines. Since one-way constraints are fragile, this will tend to result in a viscous system.

The alignment and distribution tools themselves are required to move shapes to set up relationships. This breaks shapes from their prior alignment or distribution relationships.

Compounding this problem, there is no visible indication, at least in Visio or ConceptDraw, that a shape is glued to a guideline unless that shape is currently selected. This means that since the shape has not necessarily moved away from the guideline, the constraint might be broken without any visual feedback to the user. Such behaviour is a prime example of hidden dependencies, and means the user will be unable to fully understand the state of the diagram from its on-screen representation.

These problems are illustrated in Figure 3.2 where two alignment relationships are set up, both involving shape B, a vertical alignment in Figure 3.2(a), followed by a horizontal alignment in Figure 3.2(b). In creating the second relationship, shape B's position is altered to be dependent on the position of the horizontal guideline—an action that invisibly removes the shape from the vertical alignment relationship. Moving the most recently created alignment in Figure 3.2(c) works as expected. Then in Figure 3.2(d), manipulating the older alignment relationship, we see that shape B is no longer constrained to follow the guideline. This behaviour is undesirable, since it makes it hard for the user to predict the response of the system. Obviously relationships will behave in different ways depending on the order in which they were set up, and interacting with them in the "correct" way requires premature commitment.

This particular example clearly illustrates problems with shapes being invisibly broken from guidelines. We believe the major weakness of one-way constraint-based placement tools is that relationships can be broken by direct manipulation or by any other tools that affect the positions of shapes.

In Visio, shapes will only be attached to their most recently established placement relationship (and guideline), except when they have been explicitly placed in both a vertical and horizontal relationship through snap-dragging. It should be noted that in the example presented in Figure 3.2, the action that caused shape B to be broken from the vertical alignment—the creation of the horizontal alignment—does not actually require the first constraint to be broken. Using one-way constraints it is possible to have both the x and y position of a shape constrained to follow different guidelines. This particular behaviour

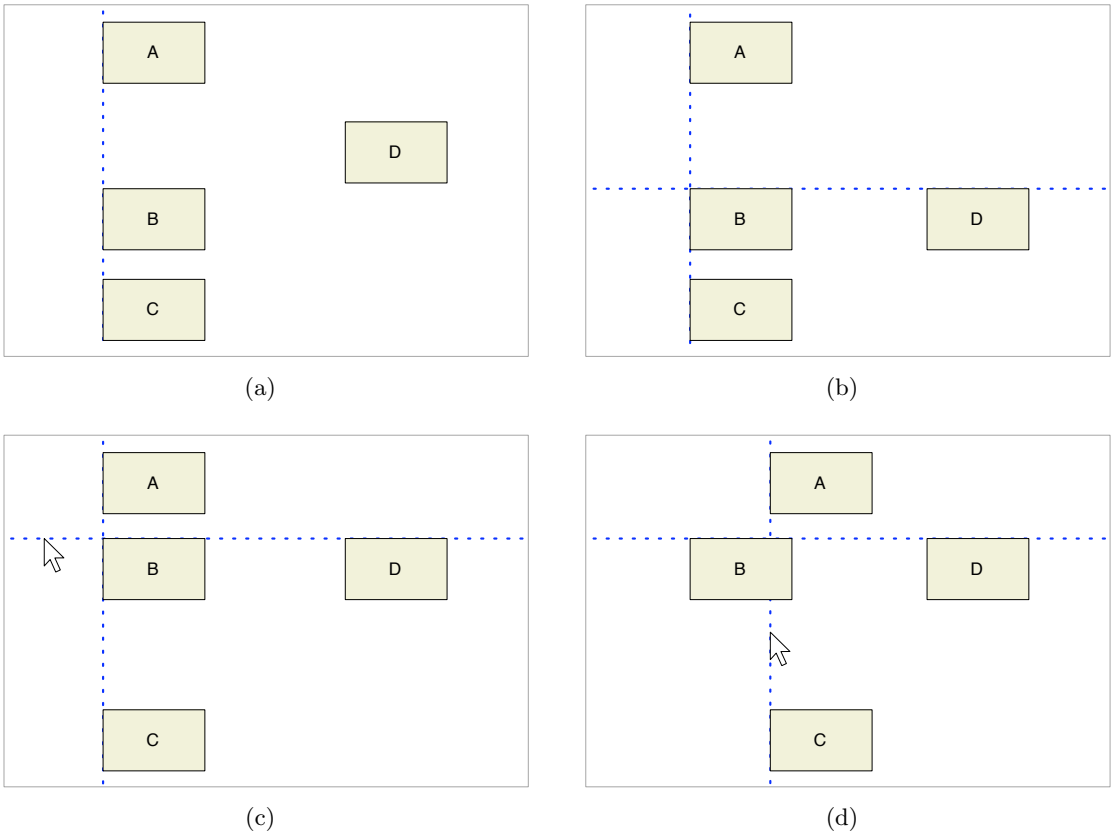
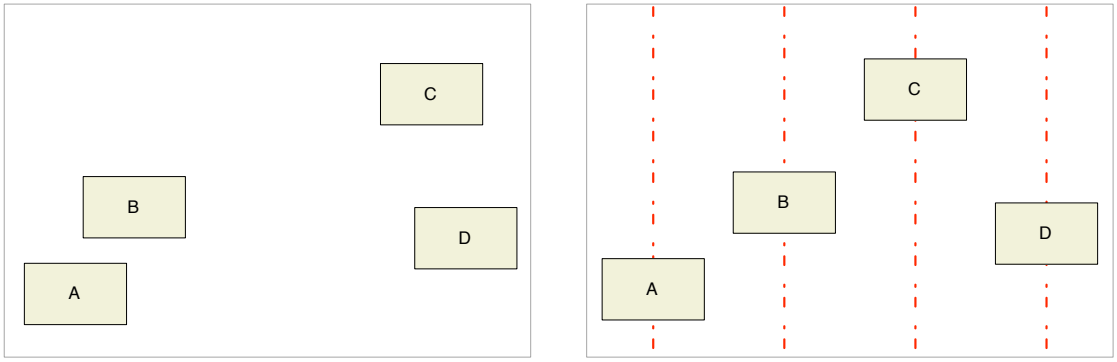


Figure 3.2: Example of unexpected constraint breaking in Visio's one-way constraint-based alignment tools. Initially, in (a), the three shapes A, B, and C are left-aligned. Shape D is then top-aligned with shape B in (b). Next, in (c), the user drags the horizontal guideline up, moving shapes B and D along with it. When the vertical guideline is moved right (d), the user discovers that shape B is no longer attached to the vertical guideline.



(a) (b)

Figure 3.3: Effect on layout due to one-way horizontal distribution of all shapes by their centre. Guidelines are created for all shapes, and the shapes are attached to these. The guidelines (and shapes along with them) are spaced equally between the outermost two.

appears to be a bad design choice in Visio. Regardless, constraint breakage will still be unavoidable in many cases due to formulae being overwritten in one-way tools. For example, vertically aligning an object that is already vertically aligned.

Like alignment, Visio's persistent distribution tools are implemented using guidelines. The distribution tool creates a guideline for each selected shape and glues the shape to it. It then takes the two outermost guidelines and distributes the other guidelines (and attached shapes) equally between these, as shown in Figure 3.3.

As a result of using the tool, we end up with a persistent relationship that can be further manipulated. The outermost guideline on each end of the distribution can be dragged, effectively resizing the entire distribution. The other guidelines in the distribution cannot be dragged, because they are dependent on the positions of the outermost guidelines.

This behaviour is sufficient for the basic case of distributing shapes, but we again run into the limitations of one-way constraints when trying to distribute shapes involved in alignment relationships. Unless we explicitly select the guidelines themselves for distribution, the tool acts upon and moves the individual shapes, effectively ignoring (and removing them from) any alignment relationships they are already part of. In Visio, the user can only distribute aligned groups of shapes by their guidelines if they wish to preserve existing alignments. This behaviour violates the principle of substitutivity, which suggests that systems should allow the user to arbitrarily substitute equivalent values for each other. It also violates the usability concept of familiarity or closeness of mapping, that is, that a user wishing to distribute shapes A, B and C, should be able to do so by selecting these shapes, making the interface closer to real world manipulation.

We can see that from the user's perspective that one-way constraint-based placement tools have a serious drawback: alignment and distribution relationships can break due to manipulation of objects involved in the relationship or because more than one constraint is applied to the same object. While some issues are introduced by specific implementations of the tools, the bigger problem is inherent in one-way constraints—each constraint has a fixed direction and an attribute can only have a single formula associated with it.

We therefore hypothesize that placement tools would be more usable if they provided truly persistent alignment and distribution relationships—two shapes put into an alignment relationship should stay aligned through all further editing until the relationship is explicitly removed. The tools could then accurately use the metaphor of an alignment relationship, without the user needing to think about them as shapes glued to guidelines. As one-way constraints cannot support this, we must consider tools that are based on multi-way constraints.

3.3 Visio and multi-way constraint-based placement tools

We chose Microsoft Visio Professional 2002 as the platform for this usability study since it provides support for developer plug-ins and in particular allowed us to extend it with multi-way constraint-based alignment and distribution. Most commercial diagram editors do not provide support for developer plug-ins. Visio was chosen over the alternative of modifying an open source editor (such as Xfig or Dia) for three reasons. Firstly, it is widely used in industry, which makes the outcome of the research relevant and interesting to a greater group of people. Secondly, it is heavily customizable and provides support for writing add-ons that can neatly extend Visio's own tools and features. Thirdly, being an Office application it shares the common Microsoft Office interface, meaning it will already be partially familiar to anyone who has experience with Office products. The relatively wide exposure of Office applications meant that by using Visio for the development and accompanying study, we were less likely to confound the measurement of the tools' usefulness with interface usability issues.

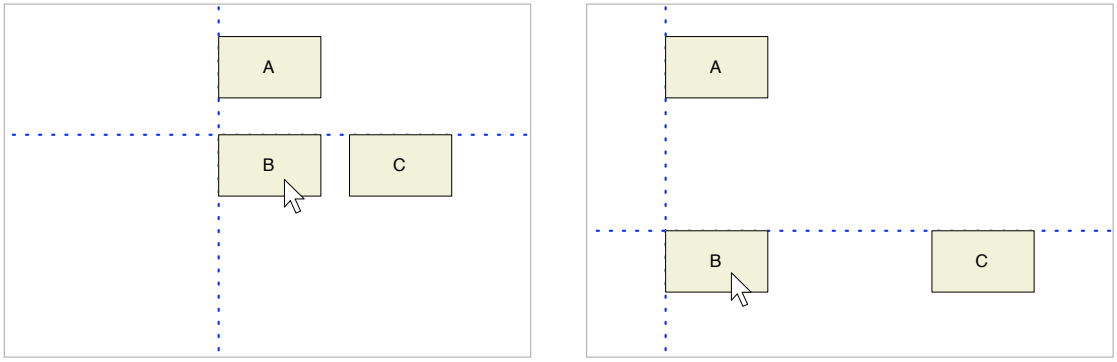
3.3.1 Software tool architecture

We first describe the multi-way constraint-based alignment and distribution tools that we integrated with Visio. Our tools were written in C++ and compiled as a Visio add-on Dynamic Link Library (DLL) with Microsoft Visual C++ 6.0.

Our implementation of multi-way tools made use of a multi-way constraint solving toolkit, QOCA (Marriott and Chok, 2002). QOCA allows us to create and solve systems of multi-way linear equality constraints. Multi-way constraints provide the ability to set up the initial alignment relationship so that moving the guideline moves the group of shapes attached to it, and moving any or all of the shapes also moves the entire group (including the guideline) where this still satisfies any other active constraints—the aligned group will stay aligned throughout all further editing.

QOCA is based on the *metric space model*, in which constraint addition and deletion are treated differently to editing the value of a variable. Variables can be editing by "suggesting" new values for them. This results in a new solution that is closest to the new suggestions, the previous assignment, and in which all active constraints are satisfied.

Internally, QOCA represents constraint variables as two floating point numbers: a current value and a desired value. It also associates with each variable a stay weight and an edit weight. Stay weights express the importance of keeping the value of a variable



(a) (b)

Figure 3.4: Effect on layout due to moving shape B, which is involved in two multi-way alignment relationships, both down and to the left. As a result, the vertical guideline and shape A are moved left, and the horizontal guideline and shape C are moved down.

unchanged if it is not being edited. Edit weights express the importance of changing a variable to its desired value if it is being edited.

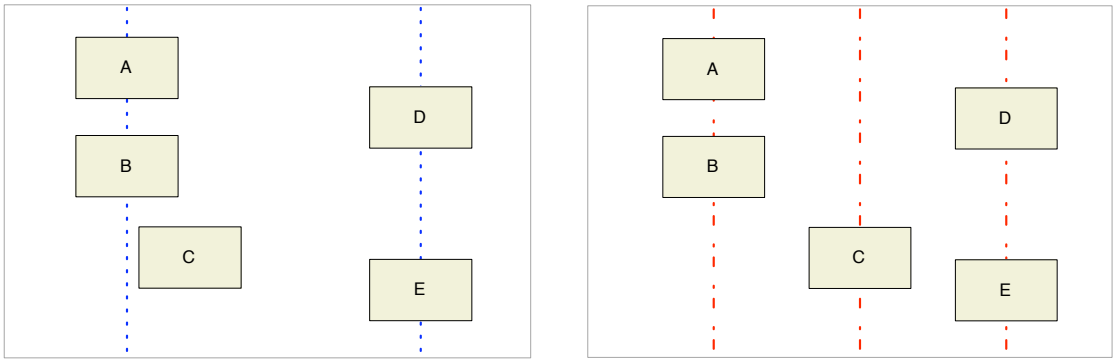
In implementing alignment and distribution tools we use QOCA's linear equality solver (`QCLinEqSolver`), which transforms the original linear arithmetic constraints and keeps them in a tableau in solved form. When solving, it uses as a metric the square of the Euclidean distance between each variable's current and desired value.

We represent the (centre) positions of shapes and guidelines as solver variables. They are given a stay weight of zero, an edit weight of 100, and are marked as "nomadic" (`qcFloat_Nomadic`) so that they try to keep the same value as when they were last solved. We also represent the dimensions of shapes as solver variables. We decided that alignment and distribution should only affect placement and not alter the dimensions of shapes. Thus we give these variables a very high stay weight of 1000000 and mark them as "manual goals" `qcFloat_ManualGoal`, causing the variable's goal value to only be set via an explicit `SuggestValue` call. While we never want the solver to adjust the values of shape dimensions we still need them to be variables in the solver rather than constants since some constraints—such as aligning the right side of a shape—depend on them and they may change when a user manually resizes a shape.

We chose QOCA over an incremental, local propagation based, multi-way constraint toolkit such as DeltaBlue (Freeman-Benson et al., 1990) since it is able to handle cycles of constraints whereas DeltaBlue can not.

Alignment

The creation of an alignment relationship works the same way as existing tools in Visio—a guideline (if one does not exist) is created and aligned with the lead object, and all other shapes in the selection are moved to align with the guideline. It is only during subsequent manipulation of the diagram that the difference between the multi-way and one-way versions of the tools becomes evident.



(a) (b)

Figure 3.5: Effect on layout due to multi-way horizontal distribution of all shapes by their centre. Shapes A and B as well as shapes D and E are already centre aligned. As a result, they are treated as a column and their existing guidelines are used for the distribution.

Internally, an equality constraint is created between each shape s and the guideline g. For example, vertical centre alignment would result in constraints of the form

$$s.x = g.x$$

and vertical alignment by the left sides of shapes would result in constraints of the form

$$(s.x - s.hw) = g.x$$

Here, hw stands for "half width"—we store shape dimensions as half the width and height to simplify the constraint equations. The multi-way nature of the created relationship is clearly visible in Figure 3.4 where, when shape B is moved down and to the left, the two alignment relationships cause both shape A and C to move.

An alignment relationship can be removed by deleting the visible indicator of the relationship—the alignment guideline. A shape can effectively be added to an alignment relationship by aligning it with any (or every) shape in the existing relationship. This can be achieved by snap-dragging a shape on to a guideline and via the Align Shapes dialog box.

Distribution

The initial effect of the distribution tool is again similar to existing tools—it considers all the shapes in the current selection, spacing them all equally (by their left, right or centre) between the two outermost shapes. The difference is that the user can distribute shapes involved in alignment relationships and those relationships will stay active. When the user applies this tool, any group of aligned shapes will remain aligned, appearing to be treated as a single object for the purpose of distribution. This behaviour is shown in Figure 3.5, where all shapes have been selected and distributed horizontally by their centre. Internally, a variable s is used to represent the separation between two adjacent guidelines ($g1$ and $g2$) in the distribution. An equality constraint is created between each pair of

adjacent guidelines to constrain the separation between them to match the distribution's separation:

$$(g1 - g2) = s$$

Selected shapes without associated guidelines have new guidelines created for them and these guidelines are the subject of the actual constraints controlling the distribution relationship. Distribution guidelines are given a different colour to distinguish between pure alignment guidelines and those involved in a distribution. The change in colour is indicated in Figure 3.5 using a variation to the line stroke, where the alignment guidelines in Figure 3.5(a) change when they become part of the distribution in Figure 3.5(b).

Once a distribution relationship has been set up, dragging an outer guideline (either by direct manipulation, or movement of a shape "attached" to one) has the effect of growing or shrinking the entire distribution. Moving the centre guideline (or guidelines) of the distribution has the effect of moving the entire set of objects involved in the distribution. Moving any other inner guidelines partially resizes as well as moves the distribution. Since this behaviour might not be completely predictable we would have liked to disallow manual movement of inner guidelines as Visio does, but this was unfortunately not possible with our add-on.

A distribution relationship can be removed by selecting and deleting all the distributed guidelines. When the user deletes just a single guideline involved in a distribution, the distribution relationship is removed. The other involved guidelines remain, but become (or revert to being) plain alignment guidelines, changing colour to indicate this.

It is possible to place the same shape in several different types of alignment, such as being left-aligned with some shapes while right-aligned with others. The alignment guidelines can themselves be part of distributions. In this way it is possible for the user to set up complex systems of constraints. Since alignment and distribution are described as "placement" tools it was decided that shapes should never be resized to force constraints to be satisfied. When a shape would need to be resized in this way, or constraints have no solution due to a new conflicting constraint, the last action is undone and a dialog box is presented to the user explaining that their action created a conflict and could not be accomplished.

3.4 Study 1: One-way versus multi-way

Using our Visio add-on, we compared the usability of once-off, one-way and multi-way constraint-based alignment and distribution tools in the first of our usability studies.[1]

[1]This study received ethics approval from the Monash Standing Committee on Ethics in Research Involving Humans (SCERH) as project 2002/036.

3.4.1 Method

Design

The placement tools should aid the user in creating and modifying diagrams quickly. We were interested firstly in different classes of tools' effectiveness, efficiency, and suitability for the task of diagram creation and manipulation, thus used task completion times and diagram correctness as our primary usability metrics. We didn't test the learnability of the tools since it was felt this was a less important metric—future studies could address this. User satisfaction was gauged, but informally.

The study consisted of a set of exercises in which the participants were asked to create, modify and manipulate diagrams resembling simple flowcharts. Flowcharts were chosen because they are a reasonably well-known and simple notation (at least for our participants) that capture the characteristics of other kinds of network-like diagrams.

The focus of the exercises was shape placement and overall diagram layout. We wanted the exercises to be simple, require no prior knowledge of flowcharts, though not so simplistic that they seemed contrived. To this end, the diagrams given in the exercises were realistic flowcharts and the layout changes requested in the exercises were presented as aesthetic improvements to the diagram.

Participants in the study were randomly assigned to one of three groups. Each group was provided with a different set of constraint-based tools for alignment and distribution. The three groups were:

- **Group OO—once-off**: Once-off alignment and distribution tools were available. These move the involved shapes but do not create lasting relationships.
- **Group OW—one-way**: Visio's native form of persistent alignment and distribution tools based on one-way constraints were available.
- **Group MW—multi-way**: Persistent alignment and distribution tools based on multi-way constraints were available.

All participants were given exactly the same exercises, but could use only the set of tools offered to their group. Each group was trained on the particular set of tools available to them. In all other respects the training was identical for all groups.

It was hypothesized that the persistent state of the relationships set up by the tools in Group OW would make them faster and less error-prone than the once-off Group OO tools. Likewise, we hypothesized that the multi-way nature of tools in Group MW would make them faster and less error-prone than the one-way constraint tools of Group OW.

Participants

Thirty people were tested; ten in each of the three groups. There were no requirements for participants other than they be computer-literate adults. All participants were undergraduate university students who were native speakers and readers of English, with normal or corrected-to-normal vision. Participants were not reused across groups.

Equipment

All tests were carried out in private, the investigator performing the experiment with participants singly. The environment was a usability lab in which the participant sat at a computer while the investigator sat behind them, observing and taking notes.

A record of each participant's interaction with Visio during the tests was obtained by taping a video feed of the test computer's display to VHS cassette. A small amount of audio data from post-test debriefing and discussion was also captured to the tape. In addition to this, the start and finish time for each exercise was taken down by the investigator. This included the time taken to comprehend the instructions. Notes taken by the investigator summarized the strategy and method taken by the user to carry out the task, as well as problems they experienced.

Short pre- and post-test surveys were used as a means of obtaining additional qualitative and quantitative data about participants' prior experience with related tools, how difficult they found the exercise and suggestions they had for the software's improvement.

At the beginning of each experiment the participant was shown a 15 minute training video. This consisted of a common introduction to Visio, as well as a specific introduction to the tool set they would be using. Following this, the participant was asked to carry out some training tasks in an informal environment where the investigator would answer questions related to the software. The last of these tasks required the participant to construct a specific 16 shape diagram from scratch. When the participant had completed these tasks and was comfortable with Visio and its tools, they proceeded to the timed exercises.

Materials

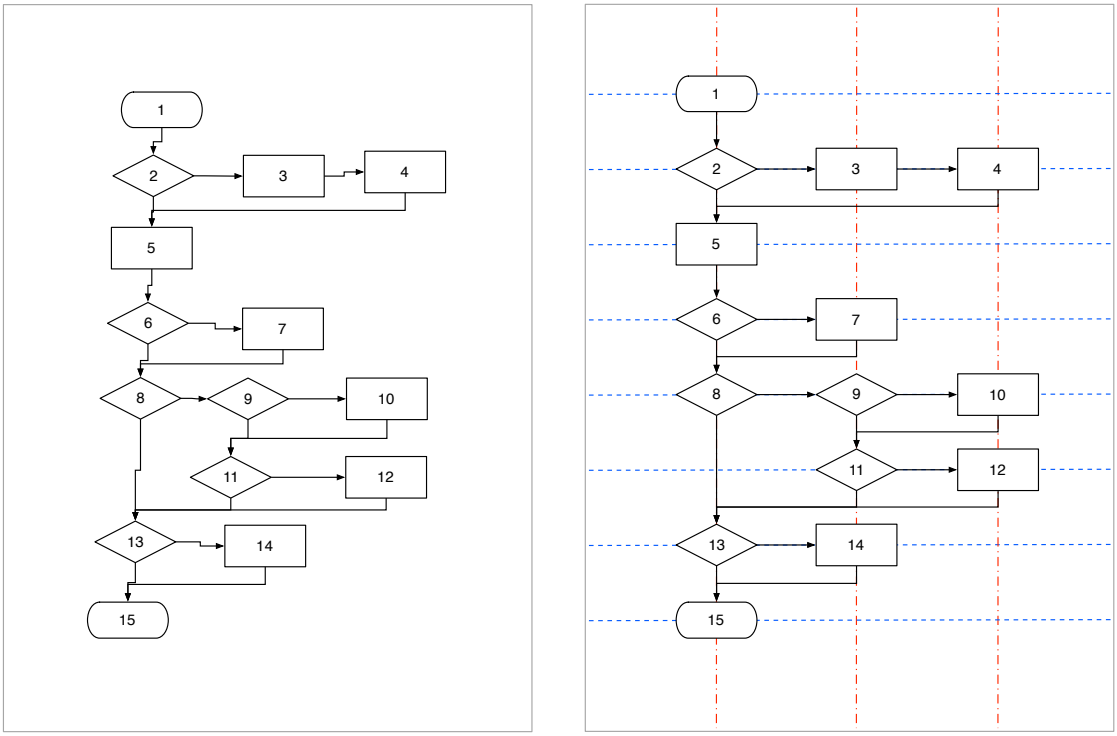
In the exercises, participants were required to modify some simple flowcharts. The exercises required the participant to make layout changes to the diagrams—spacing or aligning objects on the page to make the diagram more aesthetically pleasing. Some of the instructions and final diagrams showed a generic representation of alignment or distribution relationships. In this case the participant was required to enforce these relationships. They were also free to make use of additional placement relationships if they felt this would make the task quicker or easier.

The exercises were done one at a time, in fixed order. For each exercise, the participant was given a three-page instructional handout.[2] The first page showed a typed description of the task written in point form in plain English. The second page showed the initial diagram, and the third showed the target diagram (i.e. the result of applying the specified instructions to the initial diagram).

The five timed exercises were:

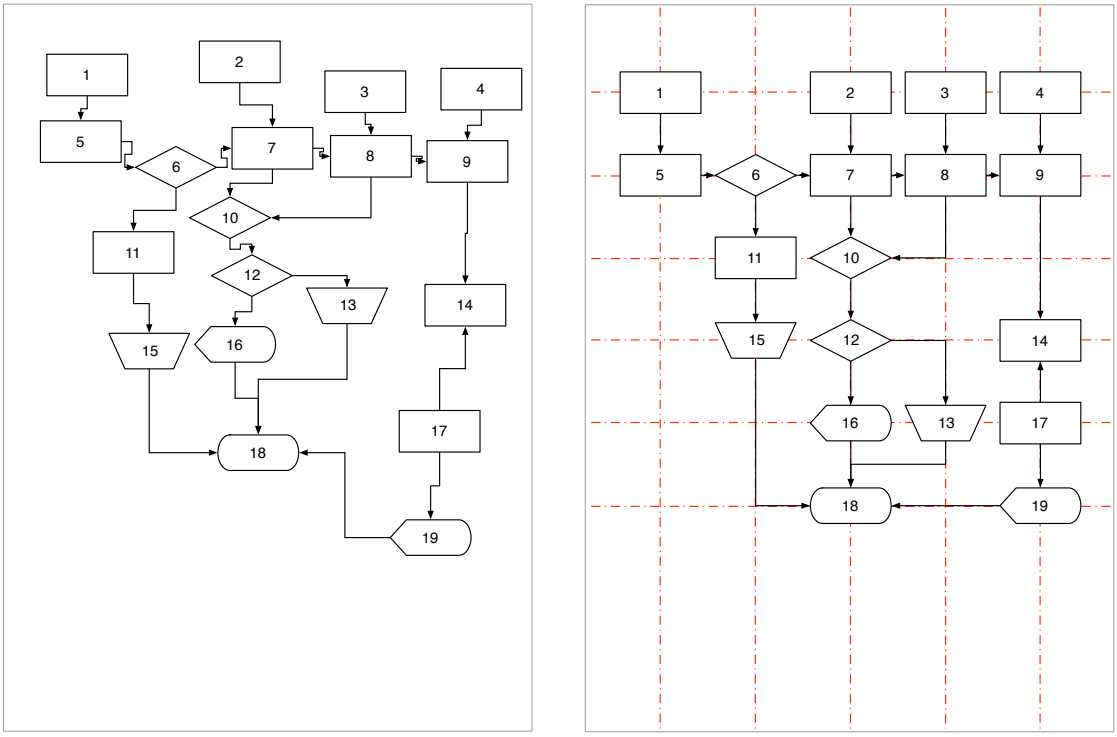
- **"Editing"**: A simple exercise intended to increase familiarity with the editor and the available tools. Participants were required to make changes to a diagram resembling

[2]Complete copies of exercises, instructions and surveys for this user study are available online: `http://www.csse.monash.edu.au/~mwybrow/wybrow-thesis-exp1-materials.tar.gz`



(a) Initial diagram (b) Target diagram

Figure 3.6: Diagrams for the "Manipulation 1" exercise. The exercise requires participants to create several alignment relationships and a single distribution.



(a) Initial diagram (b) Target diagram

Figure 3.7: Diagrams for the "Grid" exercise. The exercise requires participants to set up vertical and horizontal alignments as well as both vertical and horizontal distributions, creating a grid-like arrangement of shapes, making use of the available page space.

the one constructed during the training. They had to add two shapes to the diagram, reroute several connectors, and ensure two pairs of shapes were centre-aligned. Since the exercise was quite short and mostly a general editing task, it was not expected that performance on this task would show significant difference between the groups.

- **"Choice"**: Another editing task, beginning afresh with the training exercise diagram. Participants were required to remove three shapes, repair several connectors, and rearrange the diagram to make use of the entire page. Placement relationships were not explicitly mentioned in the instructions but could be inferred from the target diagram given. This was another short exercise, giving the participant more experience with general editing and further chances to use the placement tools.

- **"Manipulation 1" and "Manipulation 2"**: These two exercises were designed as a pair. The first exercise required the participant to add some alignment relationships and a single distribution to a pre-constructed diagram. The initial and target diagrams for this exercise are shown in Figure 3.6.

  The second exercise required the diagram to be resized to take up all of the available page. In this exercise no modification to the diagram was required apart from moving the objects in it. The required alignment and distribution relationships remain unchanged.

- **"Grid"**: The fifth and final exercise required modifications to another pre-constructed diagram. The participant was required to set up horizontal and vertical alignments and both horizontal and vertical distributions. The final diagram shows a grid-like arrangement of shapes which uses most of the available space on the page. The initial and target diagrams for this exercise are shown in Figure 3.7.

3.4.2 Results

We consider completion times for the exercises, as well as errors in the completed diagrams. For the analysis we use well-known statistical techniques (Snodgrass, Levy-Berger and Haydon, 1985). To determine overall statistical significance we use a one-way randomized Analysis of Variance (ANOVA), where we consider $p < 0.05$ to be statistically significant. In the case of unequal group variances, as determined by Levene's test, the comparison of differences between means is instead achieved with a one-way ANOVA using the General Linear Model (GLM) in Minitab.[3] As there has been no prior empirical analysis in this area, we are concerned where among the groups significant differences (if any) lie. For this reason we use Tukey's HSD test, a form of post hoc comparison, with p set at 0.05.

In our analysis we have excluded the results of exercises wherever the participant did not finish that exercise. It is interesting to note that in total five people did not finish all of the exercises, four from Group OW, one from Group MW, none from Group OO. The only exercises that were not always completed by participants were "Manipulation" and "Grid".

[3]Minitab, Minitab Inc., `http://www.minitab.com/`

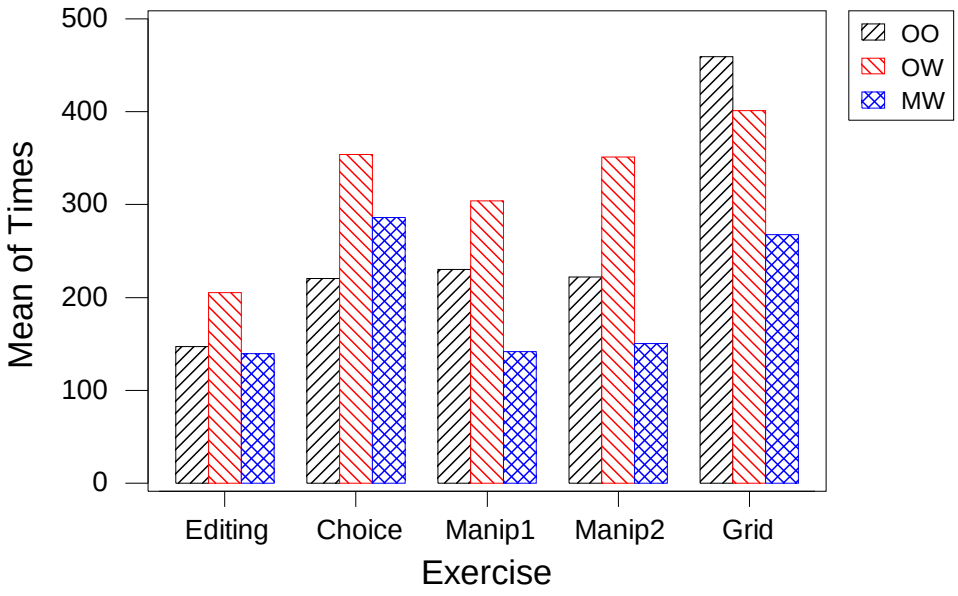


Figure 3.8: Mean completion times (in secs.) for each exercise.

Completion times

The average completion times for each exercise are shown in Figure 3.8. To determine where the statistical significance lies we perform an ANOVA for each exercise.

A one-way ANOVA shows borderline significance for the first two exercises. The first exercise ("Editing", $F = 3.48$, $p = 0.046$) was the basic editing requiring only optional use of the alignment or distribution tools. Further analysis, by applying Tukey's HSD test, reveals there to be no significant difference between groups in times for the first exercise.

The second exercise ("Choice", unequal group variances, $F = 3.52$, $p = 0.045$) involves optional use of the tools. Tukey's test does reveal a significant difference between the times for Group OO and Group OW. This difference may be explained by participants in Group OW who chose to experiment with the use of the tools during this exercise, increasing their completion times. Placement tools would not be expected to have an effect on basic editing (excluding shape placement), it is unsurprising that more statistically significant results were not seen in these exercises.

We do find there is significant difference in the completion times for the exercise "Manipulation 1" (unequal group variances, $F = 7.19$, $p = 0.004$). Times for this exercise are summarized in Figure 3.9. Figure 3.9 is a standard box-plot, showing a measure of spread. The boxes in the figure show the range of the middle 50% of the data, while the whiskers stretch to the largest and smallest values that are not "outliers". Outliers, those points more than 1.5 times outside the range of the middle 50%, are marked with a '*'. The mean completion time and standard distribution (in brackets) for each group are given below the box-plot.

To see exactly where the significance lies we use Tukey's HSD test to consider all pairwise differences between group means. Using this method we find that the only significant difference is between Group OW and Group MW. In this exercise the multi-way

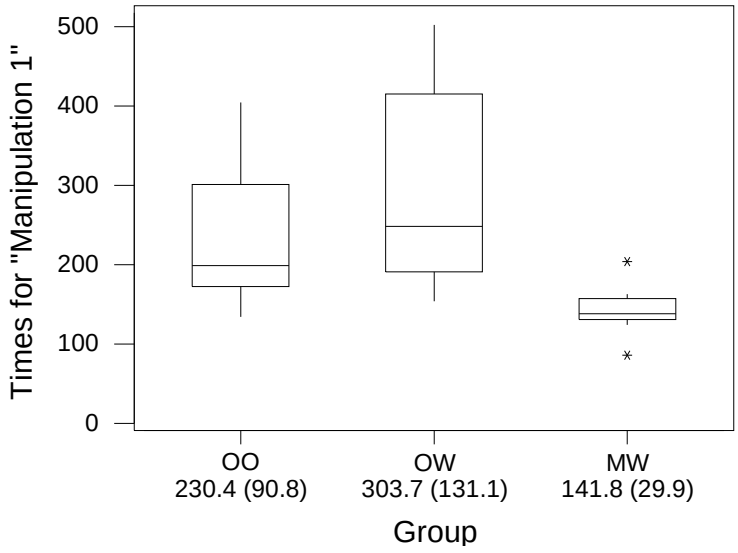


Figure 3.9: Box-plot of completion times for "Manipulation 1" exercise.

constraint-based tools of Group MW offer significant benefit over the one-way constraints of Group OW.

We again determine there is significance in the completion times for the exercise "Manipulation 2" ($F = 5.61, p = 0.010$). Times for this exercise are summarized in Figure 3.10. Once again, using Tukey's HSD test, we find that the only significant difference is between Group OW and Group MW. This exercise required that participants manipulate relationships they had set up in the previous exercise. We see that Group MW also benefits over Group OW in this aspect of editing.

In the study we made several observations that might explain why Group OW offered no significant benefit over Group OO for the "Manipulation" exercises. Participants in Group OO had to reuse the tools repeatedly to keep objects in the desired relationships. Group OW participants tended to have to do the same. Some shapes stayed in relationships, but a large number became unglued, leading not only to disassociated shapes but also to disassociated guidelines that no longer carried any meaning. Such objects cluttered the page and their manipulation tended to be misleading and confusing for participants—these objects were expressing a role that was no longer correct. In fact, some participants found it easier to delete such guidelines and continually recreate the relationships, effectively mimicking the usage of the once-off tools.

The final exercise ("Grid") also showed significant difference in completion times (unequal group variances, $F = 7.01$, $p = 0.004$). Times for this exercise are summarized in Figure 3.11. Tukey's HSD test showed that there was significance between times for Group OO and Group MW, and also between times for Group OW and Group MW. This supports that the multi-way constraints of Group MW are more beneficial for construction of heavily aligned and spaced diagrams than the once-off Group OO and one-way Group OW tools.

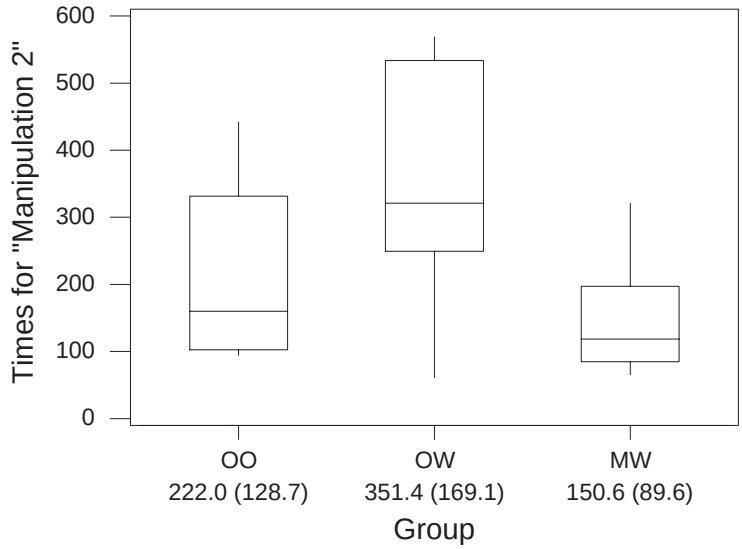


Figure 3.10: Box-plot of completion times for "Manipulation 2" exercise.

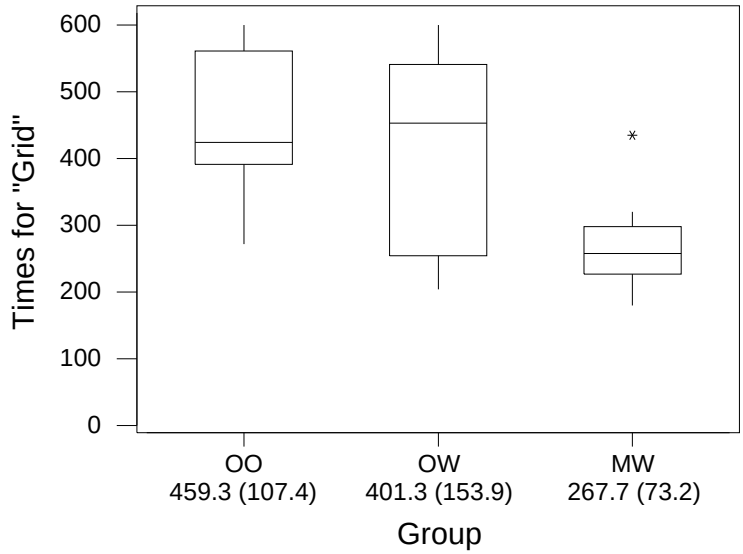


Figure 3.11: Box-plot of completion times for "Grid" exercise.

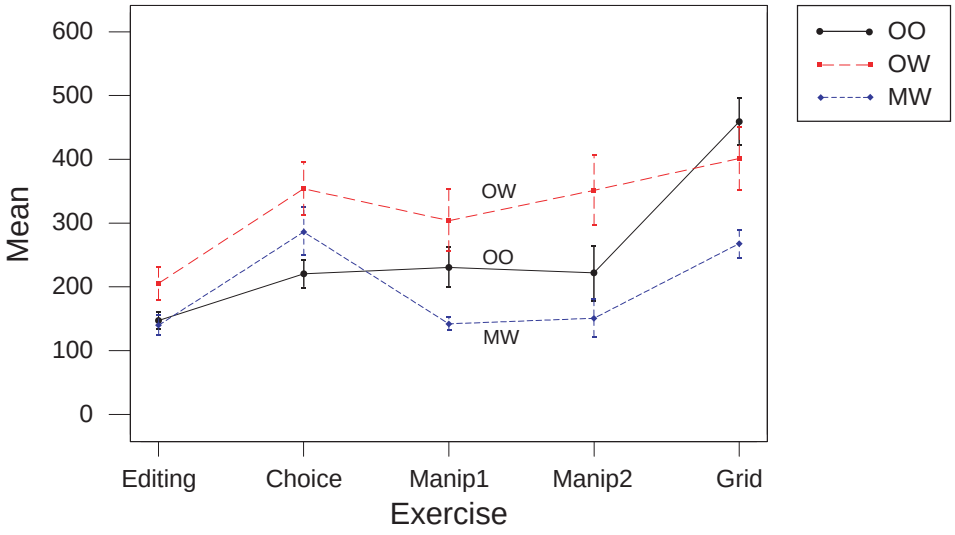


Figure 3.12: Interaction plot for Groups OO, OW and MW.

We next determine whether there was any interference between the groups and the exercises, that is, whether differences seen between groups were due only to the tasks carried out for a particular exercise. Figure 3.12 shows group means as an interaction plot with error bars. An absence of interaction is illustrated by the relatively parallel lines of Group OW and Group MW for the final three exercises. This suggests that where we have seen significance, it is not due to the benefit of the tools for the particular individual exercises, but rather it is a benefit seen across all tasks.

Perhaps the most interesting result is the interaction between Group OO and Group OW. The plot shows that while Group OO out-performs Group OW (by means) on most exercises, the result is reversed for the final exercise. Since the Group OW tools are a persistent form of the Group OO tools, we had expected Group OW to out-perform Group OO across the tests. We found no significant evidence to support this. In fact, the time values in Figure 3.12 suggest that Group OW tools provide worse performance on all but the final exercise. This supports the observation that these tools suffer from usability problems. We were surprised by the extent of these problems and their impact on the tools' usefulness in terms of diagram editing time.

The "Grid" exercise is the only exercise that appears to show any kind of positive difference between Group OW and Group OO. This may be explained by the fact that the exercise does not involve any manipulation of relationships once created. We also observed that by this exercise many participants had learned the quirks of the one-way constraint-based tools and devised a particular order in which they could use the tools that would minimize the breaking of placement relationships.

Error rates

We also collected information about the number of errors present in participants' final diagram for each exercise. Diagrams were compared by eye to the target diagram. We

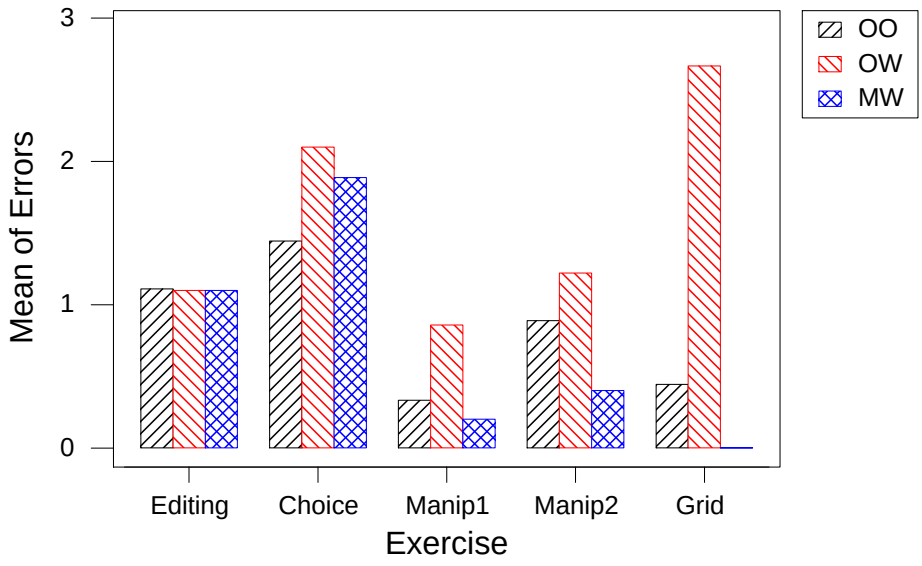


Figure 3.13: Mean errors in final diagrams for each exercise.

classified as errors failure to carry out particular task instructions, as well as shapes not part of required alignment or distribution relationships—easily determined by the presence of kinked connectors.

The raw averages for errors in the final diagrams are shown in Figure 3.13. Apart from Group MW having significantly less errors than Group OW in "Grid" ($F = 5.79$, $p = 0.009$), these results were not statistically significant. Though by looking at the graph we can see that Group MW mostly leads to less errors than Group OO and Group OW. Here again, in the exercises requiring real use of placement tools, we see that the one-way constraint-based tools of Group OW are again more detrimental to performance than their simple once-off Group OO counterparts.

Participant feedback

Some interesting qualitative results come from participant feedback provided by the post-test questionnaire. In addition to general comments or suggestions, participants were specifically asked the following two questions:

> "Could you please describe any parts of the exercises you found particularly difficult?"

> "Based on the exercises you were asked to perform, if you could change or add anything to Visio what would it be?"

In general, comments supported the hypothesis that the multi-way constraint-based version of the alignment and distribution tools were more usable than either the once-off or the one-way constraint-based versions.

The thrust of the comments from participants in the once-off group was that they wanted the tools to create permanent relationships. That is, they wanted constraint-based tools. Seven out of nine participants stated they had problems with the distribution tool not operating on alignments. A typical comment was that "aligning and distributing objects requires too much manual work"—this shows the system is too viscous. Additionally, they expressed difficulty with knowing "the correct order ... so I don't have to redo steps", which is indicative of the premature commitment of the tools. As a result, eight participants suggested adding some form of "better" distribution that "doesn't affect alignments", maybe using a kind of "sticky" alignment, or providing some way to "lock" groups of shapes or alignments. Interestingly, during the exercises many participants expected the tools to create permanent relationships, even though the training was very specific in stating this wasn't the case. This expectation echoed in their comments and demonstrates the tools lacked task migratability. While the user wants to, they are unable to hand off execution for this task to the system.

Feedback from the participants who had used the one-way constraint-based tools described the expected usability issues. Seven out of ten participants stated they had problems with shapes becoming unglued from guidelines, "shapes breaking out". Two other participants reported general difficulties with alignment and distribution, one blaming the clarity of instructions and the other feeling personally at fault for the problems—specifically in not finding the "correct" order in which to apply the tools. These comments again show the one-way tools to require premature commitment and thus require extra work from the user, leading to a viscous system.

A final participant reported the activity of manually attaching nodes to guidelines as "tedious". Three participants stated difficulties with distributions acting on all objects rather than alignment groups. Their suggestions were to "have shapes always stay on guides", or change Visio so that it "remembers all alignments, and if you want to detach them you can do so manually" and that "alignment relationships be respected when applying new distributions and when moving the object as opposed to the guideline". In other words they wanted the one-way constraint-based tools to behave like the multi-way constraint-based version.

Very few participants using the multi-way constraint-based placement tools had problems with the tools or expressed suggestions for their improvement. Two participants had difficulties with not being able to distribute guidelines directly rather than just shapes. This was basically a design oversight. As we described earlier, our distribution constraints actually operate on guidelines rather than shapes themselves so it would have been easy to implement this behaviour. It is worth noting that the error message generated in this case ("The distribution tool must be applied to 3 or more shapes.") made participants change their strategy for distributions and caused them no further problems.

One participant using the multi-way tool encountered a problem where they couldn't both left-align and centre-align two objects of apparently equal width that actually had very slightly different widths. This is essentially a reporting problem: the error message only tells them that the action they were attempting would break an existing placement

relationship. It would be much more useful to be able to let them know the set of existing constraints that were preventing the action. Three participants suggested the ability to lock shapes at a particular position once they were happy with their layout.

Four participants using the multi-way constraint-based placement tools and four participants using the one-way tools reported problems with clutter from guidelines obscuring the underlying diagram. This supports previous claims that clutter of on-screen constraint indicators could be problematic for users (Heydon and Nelson, 1994; Gleicher and Witkin, 1994). A frequently suggested solution from users was to use a different kind of visual indicator or a tool to easily hide guidelines. Another comment was that it was difficult to determine whether a guideline indicated an alignment or a distribution relationship—effectively a problem of visibility, where the distribution relationship is not being clearly shown. These problems suggest that more research is needed in order to find a visual representation for placement relationships that will scale up to large diagrams without cluttering them.

3.5 Conclusions

We have described a usability study examining the relative usefulness of different kinds of constraint-based placement tools for the general task of constructing and editing diagrams. We examined once-off tools present in most diagram editors along with one-way constraint-based tools offered by several leading editors and multi-way constraint-based tools of our own design.

Our results support our hypothesis that placement tools based on one-way constraints have usability issues and that multi-way constraint-based placement tools offer significant benefit over one-way constraint-based tools for tasks requiring the alignment and distribution of shapes.

Interestingly, our results show persistent placement tools based on one-way constraints offer no significant advantage over the simple, once-off tools offered by nearly all diagram editors. The one-way tools can be thought of as an extension of the once-off tools, yet our results suggest that they provide no added value to the user for general editing and layout tasks. In fact, it appears that one-way constraint-based tools mostly lead to slower times and more errors in the finished diagram than once-off tools.

Multi-way constraint-based tools were not found to offer a statistically significant advantage over once-off tools in all tasks, though in tasks requiring alignment and distribution of shapes they consistently resulted in faster average completion times and fewer errors in the final diagram. Given this, we believe that significance would be seen given further testing.

However, as discussed in Section 3.1 we feel that some of the usability issues for the one-way constraint-based placement tools may be a result of the user interface provided by Visio, rather than an intrinsic limitation of one-way constraints. The first issue (as exemplified in Section 3.2.1) is that Visio essentially only allows an object to be in a single alignment/distribution constraint. However, since the horizontal and vertical position of standard graphic objects are independent there is no inherent reason why an object cannot

have a one-way constraint on its horizontal position and another one-way constraint on its vertical position.

The second issue is the degree of feedback provided during direct manipulation. Visio only provides delayed feedback during direct manipulation, meaning that if the user moves a guideline with shapes attached these shapes are only moved when the user completes the action by releasing the mouse. Ideally, we would like to provide the user with the option of immediate feedback while working with such tools.

Unfortunately these are both limitations of Visio itself—behaviour that can not be modified via an add-on. A more thorough comparison of one-way and multi-way tools is still necessary. This will require the use of a different diagram editor, as we shall see in Chapter 4.

Chapter 4

Dunnart and revised placement tools

4.1 Introduction

As stated in Chapter 3, there were two main issues that we felt might decrease the usability of the Visio implementation of the one-way constraint-based placement tools. The first issue is that Visio effectively only allows an object to be in a single alignment/distribution constraint. However since the horizontal and vertical position of standard graphic objects are independent of each other, there is no inherent reason why an object cannot have a one-way constraint on its horizontal position and another one-way constraint on its vertical position.

The second issue is the degree of feedback provided during direct manipulation. Like most graphic editors Visio allows the user to drag an object or collection of selected objects to a new position, providing feedback by showing an outline of the selected objects as they follow the cursor. However, the position of other objects may indirectly depend upon the position of the selected objects because of constraints between them. When moving shapes in Visio, the position of these other objects are not updated until the user has completed the action. So if the user moves a guideline with shapes attached then these shapes are only moved once the user releases the mouse. We call this *delayed feedback*.

Some interactive constraint-based graphical systems provide what we call *immediate feedback* in which the user sees all changes to the diagram as they happen. This includes showing how unselected shapes move when they are connected by constraints to the selected shapes. It has been suggested that this behaviour is preferable since it allows users to understand a system of constraints more easily as they observe the diagram change from one state to another as a result of their actions (Gleicher and Witkin, 1994; Ryall et al., 1997). Similarly, it should allow users to notice mistakes more quickly. An example of this is that when dragging guidelines, the user would immediately see if they were dragging a different set of shapes than the ones they believed were attached to the guideline.

Since we found the Visio add-on interface too inflexible for our needs, we created a new editor, Dunnart.[1] This was written to provide basic diagram editing features, as well as revised versions of the one-way and multi-way placement tools without the previous limitations. Dunnart additionally allowed us to compare the effect of immediate and delayed feedback on the usability of both the one-way and multi-way tools. Dunnart has formed the basis for other usability studies in this thesis.

We felt a follow-up usability study was required to validate the results of the first study by removing confounding factors due to the design of Visio's one-way constraint-based placement tools. Secondly, we aimed to evaluate immediate feedback for both one-way and multi-way constraint-based placement tools.

This study showed there to be a statistically significant difference between the times taken to complete a range of diagramming tasks for multi-way compared with one-way based placement tools. It was found that participants using multi-way tools completed tasks significantly faster than their one-way counterparts. Somewhat surprisingly, the study did not show significant difference between the immediate feedback and the delayed feedback groups.

This study also uncovered two specific usability issues with these improved multi-way tools. The first of these was the common complaint of clutter within the diagram. This was caused primarily by the extra on-screen objects representing the placement relationships. Participants complained that they obscured details of the diagrams. The second issue was users misunderstanding complex sets of constraints and being surprised at the way in which constrained objects moved.

As a result, we revised the multi-way tools once more in an attempt to address these usability concerns. We allowed the guidelines to have reduced visibility, to glow, or to fade over time. We also added an Information Mode which the user could use to query how two shapes were attached to each other via the layout constraints.

A third usability study was then conducted to evaluate these changes. It found that the visibility controls, specifically fading, provided an appropriate solution for the clutter problem. Information Mode proved to be a beneficial addition, but not the complete answer to the comprehension issue.

Section 4.2 presents our new diagram editor Dunnart and the improved versions of the one-way and multi-way placement tools. Section 4.3 describes experimental design, procedure and the results of the usability study comparing one-way and multi-way tools as well as delayed versus immediate feedback. Section 4.4 explains our revisions to Dunnart to address the clutter and comprehension problems uncovered in the study. Section 4.5 details the design and results of the subsequent user study, examining the placement indicator visibility options and Information Mode.

[1] Dunnart, Michael Wybrow, `http://www.csse.monash.edu.au/~mwybrow/dunnart/`

4.2 Dunnart

For this study we decided to write our own diagram editor, rather than modifying Visio. We had found that Visio's interface was fairly limited. Changing the behaviour and on-screen representation of Visio's one-way constraint-based placement tools would have required us to reimplement these tools as plug-ins. More importantly, it seemed impossible to modify Visio's behaviour to provide immediate feedback since any add-on can only ever act in response to an event and Visio does not post events during dragging, only posting a "shape move" event when the user has completed the entire drag and drop action.

For this reason we required an editor that was built from the beginning with constraint solving and immediate feedback in mind. Rather than trying to write this on top of an existing code base that may prove to be unsuitable, we chose to write a simplified diagram editor to be used exclusively for the usability testing. The interface and available features were kept to a minimum, reducing the possibility of participants experimenting with or being confused by extraneous menu options or unnecessary tools. Having written Dunnart, we were able to instrument it in ways useful for testing. We are able to replay a participant's actions from within Dunnart, watching the mouse move around the screen like a video replay, freeing us from recording experiments with traditional cameras or screen capture. We could also collect statistics about the type and number of actions that participants used to complete the tasks.

4.2.1 Software tool design

In this section we describe the slightly revised one-way and multi-way constraint-based alignment and distribution tools that we provided in Dunnart, our new diagram editor.

Dunnart itself is written in C++ and compiles and runs on Windows, Linux and Mac OS X. It has a simple interface and uses a custom Graphical User Interface (GUI) toolkit written by the author. The widgets in this GUI are similar in style to those of applications running under Windows 2000, see Figure 4.1. Dunnart allows all the standard interaction you would expect from a diagram editor; you can add shapes to the page, move, resize and label them. You can cut, copy and paste selections, undo and redo. Dynamic connectors are available that will reroute themselves as a result of manipulation of the shapes to which they are attached.[2]

Alignment

The creation of an alignment relationship, accessed through the "Align Shapes" toolbar button and corresponding dialog box, results in a guideline (if one does not exist) being created and aligned with the lead object. All other shapes in the selection move to align with and become attached to the guideline.

[2] In this early version of Dunnart, only the positions of the two shapes to which a connector attaches were used for determining connector routes. Later versions of Dunnart perform intelligent object-avoiding connector routing, as discussed in Chapter 5.
Latter versions of Dunnart also include support for different kinds of shape and connector types as well as standard formatting features, such as colour, arrowhead type and line type properties.

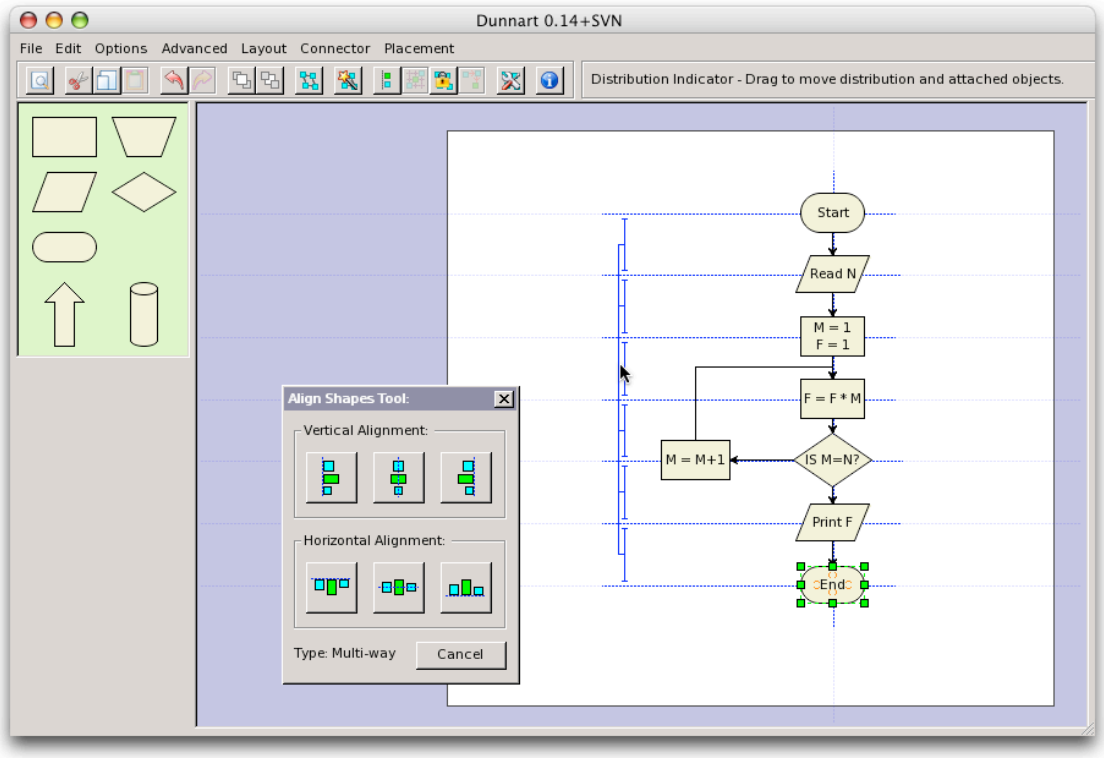


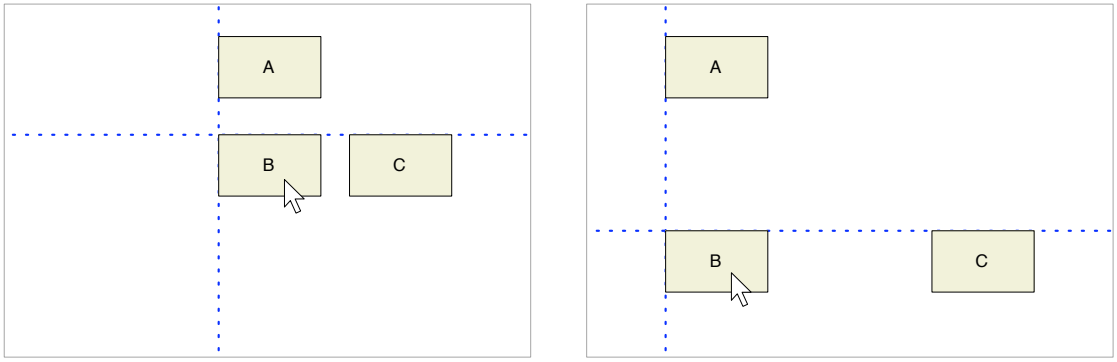
Figure 4.1: Dunnart, our constraint-based diagram editor

For the one-way tools the user can then move the guideline to move all the attached shapes. Since this is achieved with one-way constraints, any shape can be attached to at most one vertical guideline and one horizontal guideline. Vertically aligning a shape that is already aligned (with other shapes) by a separate edge will cause this formula to be overwritten and leave it attached to the more recent relationship's guideline. Likewise, if a shape is moved then its position formula will be overwritten, causing it to break from existing relationships. However, horizontally aligning a shape which is already vertically aligned works correctly.

In contrast, when shapes involved in a multi-way relationship are moved or resized the other shapes involved in the alignment (and the guideline itself) will also move. This is subtly different from grouping the objects in that a vertically aligned shape is able to slide up and down the guideline without vertically moving any of the other objects attached to the guideline, as shown in Figure 4.2.

We allow the user to add a shape to an existing guideline by dragging the shape over the guideline. The guideline will be highlighted red while the shape is roughly in alignment with it, as shown in Figure 4.3. If the user releases the mouse button, the shape will be attached to the guideline and added to the alignment relationship.

We also allow the user to free shapes from multi-way relationships. Rather than having to delete a guideline to remove a relationship, the user is able to enable "free-dragging" by holding ALT while they move a shape. This breaks the shape from any relationship it is part of, allowing it to be dragged free of all alignments. This is similar to Briar's



(a) (b)

Figure 4.2: Effect on layout due to moving shape B, which is involved in two multi-way alignment relationships, both down and to the left. Unlike if shapes A and B were grouped, a vertical alignment constraint causes shape A to move in the x dimension only.

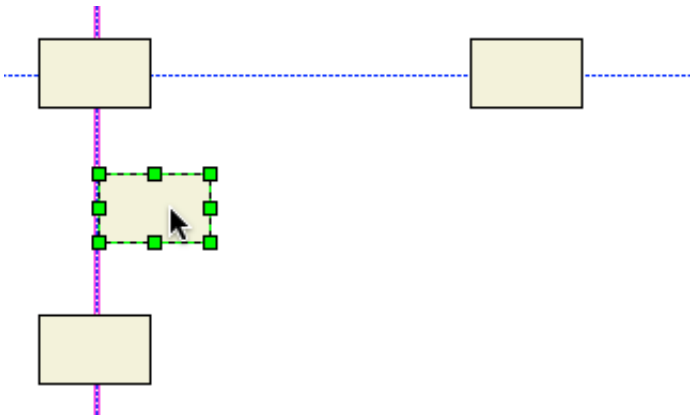
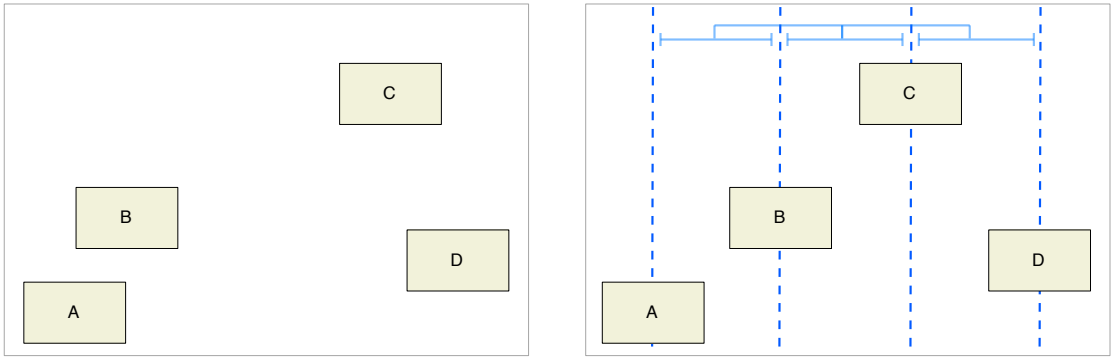


Figure 4.3: The user can add a shape into existing alignment relationship by dragging the shape over a guideline and then dropping it while the guideline is highlighted.



(a) (b)

Figure 4.4: Effect on layout due to multi-way distribution of all shapes. A distribution indicator object is created in addition to the four guidelines. This object represents the distribution relationship and may be used to interact with it.

constraint *ripping* feature (Gleicher and Witkin, 1994). While the ALT key is held the dragged shape will not try to attach itself to other guidelines.

Additionally, we allow shapes to be freed from individual alignment relationships via a shape's context menu; the menu will show an item for each relationship that a shape is part of. Clicking the menu item breaks the shape from that single relationship, leaving the others intact, while leaving the shape stationary.

When use of the alignment tool causes multiple guidelines to be aligned they are merged, rather than being discarded and left on the diagram. This avoids some of the problems we saw in the first experiment which led to many extra guidelines being dropped and subsequently littering the page.

Distribution

The new distribution tool has the same basic behaviour as other versions. The selected shapes are spaced equally between the outer two, and all shapes without associated guidelines will have guidelines created for them. Again, these guidelines will be the subject of the actual constraints controlling the distribution. Unlike the previous versions, guidelines (as well as shapes) can be selected directly for distribution.

To improve the visibility of distribution relationships, we have added a distribution indicator object as shown in Figure 4.4. This addresses the problem of users not being able to determine the difference between alignment and distribution guidelines when their only difference was colour. The indicator can also be used as a way of interacting with the distribution; it can be dragged to move the entire distribution and while it is selected it has a handle at each end that can be clicked and dragged to resize the distribution. Deleting the indicator causes the distribution relationship to be removed, leaving the guidelines intact.

Deleting a guideline involved in a distribution created with either of the new tools, whether one-way or multi-way, has the same effect: the guideline is removed from the distribution and the remaining guidelines' positions are recalculated so that they remain

equally distributed between the two outermost guidelines. If either of the outer guidelines is deleted, the distribution is not resized but continues with one less guideline (the next outermost guideline becomes the outer guideline). Once there are less than three guidelines remaining in the distribution, the distribution relationship will itself be removed, leaving only the guidelines.

For multi-way distributions, moving any distributed guideline or any shape moves the entire distribution, without resizing. This includes moving either of the outer guidelines. Distribution indicators are intended to be a physical on-screen representation of distribution relationships. For this reason, the size of distributions are controlled via handles on distribution indicators rather than through manipulation of the outermost guidelines.

In our original Visio add-on we attempted to specify the behaviour of distributions purely using the constraints and different combinations of stay and edit weights. This did not really give us enough control and we were not always able to get the desired behaviour. For example, we sometimes found that users were confused by the distributions both moving and resizing while they dragged guidelines. This is bad from a usability perspective—the user should not need to know that the system behaves the way it does because it is implemented with *technique xyz*. The implementation details of a system should never need to be known to understand its behaviour.

Thus, for revised versions of the tools we concentrated on getting consistent, predictable behaviour. Hence, distributions remain a fixed size if any of their guidelines are dragged, though they can be explicitly resized by dragging handles on the distribution indicator. By default, the guideline at the other end of the distribution will stay where it is. We now call this the locked guideline and highlight it in dark blue when the mouse cursor is over a distribution handle, see Figure 4.14. This way the user has an immediate indication—even before beginning the resize operation—that the distribution will be resized with that guideline locked in place. Furthermore, we have added the ability for the user to hit the TAB key to cycle the locked guideline through the possible choices. In this way they can resize the distribution around the centre guideline if they wish. Note: the locked guideline can be switched even while the resize operation is in progress. This gives added flexibility and control to power-users, but won't bother users simply resizing the distribution normally.

Dragging a guideline involved in a one-way distribution breaks it from that distribution, since its new position overwrites the guideline's position formula which was keeping it positioned relative to the distribution. The behaviour of the distribution in this case is the same as for a deleted guideline. In multi-way distribution relationships, the user can intentionally break a guideline from a distribution by holding the ALT key as a guideline is dragged.

If any user action results in unsatisfiable constraints, that action is automatically undone and a dialog box reports the conflict to the user.

All the tools are activated through a toolbar button and their options set with an associated dialog box containing buttons with icons representing the type of alignment

or distribution to be created. The placement tools ignore connectors for the purpose of alignment and distribution.

4.3 Study 2: Revised one-way vs. multi-way, feedback level

Our second usability experiment[3] was a more focused series of diagram manipulation/layout tasks in which the participants were given a diagram and asked to modify it in various ways.

4.3.1 Method

Design

The starting diagram for a task was always the diagram the participant had constructed in the preceding task. The tasks were designed so they maximized the use of the tools.

The initial exercise in the experiment required the user to take an existing diagram without any existing constraints and to set up new placement relationships. We did this so the user should know (especially in the case of the one-way group) they had constructed all of the relationships that they would later have to manipulate. This way, they wouldn't feel "tricked" or set up with deliberately difficult relationships.

In this study we measured the usefulness of the tools with the total exercise and individual component task completion times. We didn't consider errors since during this study we stressed the importance of constructing a correct diagram rather than finishing quickly. This was done primarily so that subsequent tasks would have the correct starting diagram.

As with the first study, we used basic flowcharts as our diagram type for all exercises. Unlike the first study, the flowcharts had meaning and defined a real process—coffee making. Participants were not required to understand the meaning of the flowcharts. Still, we felt using realistic diagrams made the modification exercises less contrived. To this end we gave scenarios—beautifying the diagram, fitting the existing diagram into a particular region of the page—describing the participants' motivation for making the modifications to the diagram.

Participants in the study were randomly assigned to one of four groups. Each group was provided with a different set of constraint-based tools for alignment and distribution. The four groups were:

- **Group OW/DF—one-way, delayed feedback**: One-way alignment and distribution tools available. Outlines showing the change in position of the selected objects are shown during dragging, but movement of objects connected by constraints is not shown until the mouse is released.

- **Group OW/IF—one-way, immediate feedback**: One-way alignment and distribution tools are available. Immediate feedback is shown during all interaction,

[3]This experiment received ethics approval from the Monash SCERH as project 2002/036.

including "live" changes to the actual objects being manipulated and the position of objects modified through constraints.

- **Group MW/DF—multi-way, delayed feedback**: Multi-way alignment and distribution tools are available. Outlines showing the change in position of the selected objects are shown during dragging, but movement of objects connected by constraints is not shown until the mouse is released.

- **Group MW/IF—multi-way, immediate feedback**: Multi-way alignment and distribution tools are available. Immediate feedback is shown during all interaction, including "live" changes to the actual objects being manipulated and the position of objects modified through constraints.

All participants were given the same exercises and could only access the particular tools offered to their group. Training differed slightly for each group so participants knew how to use the tools available to them.

It was hypothesized that the persistent state of the relationships set up by the multi-way tools in Group MW would make them more usable than the one-way Group OW tools. We also hypothesized that the immediate feedback (IF groups) would be more usable than delayed feedback (DF groups) regardless of constraint type, because the participants could see the result of their interaction immediately. It is important to note that constraint solving for both the one-way and multi-way tools in Dunnart is fast enough to allow responsive direct manipulation when working with diagrams of the size used during the study.

Participants

Thirty-two people were tested; eight in each of the four groups. There were no requirements for participants other than that they be computer-literate adults. All participants were university students who were native speakers and readers of English, with normal or corrected-to-normal vision. Participants were not reused across groups.

Equipment

All tests were carried out in private, the investigator testing participants singly. The environment for the experiment was a private office in which the participant sat at a computer while the investigator sat behind them, observing and taking notes.

Interactions made by each user during their session were recorded via the logging mechanism of Dunnart so that their actions could be played back for reference purposes. Also collected was a log file for each test containing the times and type of every action the participant made in completing the exercises. From this we were able to accurately retrieve the start and finish times (the time of the first and last "action") for each exercise. Other notes taken by the investigator summarized the strategy and method taken by the user to carry out the task, as well as problems they experienced. These were used to prompt discussion during the debriefing.

Short pre- and post-test surveys were again used as a means of obtaining some additional qualitative and quantitative data about participants' experience with related tools, how difficult they found the exercise and suggestions they had for improving the tools.

At the beginning of each experiment participants were taken through a twenty-five minute scripted training exercise, that introduced them to Dunnart (referred to as "the editor") and described its basic features while requiring them to interact and experiment with it. The placement tools were explained and their training required them to set up and interact with these relationships. This was conducted informally so that the participant was able to interrupt the training and ask for clarification on specific points or spend a little more time on any aspect of the training they felt needed extra attention. When participants completed the training, had no further questions and indicated that they were comfortable to proceed, they commenced the timed exercises.

Materials

In the exercises, participants were required to make layout changes to the diagrams—spacing the objects on the page or aligning them to make the diagram more aesthetically pleasing. The instructions and final diagrams showed alignment or distribution relationships, which we required participants to enforce in their final diagram.

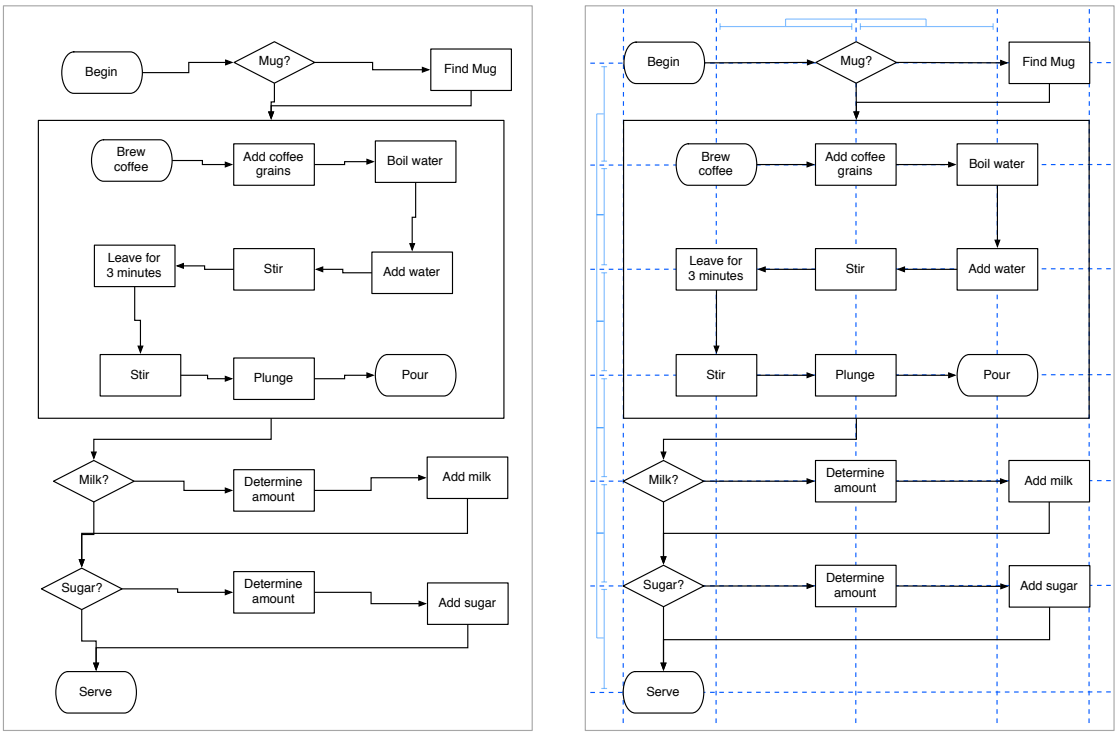
There were three component tasks to the exercise. Each task led on to the next and they were performed one at a time in fixed order. For the second and third component tasks the participant's own output from the previous task was used as their starting point. Emphasizing the importance of diagram correctness for the exercises helped minimize the effect this had on results. It was felt that participants should work with a diagram and set of constraints they had set up themselves.

For each task, participants were given a three-page instructional handout.[4] The first page showed a typed description of the task, including justification for the task and written instructions. The second page showed the target diagram, the result of applying the specified instructions to the initial diagram. The third page again showed the target diagram, this time in "print preview" mode, without any guidelines or distribution indicators visible. This was provided so that participants could check the paths of connectors and the overall structure of the diagram without the clutter of the alignment aids.

The three timed component tasks were:

- **"Beautify"**: This task required the participant to take an existing flowchart with no constraints and to add constraints to enforce multiple interconnected alignment and distribution relationships. It required little manipulation of constraints once they were created. The initial and target diagrams for this task are shown in Figure 4.5. The instructions were accompanied by the following motivational text: "Messiness in a diagram (such as kinks in connectors) confuses the reader and detracts from the message being conveyed by the diagram. For this reason our first task is to neaten the diagram by aligning and spacing objects."

[4] Complete copies of exercises, instructions and surveys for this user study are available online: `http://www.csse.monash.edu.au/~mwybrow/wybrow-thesis-exp2-materials.tar.gz`



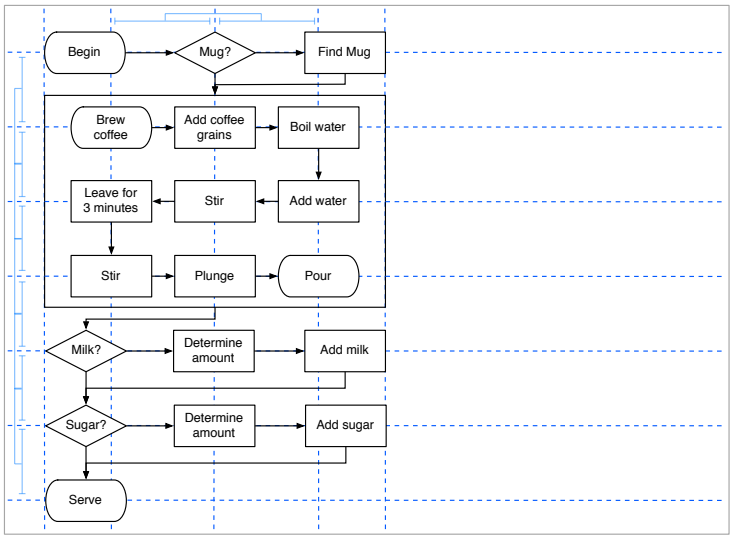
(a) Initial diagram

(b) Target diagram

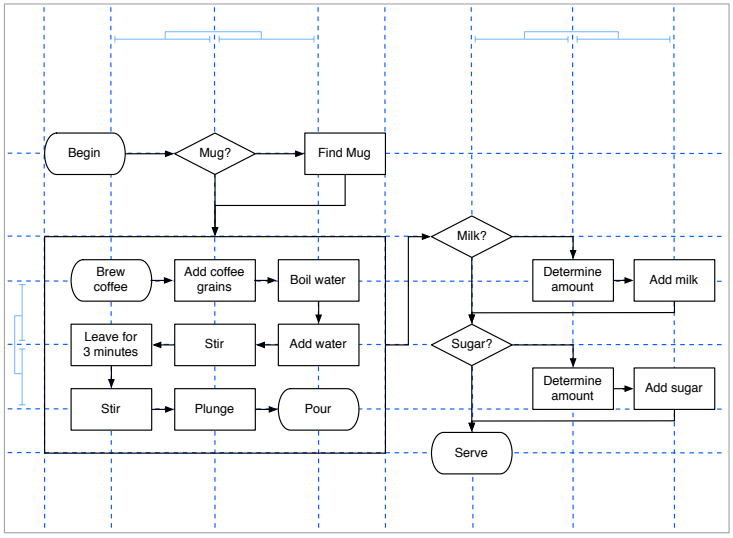
Figure 4.5: Diagrams for the "Beautify" task. The task required participants to set up relationships, with minimal manipulation of constraints. The second diagram (b) was also the initial diagram for the "Reorient" task.

- **"Reorient"**: The aim of this task was to study user manipulation of existing constraint relationships. It required the participant to change the layout of the diagram without changing the relationships between objects. The initial diagram for this task is shown as Figure 4.5(b) and the target diagram for this task is shown in Figure 4.6(a). This task had the following motivation: "We decide that the diagram could best be displayed with a different page orientation and accompanied by some descriptive text. To make room, we need to rearrange and reduce the diagram, without modifying its content, so that it fits into the left half of the landscape page."

- **"Rearrange"**: The primary aim of this task was to study removal and addition of objects to relationships (e.g. taking shapes from alignments and putting them in other alignments, taking guides from one distribution and putting them in another). There was also a smaller amount of manipulation of guidelines and distributions. Part of the purpose of this task was to determine whether the more persistent nature of multi-way constraints as compared to one-way constraints would make them less useful when placement relationships needed to be altered, or to have selected shapes broken from them. The initial and target diagrams for this task are shown in Figure 4.6. The following motivation was given for this task: "For a report of some kind we decide the diagram is best displayed alone, with a slightly different layout, on a full page."



(a) Initial diagram



(b) Target diagram

Figure 4.6: Diagrams for the "Rearrange" task. The task required participants to remove and add objects to existing constraint relationships, along with some manipulation. The first diagram (a) was also the target diagram for the "Reorient" task, which required manipulation of existing constraint relationships.

The tasks were designed to set up and expose the participant to certain common situations that occur in interactive constraint-based systems. "Beautify" compels the user to construct distributions involving several groups of aligned shapes. It examines how they interpret and formulate these relationships, given the alignment and distribution tools. Importantly, the task has the user themselves create all the constraints that will be present in the following task.

The "Reorient" task is structured in such a way that the user not only has to move constrained objects, but also has to resize both distributions and a shape—the bounding box containing the coffee brewing sub-process—involved in three vertical alignment relationships, one of which is part of an additional distribution. This means that the seemingly simple first task and beginning of this task purposefully set up a quite complex system of interlinked constraints. The act of interacting with and modifying this system of constraints tests the users' understanding of the tools' behaviour. Complexity is important here since many possible usability problems with constraint-based tools are due to the difficulty of relaying information about the underlying constraints in an accessible, comprehensible manner. On the other hand, simple systems of constraints are often trivial to understand. Hence it is important to have tasks involving complex systems of constraints, but systems the user has themself created so that they seem accessible.

The "Rearrange" task was designed so that the participant needed to make changes to the diagram necessitating removal or alteration of existing constraints. This task again set up situations that tested the user's understanding of the placement relationships within the diagram, as well as how they might alter or replace them when the constraints imposed are no longer desired.

These exercises were arrived at after several iterations of pilot tests. To the participant, they involve a realistic flowchart, and plausible motivation is given for the changes they are required to make in each task. Additionally, they subtly lead the user to create complex systems of constraints that help examine potential usability issues with the tools.

4.3.2 Results

In this section we present and discuss the results of the usability study, examining the times taken to complete each task. We used the same statistical analysis methods as those used in the first study (see Section 3.4.2). All participants completed the tasks.

The average completion times for each component task as well as for the complete exercise are shown in Figure 4.7. To determine where the statistical significance lies we perform an ANOVA for each.

The complete exercise series showed significant difference in completion times ($F = 8.12$, $p < 0.001$). Times for this exercise are summarized in Figure 4.8. Tukey's HSD test showed that there was significant difference between times for Group OW/DF and the multi-way tools, Group MW/DF and Group MW/IF, as well as significance between times for Group OW/IF and both multi-way tools. This shows that the multi-way constraints, with or without immediate feedback, were more beneficial than either of the one-way based groups. Rather surprisingly, it also shows that providing immediate feedback did

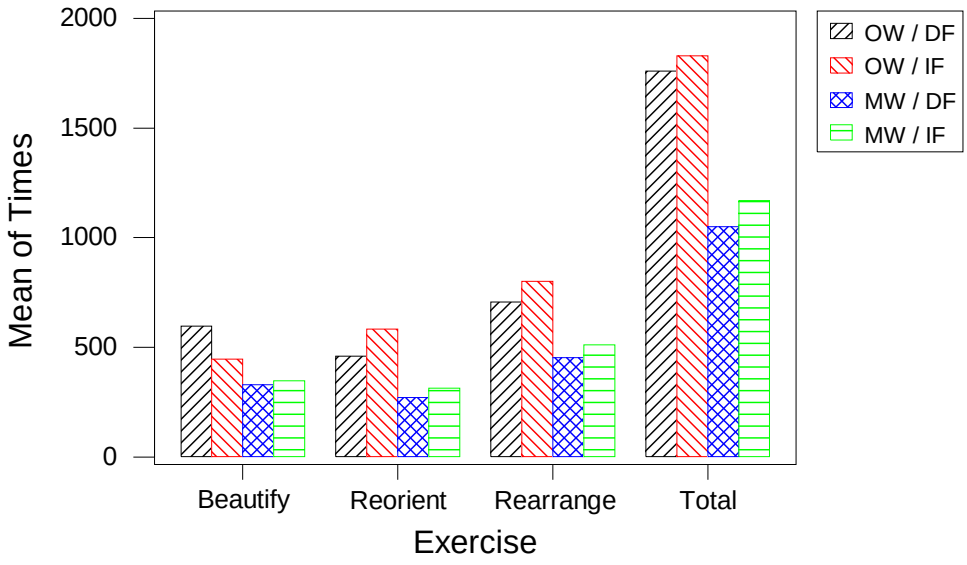


Figure 4.7: Mean completion times (in secs.) for each exercise.

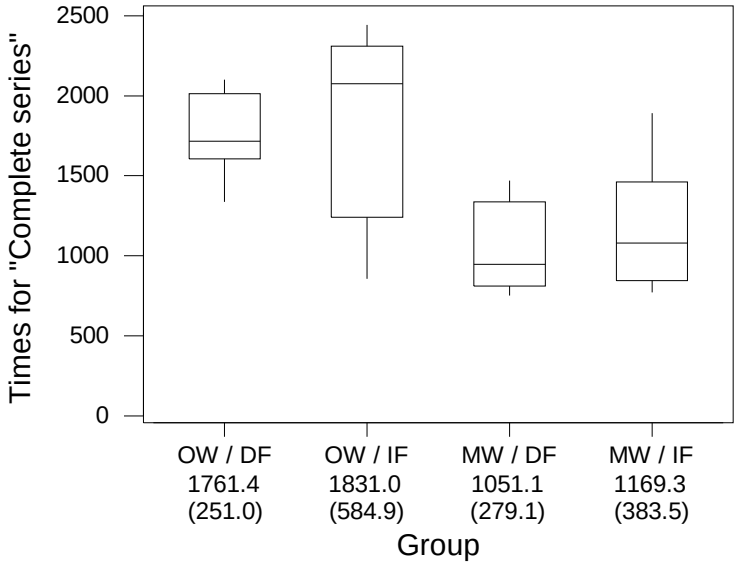


Figure 4.8: Box-plot of completion times for the complete exercise.

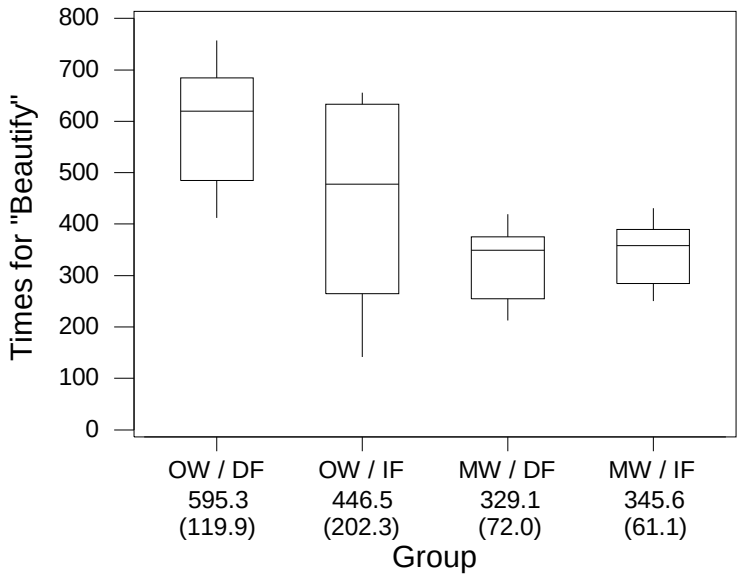


Figure 4.9: Box-plot of completion times for "Beautify" task.

not make a significant difference to completion times of users for either the one-way or the multi-way tools. Indeed, is appears immediate feedback increases completion time.

Next we consider the individual component tasks. In general these support the results for the overall exercise. A one-way ANOVA shows clear statistical significance to differences between groups in the first task "Beautify" (unequal group variances, $F = 7.46$, $p = 0.001$). Times for this task are summarized in Figure 4.9, a standard box-plot, as described in Section 3.4.2.

To see exactly where the significance lies we use Tukey's HSD test to consider all pairwise differences between group means. Using this method we find that the only significant differences are between Group OW/DF and Group MW/DF, as well as between Group OW/DF and Group MW/IF. In this task it is clear that the multi-way constraint-based tools of Group MW/DF and Group MW/IF definitely offer some benefit over the one-way constraints, without feedback, of Group OW/DF. There is no significant difference between the multi-way tools and the one-way tools with feedback.

We again determine significance in the completion times for the second task "Reorient" ($F = 5.04$, $p = 0.006$). Times for this task are summarized in Figure 4.10. For this task, using Tukey's HSD test, we find that the only significant differences are between Group OW/IF and Group MW/DF, as well as between Group OW/IF and Group MW/IF.

This task saw participants using existing placement relationships (set up in the previous task) to resize the diagram. We see that the multi-way tools of both Group MW/DF and Group MW/IF offer significant benefit over the one-way tools of Group OW/IF, though not those of Group OW/DF. Interestingly, the immediate feedback version of the tool is significantly slower than the multi-way tools, rather than the non-feedback version, as we had anticipated. A possible explanation for this is presented at the end of this section.

The third task in the series, "Rearrange" also showed a significance in completion times (unequal group variances, $F = 6.49$, $p = 0.002$). Times for this task are summarized

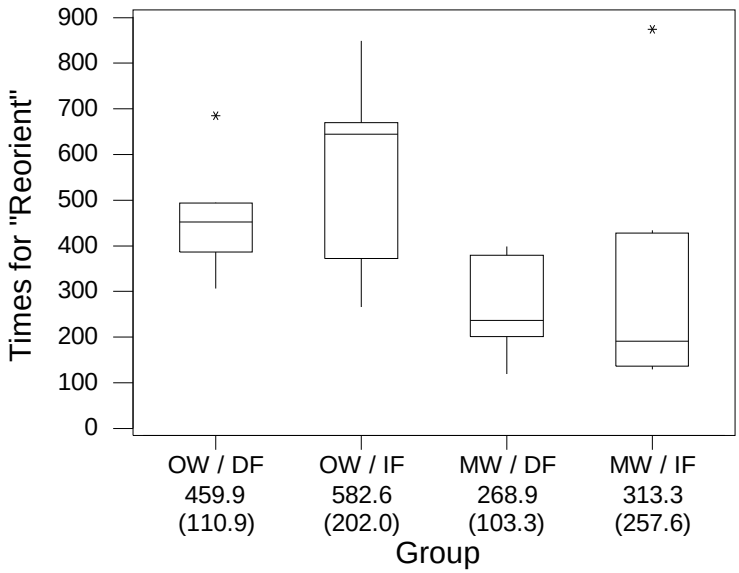


Figure 4.10: Box-plot of completion times for “Reorient” task.

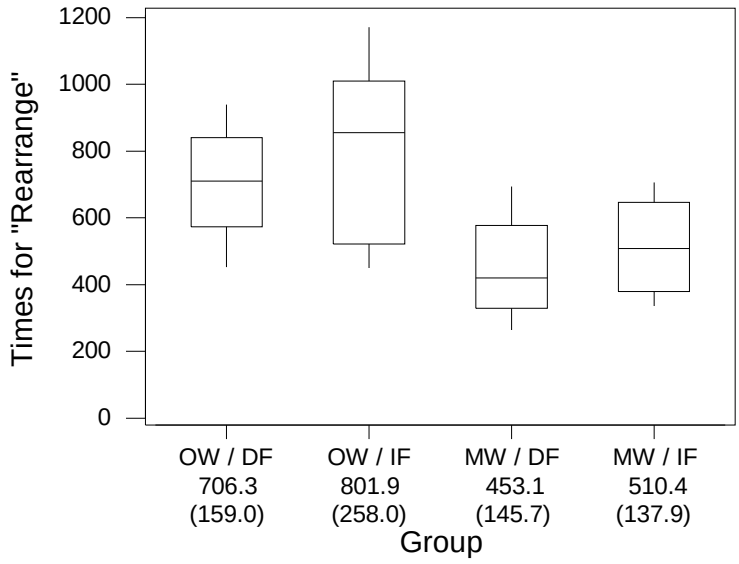


Figure 4.11: Box-plot of completion times for “Rearrange” task.

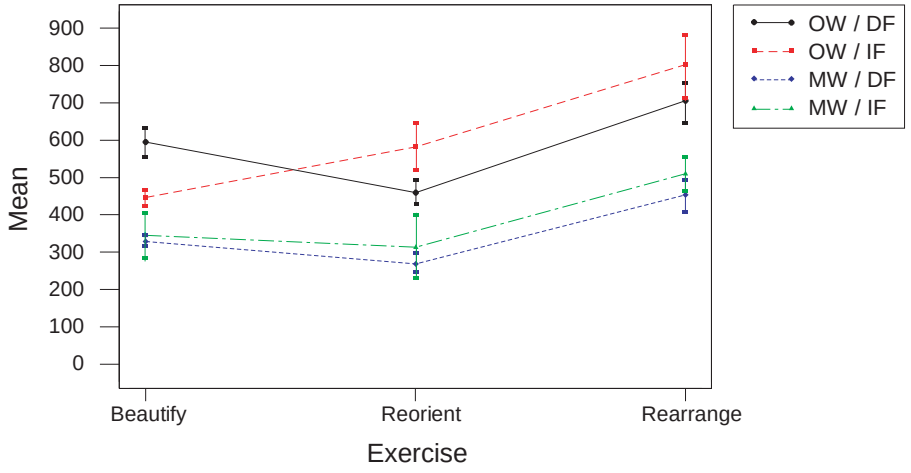


Figure 4.12: Interaction plot for Groups OW/DF, OW/IF, MW/DF and MW/IF.

in Figure 4.11. Tukey's HSD test showed that there was significance between times for Group OW/DF and Group MW/DF, and also between times for Group OW/IF and the multi-way tools, Group MW/DF and Group MW/IF. This shows that the multi-way constraints without feedback of Group MW/DF are more beneficial for manipulation and alteration of existing constraint-based placement relationships than either of the one-way based tools. In addition, the multi-way Group MW/IF tools are significantly more beneficial than one-way feedback Group OW/IF tools.

We are also interested in whether there was any interference between the groups and the individual component tasks. Figure 4.12 shows group means as an interaction plot with error bars. An absence of interaction is illustrated by the consistent separation of the lines for the one-way lines versus the lines representing the multi-way group. Since each task was designed to test a different type of interaction with the constraint tools, this shows that the benefit of the multi-way tools compared with the one-way tools was not limited to a particular task. Ultimately, multi-way tools can be considered more beneficial in terms of task time than one-way tools.

Probably the most surprising result of our experiment was that there was no significant difference between the feedback version and the version without feedback of either type of tools, in any of the component tasks. In fact, very surprisingly, the feedback versions performed worse than the non-feedback versions of the tools for all cases except the one-way tools in the first task (see the line crossing of Group OW/DF and Group OW/IF in Figure 4.12). This is likely because when setting up relationships and testing them, it is beneficial to have feedback for the one-way tools where relationships can break easily.

The last two tasks required considerably more object movement. This meant that participants often had to undo erroneous actions or actions that inadvertently affected other constrained objects. We observed that participants with immediate feedback were more likely to undo the move by manually dragging objects back to their last position, than using the undo command. Since precisely placing objects by dragging is slow and

the undo command is instantaneous, perhaps the times for the feedback group could have been increased as a result of the feedback.

The version of Dunnart used during the study lacked an "escape" option. Present in tools of many diagram editors, hitting the Escape key allows the user to cancel the current action while it is in progress. While we are unable to determine exactly how many participants attempted to cancel actions by hitting escape (several mentioned the omission in the debriefing), we conjecture that the presence of an "escape" option may have avoided the slowdown we observed from undo-dragging.

Though the presence of immediate feedback did lead to slower overall completion times for the groups using it, a noticeable benefit to its presence was observed during the experiments. Participants in these groups would regularly make use of the immediate feedback by dragging an object back and forth slightly to determine which objects it was attached to. This act of *jiggling* objects to determine the effect on the environment is a real world action that was not taught to participants, but one they instinctively used once they saw their actions had immediate consequences. Jiggling appeared to be used by participants as a fast confirmation that objects were connected in the way they believed them to be. After jiggling, participants would often try to (slowly) return the diagram to its exact state prior to the jiggling action. Here, users would definitely benefit from the provisionality afforded by hitting the Escape key to cancel the jiggling action.

It is known that motion highlighting, the act of highlighting specific nodes or edges in a graph with movement, improves users response times and accuracy answering questions about graph structure when used to draw attention to relevant portions of graphs (Ware and Bobrow, 2004). An interesting result from our study is the user's intuitive, manual use of the technique for aiding their own understanding of constraint-based relationships.

4.3.3 Discussion

Careful examination of the replays of the experiments revealed interesting information about participants' interaction with the tools and indications for possible improvements to the placement tools and to Dunnart itself. In particular, we examined actions with unexpected consequences. These were identified as actions made by the participant which caused an observed result and were undone immediately, or corrected manually with an opposite action. To further strengthen the argument that the participant expected a different result, these actions were often immediately followed by the same or a similar attempt at the same action.

Reinforcing our hypothesis that the multi-way based tools are more usable than one-way based versions was the observation that only participants using the one-way tools unintentionally broke placement relationships, that is, detached shapes from guidelines or guidelines from distributions. Even with our improved one-way tools this occurred frequently, between three and twelve times for *all* participants in the one-way group.

Roughly sixty percent of participants dragged an object and unexpectedly found one or more other shapes moving as a result. The breakdown of these participants was OW/DF: 4,

OW/IF: 3, MW/DF: 6, and MW/IF: 6. As expected, this was less common in the one-way groups where it could only happen when dragging placement indicators. In these groups dragging shapes always breaks them from relationships, so related objects are never unintentionally dragged. For one-way participants the problem could sometimes be attributed to shapes accidentally being dropped and attached to a guideline, which became noticed later when moving the guideline. For the multi-way groups, where this problem was more frequent, participants had difficulty understanding exactly why a large group of shapes were moved as a result of moving a single object.

Most likely, users did not realize that relationships acted in chains—that moving a shape would move everything attached to it, and in turn everything that was attached to each of those secondary objects, and so on. Effectively, if the user didn't have a strong understanding of the way in which placement constraints interacted with each other, then it became a hard mental operation to fully comprehend object behaviour. This reaffirms previous claims of the difficulty in how best to communicate indirect connections between objects as a result of constraints. It also suggests that further usability research is needed to examine the reason for users misunderstanding constraint relationships and to develop better methods of communicating these relationships. We shall return to this problem in our subsequent usability study.

As with the previous study, many of the participants complained about clutter and/or suggested that the visibility of placement indicators could be toggled on or off in some way. Clutter related issues were pointed out by roughly fifty percent of participants in the multi-way groups, though only by a couple of participants in the one-way groups. We surmise that this difference is the result of the one-way group having more important complaints and suggestions affecting the usability of the tools. The clutter issue for placement indicators will be examined as part of our third usability study in the remainder of this chapter.

One common action was for participants to attempt to resize distributions by dragging their outermost guidelines. While this behaviour is the behaviour shown by Visio's tools, this alone cannot explain it since nearly two-thirds of participants attempted this. Of these, only one-fifth had used Visio in the past and less than one-third were from the immediate feedback groups. One explanation for this might be that these participants better understood distribution indicator objects and their use from the training as a result of immediate feedback. Interestingly, participants who found the action didn't work with one outer guideline would often immediately undo and try to resize the distribution with the other outer guideline.

Possibly this is an issue with closeness of mapping—that is, the user expects the guidelines and shapes to have weight, but the distribution to have no strength. This would result is the total shapes involved in the distribution moving as little as possible. Effectively, dragging the outer guideline would resize the distribution. In our model the distribution itself has strength to maintain its separation, except when altered by the user while being resized. Resizing via the outer guidelines is certainly behaviour that could be implemented with a mix of constraints and code for the standard case where

there aren't distributions within distributions. Further usability studies could examine the exact behaviour that naive users expect from the tools as described and presented in their current form.

A final, unexpected observation was that 80% of participants had the expectation (confirmed by discussion in the debriefing) that the software would reason about the current positions of shapes and treat what appeared to be rows and columns as groups, and align or distribute these accordingly. E.g., if there were six objects roughly in two columns, then applying a left alignment to the entire six objects would result in two alignment relationships rather than just the one. The breakdown of these participants was OW/DF: 6, OW/IF: 7, MW/DF: 6, and MW/IF: 7. Many participants attempted this repeatedly, suggesting it could be worth adding a feature that allows automatic inference of constraints.

In addition to demonstrating the benefits of placement tools based on multi-way constraints, this experiment highlighted two significant usability issues with the tools, both of which were also evident in the first study. Results and comments solicited from participants in the study suggested that further research should focus on how to best represent constraints so that the diagram does not become too cluttered. They also prompted investigation into better visual and non-visual feedback during interaction to aid the user in comprehending complex systems of constraints and explaining unexpected interactions between constraints.

4.4 Constraint clutter and comprehension

The placement tools in Dunnart were revised in an attempt to address two recurring problems experienced by the participants in the previous studies. Firstly, that constraint indicators could obscure objects in the diagram and cause clutter. Secondly, complicated systems of constraints were frequently difficult for users to understand.

4.4.1 Dunnart revisions

A couple of different visibility options were added to the placement indicators to attempt to solve the problem of clutter. It was felt that reduced visibility—making the indicators slightly transparent by reducing their alpha level—was a good idea. However, they still needed to be highly visible at some times, so we investigated having them glow when active and fade when inactive for some time.

We added several controls that could be used to adjust the visual appearance of the indicators. These controls, as shown in Figure 4.13, are displayed in the interface on the left side of the canvas.

Indicator visibility

The "Glow when active" option causes alignment and distribution indicators to be highlighted when they are actively causing shapes to be moved. When guidelines or distribution

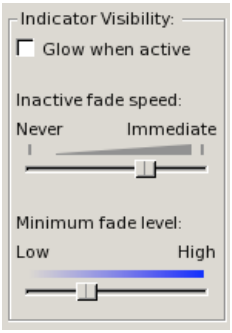


Figure 4.13: Indicator visibility controls

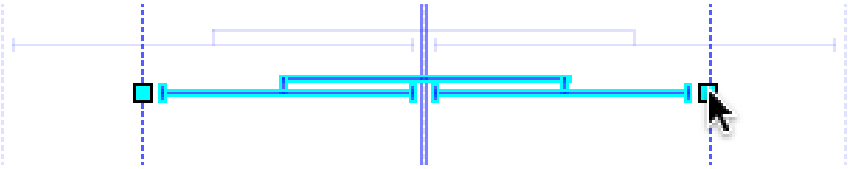


Figure 4.14: The visual difference between active and inactive indicators, with fading enabled and a low minimum fade level. The smaller distribution is being resized around the centre guideline. At this point the active indicators are fully visible, and the inactive indicators have faded into the background.

indicators are moved either directly by manipulation or indirectly via constraints, they glow a bright orange before returning to their previous appearance after a second.

The “Inactive fade speed” slider controls whether indicators are faded over time when inactive. The “Never” setting at the far left of this slider is the default setting. In this setting the indicators are never faded. This is equivalent to the previous revision of the tools. In this setting the “Minimum Fade Level” slider below is disabled and has no effect.

The “Immediate” setting at the far right of the fade speed slider causes all indicators to have a fixed, constant faded level that can be controlled by the “Minimum Fade Level” slider. The far left of the fade level slider has the indicators fading to almost invisible, whereas the far right has them almost at full visibility.

Any other position for the upper, “Inactive fade speed” slider causes the placement relationship indicators to fade into the background when they have been inactive for some amount of time. When they are active they will always be fully visible. The exact slider position determines how quickly the indicators should fade out when inactive. Fade times selectable by this slider range from 0.5 up to 6 seconds. While fading is enabled, the lower “Minimum fade level” slider controls the minimum level of visibility that the indicators will fade down to. Whenever an indicator becomes active and is moved, it is set to maximum visibility, and restarts its process of fading. This behaviour can be seen in Figure 4.14.

These controls could have been expressed through the user interface in different ways. The options and controls described were chosen after a couple of iterations of feedback since we felt they allowed most useful alternatives while remaining easy to comprehend. We originally allowed the user to set the upper visibility level that fading indicators return to

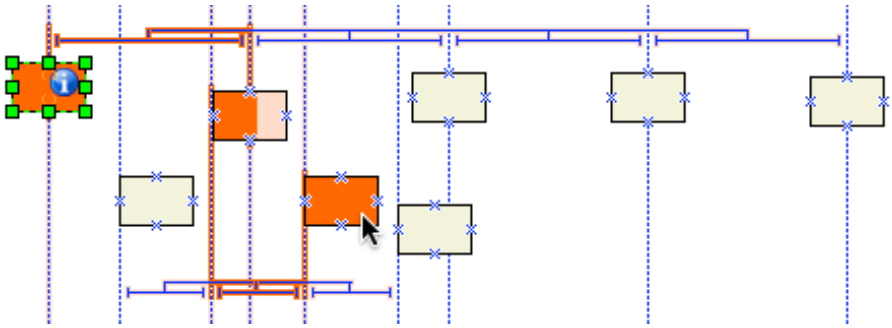


Figure 4.15: Highlighting of constraint relationships between two objects with Information Mode. The path of placement connections between the selected shape and the shape under the mouse cursor is shown.

when active, but later removed this. It adds another layer of complexity to comprehension of visibility controls and our pilot tests showed it to not be useful—users tend to want the indicators to be fully visible while they are active.

Constraint information mode

The second problem encountered during the previous study was the comprehension issue of participants sometimes not understanding why objects move unexpectedly and how they are attached to one another. In an attempt to combat this problem we created a query tool dubbed "Information Mode" which shows how two shapes are attached to each other via constraints. This tool finds the path of constraints between two objects and illustrates this to the user by highlighting the relevant constraint indicators.

An example of how Information Mode appears to the user is shown in Figure 4.15. While Information Mode is active and a single shape is selected, Dunnart will highlight the placement relationships between the selected object and the object under the mouse. Since shapes may be attached separately in the x and y dimensions, Dunnart will highlight the most direct chain of constraint indicators in each dimension. Information Mode is activated or deactivated via a toolbar button. Early feedback from some pilot tests revealed a preference for it to be a persistent mode rather than being triggered by holding down a key. This preliminary testing also revealed user difficulty in determining connections when every relevant constraint indicator was highlighted. Hence the two-tier highlighting seen in the final version in Figure 4.15. We lightly highlight the full indicators on the chain of connections between the two objects, and more strongly highlight a subsection of these indicators in an attempt to emphasise the direct path between the objects.

The only other (minor) modification to Dunnart caused any in-progress action to be undone by hitting Escape at any time, as requested by participants of the previous experiment. This offers the user provisionality—that is, the ability to test what might happen without committing to the action.

4.5 Study 3: Constraint clutter and comprehension

Our third usability study[5] evaluated the effectiveness of a constraint information mode to aid user comprehension and several visualisation techniques aimed at reducing diagram clutter due to constraint indicators.

4.5.1 Method

Design

The third study was designed to evaluate the effectiveness of the revisions to the placement tools. Firstly, to determine if giving constraint indicators a reduced visibility, or having them fade out or glow, would reduce the problem of clutter for users. Secondly, to evaluate the constraint information mode as a method of helping users determine the constraints between objects in their diagrams.

The study design mirrors the previous experiment. Since the motivation for our revised tools came from difficulties participants encountered working on the tasks from that experiment, it was logical to evaluate the revisions with the same study procedure and experimental tasks. Details of the exercises can be found in Section 4.3.1.

For this experiment there was only one group: those using the revised multi-way versions of the tools with immediate feedback. We would be comparing their experience with that of the MW/IF group from Study 2.

The training was very similar to the training given to the MW/IF group from Study 2. It differed in its additional presentation of the indicator visibility controls and the explanation of the constraint information mode. These feature were presented at the end of the multi-way tool training. The visibility control features were presented as tools to help the user in working with the diagram if the presence of a large number of on-screen constraint indicators made this difficult. Participants were introduced to these tools in the order of the controls in the interface. They were told that they could use them freely if they felt it necessary. Information mode was presented as way to determine how an object was attached to other objects, if this was not apparent at any point to the user. They were again informed that they were free to use this tool at their own discretion. Participants were given a reference sheet showing the visibility controls and Information mode button with details of each control's purpose.

It was hypothesized that participants would choose their preferred visibility settings and that we would see an overall trend for these, representing the most useful settings. We would then be able to use this information to gauge the relative effectiveness and user satisfaction for different visibility settings. It was also hypothesized that the constraint information mode would be useful to the user in decoding particularly complex systems of constraints. We expected to see less criticism of the tools in regards to clutter problems, and observe less confusion of complex constraint relationships from participants.

[5]This study received ethics approval from the Monash SCERH as project 2006/085.

Participants

Ten people were tested. There were no requirements for participants other than that they be computer-literate adults. All participants were university students or staff who were native speakers and readers of English, with normal or corrected-to-normal vision. Participants were not reused from previous experiments.

Equipment

All tests were carried out in private, the investigator testing participants singly. The environment for the experiment was a private office in which the participant sat at a computer while the investigator sat behind them, observing and taking notes.

Interactions made by each user during their session were recorded via the logging mechanism of Dunnart so that their actions could be played back for reference purposes. Also collected was a log file for each test containing the times and type of every action the participant made in completing the exercises. Notes taken by the investigator summarized the strategy and method taken by the user to carry out the task, as well as problems they experienced. These were used to prompt discussion during the debriefing.

The default settings for the visibility controls resulted in tools that looked the same as those used by the MW/IF group from Study 2. All interaction with the visibility controls was logged and written to the log file, so it could be easily determined when the user made changes to them, and how long they worked with particular settings.

Short pre- and post-test surveys were used as a means of obtaining some additional qualitative and quantitative data about participants' experience with related tools, how difficult they found the exercise and suggestions they had for improving the tools.

The post-test survey also asked specifically which *active constraint* setting participants preferred when working with the placement tools. They were offered the choices of "None (indicators are always shown the same way)", "Fading (unused indicators fade out over time)", and "Glowing (active indicators glow)". In addition they were asked whether they preferred placement indicators to be "Opaque (solid looking)" or "Transparent (slightly see-through)".

At the beginning of each experiment participants were taken through a thirty minute scripted training exercise that introduced them to Dunnart and described its basic features while requiring them to interact and experiment with it. The placement tools they were to use were explained and the training required them to set up and interact with these relationships. This was all conducted informally so the participant was able to interrupt the training and ask for clarification on specific points or spend a little more time on any aspect of the training they felt needed extra attention. When participants completed the training, had no further questions and indicated that they were comfortable to go on, they proceeded to the timed exercises.

Materials

The exercises for this experiment were the same as those from the previous experiment. They are described in detail in Section 4.3.1.[6]

4.5.2 Results and discussion

For this experiment we wanted participants to experiment with the visibility settings they preferred, so we removed the emphasis on completing exercises quickly as well as the text on the handouts indicating they were being timed. Overall, as expected, this resulted in the participants completing the exercises slightly slower than the members of Group MW/IF from the previous study. For this reason we do not further analyse completion times.

Default visibility settings

Interestingly, we found that not all participants experimented with the indicator visibility controls during the experiment. Six out of ten participants chose preferred visibility settings and used them during the exercises. The other four kept the default settings of full visibility—no fading and no glowing—effectively giving them exactly the experience of members of Group MW/IF from the previous experiment.

Of these four participants, all of them reinforced their preference for the defaults with their choices in the post-test survey. All selected "No effect" over "glowing" and "fading". They also all indicated they preferred the placement indicators to be "not faded". Importantly, none of these participants reported "clutter" being a problem during the experiment.

To explain this we examined more closely the past experience of these participants and found that all four had very little or no experience with diagramming software. In contrast, of the other six participants, five had a significant amount of experience with diagram or CAD software.

It is likely that the inexperienced participants would have had more difficulty assimilating all the intricacies of the training, and thus would have been less comfortable to experiment with visibility controls. In support of this, a similarly inexperienced participant in a pilot test prior to the study stated that they felt the training was fairly complex. They did not use the visibility controls, and when asked about it they responded that they felt that print preview was a simple and familiar alternative for viewing the diagram without constraint indicators. They went on to explain that they "switched off" for the description of the visibility controls and felt no need for their use. Accordingly, after the pilot tests we revised the study materials, simplifying the description of the visibility tools in the training and providing an additional reference sheet explaining the individual controls and their effect. We also removed the print preview command from Dunnart for the study sessions.

[6] Complete copies of exercises, instructions and surveys for this user study are available online: `http://www.csse.monash.edu.au/~mwybrow/wybrow-thesis-exp3-materials.tar.gz`

Simulating print preview

An interesting behaviour we noticed during the study was the act of using the visibility controls as a replacement for the print preview command. Participants did this by setting the minimum fade level to its lowest value, meaning indicators fade to almost invisible. They would then set the fade speed to immediate, or close to. We observed participants check their finished diagram matched the exercise target diagram by quickly using the visibility controls to hide or show the placement relationships. They would leave the settings like this for several seconds while checking the on-screen diagram by eye before restoring the visibility settings back to their preferred working settings. Four out of five of the experienced participants, and one out of five of the inexperienced participants used the controls in this way. In the post-test survey, one participant suggested that a key could be depressed to instantaneously fade out placement indicators and released to return them to their previous visibility.

Indicator visibility preferences

The post-test surveys revealed that all five of the experienced participants stated they preferred that the indicators fade when not in use, rather than glowing or having no visual change. All five also stated that they preferred the indicators to be partially faded when not in use, rather than being fully opaque.

For the inexperienced group, four of five participants stated in the survey that they preferred that indicators did not fade or glow when active or inactive. The same four also indicated that when not in use they preferred indicators to always be fully opaque. The other inexperienced participant stated their preference for both fading of inactive indicators as well as glowing of active indicators. They preferred indicators to be partially faded when not in use.

It is interesting to note that only one participant expressed a preference for and made significant use of the glowing feature for active indicators. While they expressed a preference for the feature we noticed it caused them some difficulties. This participant was seen to have more problems with attaching shapes to guidelines. They repeatedly attempted to attach shapes to a guideline by dragging and dropping the shape over a guideline only to have it miss and not become attached. The visual cue that indicates an imminent attachment is the guideline glowing orange. It would seem that the constant flashing of similarly highlighted indicators as objects were dragged caused the user to become somewhat desensitised to the visual cue and then ignore it. This desensitisation is supported by the participant's feedback in the post-test discussion in which they stated there was "a lot of red, I'll just ignore that for a while". This, and the absence of glowing use by the other nine participants during the study exercises, suggests that making active placement relationship indicators glow when they are active is not a useful visual cue.

Of the users who used visibility settings other than the defaults, for general editing they all used a minimum fade level corresponding to an alpha level of either 70 or 100, where an alpha level of 0 would be invisible, and 255 would be fully opaque. In terms of on-screen appearance this corresponds to the indicators being at close to their lowest

visibility while being still clearly visible. With a setting any lower than this, it becomes difficult to see the indicators at all.

Four participants selected preferred values where inactive indicators faded out over a four to six second interval. Two chose to have the indicators fade to their reduced level immediately when inactive. There was no clear consensus here in terms of preference for fade speed. It would appear that experienced users prefer to have inactive indicators be at reduced visibility, and that they fade immediately when inactive, or fade out gradually, without the fading itself being noticeable.

One participant spent part of the exercises with the minimum fade level set to its lowest setting, meaning the indicators were effectively invisible. Several times they had difficulties with these visibility settings when they attempted to click and drag an empty point on the canvas to select multiple objects, only to accidentally drag a hidden indicator. This participant's suggestion in the post-test survey was to enforce a minimum visibility level for placement indicators so that indicators could never be completely invisible when working with them (except as a print preview action).

Importantly, no participants complained of clutter caused by constraint indicators. This suggests that fading of inactive constraint indicators down to a reduced level of visibility is an appropriate solution to the issue of clutter.

Constraint inference expectations

As with the previous study, slightly over half of the participants first expected the placement tools to do some amount of constraint inference. These six participants attempted to align or distribute groups of shapes and have their relative positions be translated into separate alignments automatically. However when asked about this in the post-test discussion, the consensus among those participants was definitely for the tools not to do this automatically. It would appear that many have been previously annoyed by computer tools that attempt (badly) to infer their intentions. For example one participant responded "don't make it too clever ... not like the bullet points in [Microsoft] Word". Another stated simply "if it did it automatically, it'd be a pain".

Information mode

Information mode was used by about half the participants, mostly the experienced group. It appeared to be generally useful, both for determining constraint connections between objects in the diagram and for determining where these connections were missing. Participants also offered various suggestions for its improvement. One participant pointed out that the tool could be easily adapted to work with connectors in addition to constraints, to allow the user to determine physical connection of shapes within the diagram. Two participants stated that its output was too complex, one stated it needed to be explained better. One participant suggested having a separate colour to show connections between shapes in the x and y dimensions. One participant suggested that information mode should be active while a certain key combination was being held down, since you tend to want to use

it just for a certain query and also because they (and another participant) left it turned on while carrying out further manipulation of the diagram and found it distracting.

One of the inexperienced participants said they didn't find the Information Mode useful, and suggested instead a visualisation purely to show objects directly attached to the selected object. This suggests they didn't understand the higher level metaphors well and what it means for there to be connections between them, that is, objects in an alignment that is part of a distribution. As a result, they were not so interested in how objects were connected to each other at a high level, but instead just wanted the answer to the more immediate question of "what is this object attached to?"

General suggestions

A simple interface improvement suggestion was to automatically reposition distribution indicators when they overlapped each other and to keep them on the canvas when the canvas view was changed. Guidelines are already drawn infinitely long so always have an on-screen representation. In the same way, an indicator of each distribution should always be visible on-screen while the guidelines in the distribution are visible.

Another general suggestion was to be able to "fix a section/piece of the diagram" that had been satisfactorily arranged. The idea was that this would then fade away slightly and remain unchanged throughout further editing of the rest of the diagram.

4.6 Conclusions

Despite the large amount of research in the area of constraints and graphical editors, there have been few if any formal studies to compare the usability of the various constraint-based systems that have been presented. In particular, there has been no investigation of the general claims that multi-way constraint-based tools are better than one-way constraint-based tools.

We have described a formal experiment comparing the usability of one-way and multi-way constraint-based alignment and distribution tools in diagram editors, the results of which provide strong support for our hypothesis that multi-way constraint-based placement tools are more usable than one-way constraint-based placement tools.

We believe the reason that the multi-way constraint-based placement tools are more usable than the one-way constraint-based placement tools is that one-way constraints have a fixed direction and an attribute can only have a single formula associated with it. This means that alignment and distribution relationships can silently break due to manipulation of objects involved in the relationship or because more than one constraint is applied to the same object.

Of course this is not to say that multi-way constraints are better than one-way constraints in other tools or other applications. For purposes in which the direction of constraint solving is fixed such as widget layout (Myers, Giuse, Dannenberg, Vander Zanden, Kosbie, Pervin, Mickish and Marchal, 1990; Myers, McDaniel, Miller, Ferrency, Faulring, Kyle, Mickish, Klimovitski and Doane, 1997; McCormack, Marriott and Meyer, 2004)

and more generally adaptive page layout (Weitzman and Wittenburg, 1994; Hurst, Marriott and Moulder, 2003; Jacobs, Li, Schrier, Bargeron and Salesin, 2004) or incrementally updating views of data (McDonald, Stuetzle and Buja, 1990; Myers and Kosbie, 1996), one-way constraints seem preferable to multi-way constraints because of their simplicity, efficiency and expressiveness.

Our experiment also tested the impact of visual feedback provided during direct manipulation. We evaluated delayed feedback, where the constraint-based placement relationships are updated only after movement actions are completed. This was compared against immediate feedback, where the positions of all objects are updated immediately and the user can see all changes to the diagram continuously during direct manipulation. While providing immediate feedback showed obvious benefits and was preferred by user, it also appeared to slow users down, causing them to take longer to complete diagramming tasks. While this slowdown was not statistically significant it is interesting since the general belief has been that immediate feedback is purely beneficial (Gleicher and Witkin, 1994; Ryall et al., 1997). This result certainly does not provide adequate reason to remove immediate feedback from interactive applications, but rather suggests an interesting property of such interaction that should be taken into account when designing direct manipulation systems.

Also intriguing, we observed participants instinctively jiggling constrained objects to confirm their attachments to placement constraints. The value of automatic movement highlighting of objects has been shown previously, but Dunnart provides a live environment in which users use this technique without prompting.

The experiment also highlighted two main usability issues with our multi-way constraint-based placement tools. Firstly, constraint indicators could obscure the diagram and cause clutter. Secondly, complicated systems of constraints were frequently difficult for users to understand. To address these problems we again revised the tools, adding visibility controls for indicators as well as a constraint information mode.

We conducted a follow-up user study examining the usefulness of these additions. We found that the visibility controls, specifically, fading indicators while they are inactive down to a minimum fade level, provided an effective solution for the clutter problem. The study showed Information Mode to be a beneficial addition, but not the complete solution to the issue of user comprehension.

While these studies were conducted specifically for constraint-based tools for alignment and distribution, the findings extend to other similar interactive constraint-based tools. Such tools equally benefit from findings related to immediate feedback and visibility settings.

Further research could examine the usefulness of automatic inference of placement relationships. It could also further explore techniques for improving the user's understanding of complex systems of constraints.

Chapter 5

Connector routing

5.1 Introduction

Most diagram editors provide some form of automatic connector routing. They typically provide orthogonal connectors as well as a type of poly-line or curved connectors. Diagram editors provide an initial automatic route when the connector is created and again each time the connector endpoints (or attached shapes) are moved. The automatic routing is generally chosen by an ad hoc heuristic.

As discussed in Section 2.4.5, the connector routing approaches of popular diagramming tools are less than ideal and suffer various deficiencies. They tend not to keep connectors optimally routed as shapes are added and removed from the diagram. Connector routes mostly ignore shapes rather than routing around them and the routes produced are often surprisingly unpredictable, see Figure 5.1.

Automatic connector routing in diagram editors is, of course, essentially the same problem as edge routing in graph layout. We aim to route a poly-line, orthogonal poly-line or spline between two nodes and find a route which does not overlap other nodes. Ideally, the chosen route should be aesthetically pleasing: short, with few bends and minimal edge crossings. The main difference between routing in graph layout and in diagram editors is that the former is typically performed for a once-off static layout, often as a separate phase after nodes have been positioned. On the other hand, connector routing in diagram editors is dynamic and needs to be performed whenever the diagram is modified.

One well-known library for edge routing in graph layout is the Spline-o-matic library developed for GraphViz.[1] It supports poly-line and Bézier curve edge routing and has two stages. The first stage is to compute a *visibility graph* for the diagram. The visibility graph contains a node for each vertex of each object in the diagram. There is a visibility edge between two nodes iff they are mutually visible, that is, there is no intervening object, see Figure 5.2. In the second stage connectors are routed using Dijkstra's shortest path algorithm to compute the minimal length paths in the visibility graph between two points. A third stage, actually the responsibility of a separate rendering engine, is to compute the visual representation of the connector. This might include adding rounded

[1] Spline-o-matic, AT&T Research, `http://www.graphviz.org/Misc/spline-o-matic/`

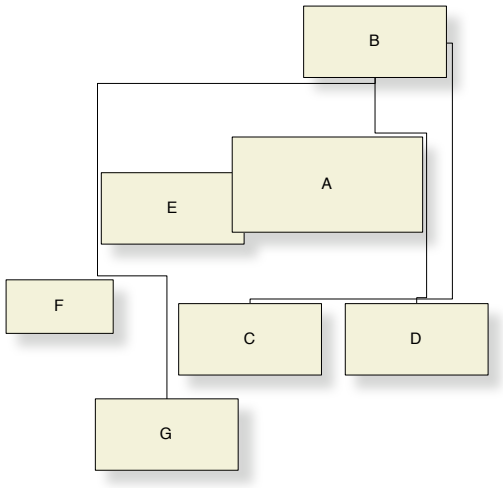


Figure 5.1: Orthogonal connector routing produced by Visio. Connector routes tend to run close to the edge of shapes, although the particular shapes they brush up against reveal characteristics of the routing algorithm used and are not always predictable. For example, the connector from shape B to C could brush against either shape D (as it does here) or shape A and still have exactly the same path length.

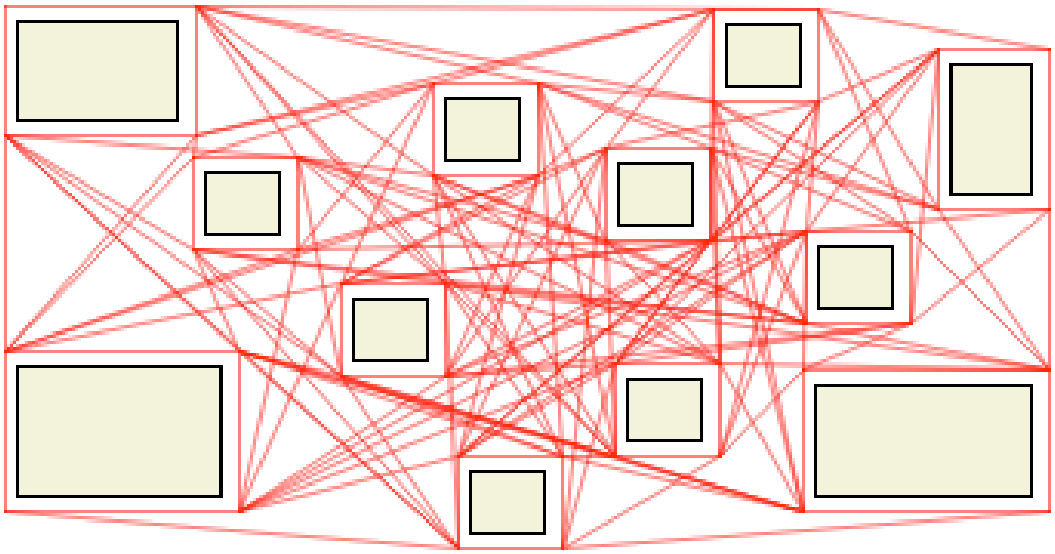


Figure 5.2: A diagram with its corresponding visibility graph overlaid above it. The red lines are edges in the visibility graph. They indicate an unobscured line-of-sight.

corners or ensuring connectors don't overlap unnecessarily when going around the same object vertex.

In this chapter we describe how this three-stage approach to edge routing can be modified to support incremental shortest path poly-line connector routing for diagram editors. This approach is preferable to previous ad hoc methods since it results in more predictable routing without overlap between shapes and connectors. Additionally, the dynamic nature of the approach allows for easy use within interactive applications.

To be useful in a diagram or graph editor we need the routing process to be fast enough for reasonable size diagrams, with up to some hundred objects. The performance requirement for direct manipulation is especially stringent if we wish to update the connector routing during direct manipulation, that is, to reroute the connectors during the movement of shapes, rather than only after their final placement. This requires visibility graph updating and connector re-routing to be performed in milliseconds.

The incremental algorithms presented are fast enough to support this. Two key innovations allowing this are: an *invisibility graph* which associates each object with a set of visibility edges that it obscures (this speeds up visibility graph update for object removal and direct manipulation); and a simple pruning function which significantly reduces the number of connectors that must be considered for re-routing after object removal. In addition we investigate the use of an $A^\star$ algorithm (Hart, Nilsson and Raphael, 1968) rather than Dijkstra's shortest path algorithm to compute optimal paths.

The rest of this chapter is laid out as follows. Section 5.2 provides some background on path planning and visibility graph approaches. Section 5.3 presents our incremental connector routing algorithms. Section 5.4 discusses our implementation of the algorithms in the open-source software library `libavoid`. Section 5.5 presents various techniques that can be used to improve the routings generated by our methods.

5.2 Background

There has been considerable work on finding shortest poly-line paths and shortest orthogonal paths between objects, in part because of the importance of these problems in path-planning for robots and printed circuit board (PCB) layout. See for example Lee (1961), Brennan (1975), Heyns, Sansen and Beke (1980), Sharir (1997), and Mitchell (2000). Most of this research has focused on finding paths given a fixed object layout and has not considered the problem of dynamically moving obstacles and the subsequent need to incrementally update underlying visibility structures.

The most closely related work is that of Miriyala, Hornick and Tamassia (1993) who give an efficient $A^\star$ algorithm for computing orthogonal connector paths. Like us, they are interested in the incremental case. They rely on a *rectangulation* of the graph and previously drawn edges which is essentially a visibility graph. The main difference to our method is that they only consider orthogonal paths. Other differences are that their algorithm is heuristic and routes are not guaranteed to be optimal even if minimizing edge crossings is ignored (see for example Figure 9 of (Miriyala et al., 1993)). They do not discuss object removal and how to maintain optimality of connectors.

There are several well known algorithms for constructing visibility graphs that run in less than the naive $O(n^3)$ approach.[2] Lee (1978) provided an $O(n^2 \log n)$ solution with a rotational sweep. Later, Welzl presented an $O(n^2)$ duality-based arrangement approach (Welzl, 1985). Asano, Asano, Guibas, Hershberger and Imai (1986) presented two more arrangement based solutions both running in $O(n^2)$ time. Another $O(n^2)$ approach was given by Overmars and Welzl (1988) using rotational trees. Ghosh and Mount (1991) showed an output sensitive solution to the problem which uses plane sweep triangulation and funnel splits. It runs in $O(m + n \log n)$ time, where m is the number of visibility edges. Only Lee's algorithm and Asano's algorithm support incremental update of a visibility graph.

Given a visibility graph with m edges and n nodes the standard implementation of Dijkstra's shortest path algorithm is $O(n^2)$ or $O(m \log n)$ if a binary heap-based priority queue representation is used. Fredman and Tarjan (1987) give an $O(m + n \log n)$ implementation using Fibonacci heaps. There are techniques based on the continuous Dijkstra method to compute a single shortest path in $O(n \log n)$ time which do not require computation of a visibility graph (Mitchell, 2000). These methods are more complex and so we chose to use a visibility graph-based approach. In practice we conjecture that they will lead to similar runtime complexity assuming $O(n^2)$ connectors.

5.3 Incremental Poly-line Connector Routing

We support the following user interactions:

- *Object addition:* This makes existing connector routes invalid if they overlap the new object and requires the visibility graph to be updated.
- *Connector addition:* This simply requires routing the new connector.
- *Object removal:* This makes existing connector routes sub-optimal if there is a better route through the region previously occupied by the deleted object. It also requires the visibility graph to be updated.
- *Connector removal:* This is simple—just delete the connector.
- *Direct manipulation of object placement:* This is essentially object removal followed by addition. It obviously requires updates to the visibility graph.

5.3.1 Algorithms

We assume that we have diagram objects (shapes) which have an associated list of vertices making up their perimeter and a list of connector points. For simplicity we restrict ourselves to convex objects and for the purposes of complexity analysis we assume the number of vertices is a fixed constant, that is, say four, since the bounding box can be

[2] The naive approach is: for each pair of vertices on shape boundaries, check to see if the edge between these two vertices intersects each line segment from each obstacle. If it intersects no obstacles, the edge is added to the visibility graph.

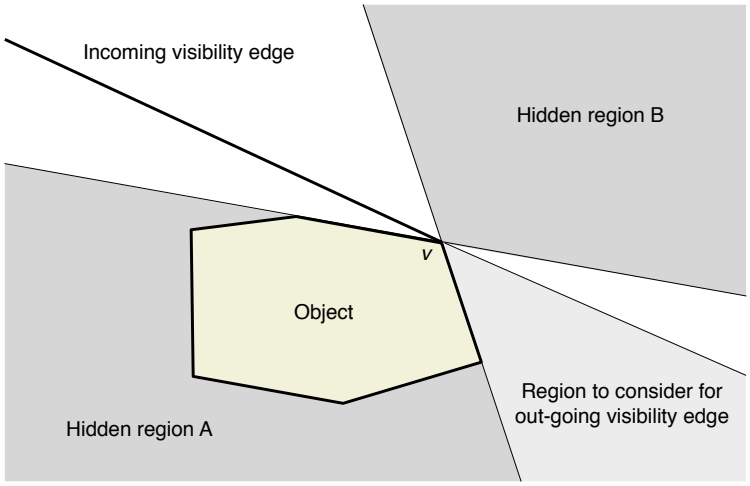


Figure 5.3: Hidden regions which can be ignored when constructing the visibility graph and when finding the shortest path

used for many objects. The algorithms themselves work for arbitrary convex polygons, so circles, for example, can be approximated by a dodecagon.

We also have connectors, these are attached to a particular connection point on an object and have a current routing which consists of a list of edges in the visibility graph. Of course, connectors are not always connected to objects, and may have endpoints which are neither object vertices nor connection points. In this case an extra node is added to the visibility and invisibility graphs for the free endpoint.

The most important data structure is the visibility graph. Edges in this graph are stored in a distributed sparse fashion. Each object has a list of vertices, and each of these vertices has a list of vertices that are visible from it. We treat the connection points on objects as special vertices. They are stored in the visibility graph so they can be used by multiple connectors, but they are only referred to when computing the path for a connector that is attached to that connection point.

Actually, not all visibility edges are placed in the visibility graph. As noted by Rohnert (1986), in any shortest path the path must bend around each vertex it visits. Consider the vertex v with incoming visibility edge shown in Figure 5.3. Clearly in any shortest path, the outgoing visibility edge must be to a vertex in the region indicated. Any other point is either obscured by the shape or would be more directly accessible without bending around the shape at vertex v. Similarly, no shortest path will ever involve an edge from or into either of the "hidden" regions. For this reason, edges between vertices in the hidden regions and vertex v need not be included in the visibility graph.

The other important data structure is the *invisibility graph*. This is a new data structure we have introduced to support incremental recomputation of the visibility graph when an object is deleted. It associates each non-visible edge with a blocking object. This should not be confused with the invisibility graph of (Ben-Moshe, Hall-Holt, Katz and Mitchell, 2004) which simply represents the visibility graph negatively.

The invisibility graph consists of all edges between object vertices which could be in the visibility graph except that there is an object intersecting the edge and so obscuring visibility. Edges in the invisibility graph are associated with the obscuring object and with the objects to which they connect. Thus, each object has a list of visibility edges that it obscures. If the edge intersects more than one object the edge is associated with only one of the intersecting objects.

Connector Addition and Deletion

Connector deletion is simple. All we need to do is to remove the connector and references to the connector from its component edges in the visibility graph.

Connector addition requires us to determine the optimal route for the connector. The simplest approach is use a shortest path algorithm such as Dijkstra's (Dijkstra, 1959). Dijkstra's method has $O(n^2)$ worst case complexity while a priority queue based approach has worst case complexity $O(m \log n)$ where m is the number of edges in the visibility graph and n the number of objects in the diagram.

A potentially better approach is to use an $A^\star$ algorithm which uses the Euclidean distance between the current vertex on the path as a lower bound on the total remaining cost (Mitchell, 2000). We modify the priority queue based approach so that the priority for each frontier node x is the cost of reaching x plus $||(x, v)||$ (the straight-line distance from x to v) where v is the connector endpoint we are computing the path to. In practice we would hope that this is faster than Dijkstra's shortest path algorithm since the search is directed towards the goal vertex v rather than exploring all directions from the start vertex in a breadth-first fashion.

We conjecture that in practice the $A^\star$ algorithm will have $O(n \log n)$ rather than $O(m \log n)$ complexity. The reason is that one would expect an inverse relationship between the *degree of connectedness* of the visibility graph—that is, the average number of objects visible from each object—and the number of edges in the connector paths.

Assume there are n objects and m edges in the visibility graph. If the visibility graph is very dense then m is $O(n^2)$, meaning most objects are visible from the majority of other objects and so connector paths are quite short. In this case the optimal path will be computed in $O(n \log n)$ time, assuming the number of edges in the path is bounded in length by some constant and each vertex has $O(n)$ vertices visible from it. On the other hand, if few objects are visible then the visibility graph will be less dense, say $O(n)$, but connector paths may be long and so again will require $O(n \log n)$ time to compute. Thus, in practice, we expect to compute the connector paths in $O(n \log n)$ time rather than $O(m \log n)$ time.

Object Addition

When we add an object we must first incrementally update the visibility and invisibility graphs, then recompute the route for any connectors whose current route has been invalidated by the addition. The precise steps when an object o is added are:

1. Find the set of edges E_o in the visibility graph that intersect o.

2. Find the set of connectors C_o that use an edge from E_o.

3. Remove the edges in E_o from the visibility graph and place them in the invisibility graph, associating them with o.

4. For each vertex (and connection point) v of o and for each vertex (and connection point) u of each other object in the diagram o' determine if there is another object o'' which intersects the segment (v, u). If there is, add (v, u) to the invisibility graph and associate it with o''. If not add (v, u) to the visibility graph.

5. For each connector $c \in C_o$ find its new route.

The two steps with greatest expected complexity are Step 1, computing the visibility edges E_o obscured by o, and Step 4, computing the visible and obscured vertices for each vertex v of o.

The simplest implementation of Step 1 has $O(m)$ complexity since we must examine all edges in the visibility graph to see if they intersect o. We could reduce this to an average case $O(\log m)$ using a spatial data structure such as a PMR quad-tree (Nelson and Samet, 1986).

Naive computation of the visible and obscured vertices from a single vertex has $O(n^2)$ complexity. However more sophisticated algorithms for computation of the visibility graph have been developed. One reasonably simple approach which appears to work well in practice is Lee's rotational sweep algorithm (Lee, 1978). In this approach, the vertices of all objects are sorted with respect to the angle they make with the original vertex v of o and then these are processed in sorted order. During the sweep the algorithm keeps a list of the open object edges, ordered by their distance from v. Since only the closest edge is visible and obscures those behind it, the algorithm can check fewer points, resulting in $O(n \log n)$ complexity. This is the method we use for determining point visibility.

Object Deletion

Perhaps surprisingly, object deletion is potentially considerably more expensive than object creation. The first stage is to incrementally update the visibility graph.

Assume initially that we do not have an invisibility graph. We first need to remove all edges in the visibility graph that are connected to the object o being deleted. We then need to add edges to the visibility graph that were previously obscured by o. For each vertex (and connection point) v of each object and for each vertex (and connection point) u of some other object in the diagram we must check that (u, v) is not in the visibility graph and that it intersects o. If so we need to check whether there is any other object which intersects the segment (u, v). If there is not then it must be added to the visibility graph.

Identifying these previously obscured edges is potentially very expensive: $O(n^2)$ to compute the candidate new edges and then an $O(n)$ test for non-overlap for each edge of which there may be $O(n^2)$. Thus the worst case complexity of this method is $O(n^3)$.[3]

[3]Based on this one might consider recomputing the entire visibility graph using the Sweep Algorithm since this has $O(n^2 \log n)$ complexity.

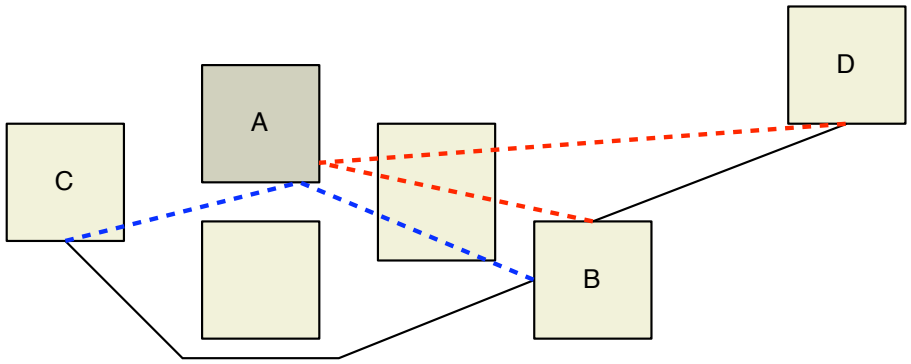


Figure 5.4: Example recomputation of connectors after deleting object A. The connector from B to D does not need to be reconsidered since the shortest path from the connection points via shape A (dotted, red) is clearly longer than the current path. But the connector from B to C needs to be reconsidered (even though in this case it will not move) since the shortest path between its connection points via shape A (dotted, blue) is shorter than the current path.

In order to reduce the expected (but not the worst case) cost we have introduced the invisibility graph. By recording the reason for not including an edge between two vertices in the visibility graph we know almost exactly which edges we need to retest for visibility. More exactly when we remove o we take the set of edges I_o associated with o in the invisibility graph and then test for each of these whether they intersect any other objects. Note that I_o can be expected to be considerably smaller than the candidate edges identified above since an edge (u, v) is only in I_o if it intersects o *and* o was the object first discovered to intersect (u, v).

Thus, although the invisibility graph does not reduce the worst case cost, we can expect it to substantially reduce the average cost of updating the visibility graph. Furthermore, construction of the invisibility graph does not introduce substantial overhead in any of the other operations, the only overhead is the space required to store the edges. Note that when we remove an object we also need to remove edges to it that are in the invisibility graph.

The second stage in object deletion is to recompute the connector routes. This is potentially very expensive since removing an object means that we could have to recompute the best route for all connectors as there may be a better route that was previously blocked by the object just removed.

However, we can use a simple strategy to limit the number of connectors reconsidered. Let A be the region of the object removed and let u and v be the two ends of the connector C. We need only reconsider the current route for C if $\exists y \in A$ s.t. $||(u, y)|| + ||(y, v)||$ is less than the cost of the current route since otherwise any route going through A will be at least as expensive as the current one, for example, see Figure 5.4.

Thus we need to compute $\min_{y \in A} ||(u, y)|| + ||(y, v)||$. Assuming A is convex we can compute this relatively easily. If the line segment (u, v) intersects A then the lower bound is $||(u, v)||$ and we must reroute C. Otherwise, for each line segment (s, t) on the boundary of A we find the closest point y on that segment to segment (u, v). The closest point in A

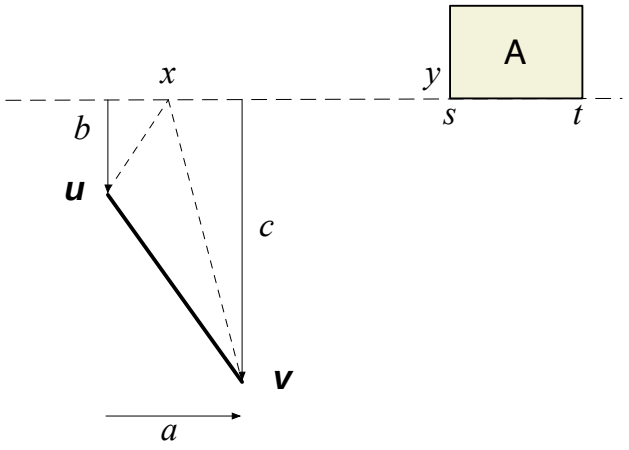


Figure 5.5: Computing the closest point $y \in A$ to the segment (u, v).

is simply the closest of these. Now consider the line segment (s, t). We first compute the closest point x on the infinite length line $\overleftrightarrow{st}$. If x is in the segment (s, t) it is the closest point, otherwise we set y to s or t whichever is closest to x. We can assume without loss of generality that (s, t) is horizontal. Let b and c be the vertical distance from $\overleftrightarrow{st}$ to u and v respectively and a the horizontal distance between u and v, as shown in Figure 5.5. We are finding the value for x which minimizes $\sqrt{x^2 + b^2} + \sqrt{(x - a)^2 + c^2}$. There are two solutions: $x = \frac{ab}{b+c}$ when $b \cdot c \geq 0$ and $x = \frac{ab}{b-c}$ when $b \cdot c \leq 0$. In the case $b = c = 0$, x is any value in $[0, a]$.

Now consider the case when we have determined that there may be a better path for the connector because of the removal. Instead of investigating all possible paths for the connector we need only investigate those that pass through the deleted object. Let A be the region of the object removed and let u and v be the two ends of the connector C and assume that the current length of the connector is c. Requiring the path to go through A means that we can use the above idea to provide a better lower-bound when computing the remaining length of the connector. The priority for each frontier node x is the cost of reaching x plus $\min_{y \in A} ||(x, y)|| + ||(y, v)||$ if the path has not yet gone through A. Furthermore we can remove any node whose priority is $\geq c$ since this cannot lead to a better route than the current one.

Direct manipulation of object placement

The standard approach in diagram and graph editors for updating connectors during direct manipulation is to only reroute the connectors once the object has been moved to its final position. The obvious disadvantage is that the user does not have any visual feedback about what the new routes will be and may well be (unpleasantly) surprised by the result. One of the main ideas behind direct manipulation is that the user should be given visual feedback about the consequences of the manipulation as they perform it rather than waiting for the manipulation to be completed (Shneiderman, 1983). Additionally, it makes the systems seem more responsive, and allows progressive evaluation of the diagram

state. Thus it seems better for diagram and graph editors to reroute connectors during direct manipulation.

We have identified two possible approaches. In the *complete feedback* approach all connectors are rerouted at each mouse move during the direct manipulation. The advantage is that the user knows exactly what would happen if they left the object at its current position. The disadvantage is that this is computationally expensive and may be distracting to the user. For these reasons we have also investigated a *partial feedback* approach in which for intermediate positions we only update the routes for connectors attached to the object being manipulated and leave other connectors alone.

Complete feedback

The simplest way of implementing complete connector-routing feedback is to regard each move as an object deletion followed by object addition. Assume that we move object o from region R_{old} to R_{new}. Then we

1. Find the set of edges I_o associated with o in the invisibility graph which do not intersect R_{new} and remove them from the invisibility graph.

2. For each edge $(u, v) \in I_o$ determine if there is another object $o' \in O$ which intersects the segment (u, v). If there is add (v, u) to the invisibility graph and associate it with o'. If not add (v, u) to the visibility graph.

3. Find the set of edges E_o in the visibility graph that intersect $R_{new} \setminus R_{old}$ but are not from o.

4. Find the set of connectors C_o that use an edge from E_o.

5. Remove the edges in E_o from the visibility graph and place them in the invisibility graph, associating them with o.

6. For each vertex (and connection point) u of o and edge (u, v) in the invisibility graph check that the object o' associated with the edge still intersects it. If it does not, temporarily add the edge to the visibility graph.

7. For each vertex (and connection point) u of o and edge (u, v) in the visibility graph check if there is another object $o' \in O$ which intersects the segment (u, v). If there is add (u, v) to the invisibility graph and associate it with o'. If not, keep (u, v) in the visibility graph.

8. For each connector $c \in C_o$ find its new route.

9. For every connector not in C_o determine if there is a better route through $R_{old} \setminus R_{new}$.

Note that in the above we can conservatively approximate the regions $R_{new} \setminus R_{old}$ or $R_{old} \setminus R_{new}$ by any enclosing region such as their bounding box.

Partial feedback

While we have the ability to reroute all connectors while a shape is being dragged through the diagram, this might not always be ideal. Immediate feedback like this is desirable in the sense that it adds consistency and continuously represents the state of the system during dragging. However, immediate feedback results in a lot of extra movement which can cause unnecessary distraction. Also, users will typically move shapes through many intermediate positions without intending to permanently place them there. It is unlikely they care how routing would be affected if the shape had been placed at these intermediate positions.

A solution is to show only partial feedback which shapes are being dragged in the diagram. In this case, only connectors attached to the dragged objects are rerouted during the dragging action. At the beginning of the action, the dragged shapes are removed as obstacles from the visibility graph, so all connectors that were routing around them are rerouted. Then, when the drag action is completed, they are added back into the visibility graph, and the connectors they overlap at their final destination are rerouted around them.

We implement partial connector-routing feedback by performing object deletion once the object o has moved and then at each step during the movement action we:

1. Compute the vertex and connector points which are visible from a vertex of o and temporarily add these to the visibility graph

2. Recompute shortest routes for all connectors to/from o

Once the move has finished we perform object addition for o. Clearly this is substantially less work than required for complete feedback.

This partial feedback behaviour is very much like lifting up the objects above the diagram while they are being dragged, and then placing them back down at their final destination. To strengthen the metaphor, we show shadows for the selected objects as they are picked up and dragged. The behaviour here should be more familiar and predictable to users since it maps closely to real world interaction.

A variation on this idea (that we haven't yet explored) is to have shapes drop back down to the page and cause rerouting during the drag operation if they are not moved for some amount of time. This would allow the user a form of progressive evaluation, where they can hover the shapes at a potential final placement position for a moment to see what effect it would have on routing. This needn't interfere with the existing partial feedback but can instead compliment it.

Efficient multi-shape movement

When multiple shapes are moved at the same time, rather than just deleting and adding each one in turn, the visibility graph can be updated more efficiently by queuing the events and processing them in a specific order. We maintain a list of moved objects and then process this list in the following way:

1. For each moved shape, remove that shape's vertices from the visibility graph.

2. Check all edges in the invisibility graph that were blocked by the moved shapes, taking into account the new positions of these shapes as obstacles.

3. For each moved shape, check all visibility edges to see if this shape blocks them, and then calculate visibility for the vertices of this shape.

5.3.2 Evaluation

We have implemented our incremental connector algorithms in Dunnart and have conducted an experiment to evaluate the techniques. It should be noted that this evaluation examines the speed of these algorithms, rather than their usability. Previous empirical evaluation (e.g. (Purchase et al., 2002; Ware et al., 2002)) gives us a good idea of desirable properties for connectors in network diagrams, that is, short paths, minimum number of bends, minimum amount of bendiness and few connector crossings. Thus, we felt that it was important to provide a fast method that could continuously generate connector routes with such properties in interactive applications. We have received informal feedback and feature requests from users of our connector routing implementations which we present in Section 5.4.3.

The version of Dunnart used for this evaluation was compiled with `gcc` 3.2.2 at `-O3`. We ran Dunnart on a Linux machine (glibc 2.3.3) with 512MB memory and a Pentium 4, 2.4GHz processor.

In our experiment we compared the Spline-o-matic (SoM) connector routing library of GraphViz (which is non-incremental) with a static version (Static) of our algorithm in which the visibility graph and connector routes are recomputed from scratch after each editor action, and the incremental algorithm (Inc) given here with various options.

The experiment used various sized grid arrangement of boxes, where each outside box is connected to the diagonally opposite box by a connector and each box except those on the right and bottom edge is connected to the box directly down and right. Figure 5.6 shows an example layout for a 6x6 grid. For an $n \times n$ grid we have n^2 objects and $2(n-1)+(n-1)^2$ connectors. We also used a smallish but more realistic diagram **bayes** from Woodberry (2003) (a Bayesian network with 35 objects and 61 connectors) shown in Figure 5.7. The experiments were: for each object to delete it from the grid and the add it back in. We measured the time for each deletion and each addition giving the average under the **Add** and **Delete** rows. We have separated the results into the time manipulating the visibility graph (*Vis*) and the time for performing all connector (re)routing (*Paths*). We also measured the average time taken to compute the new layout for each mouse position when moving the marked corner box through and around the grid as shown in Figure 5.6. A similar movement was performed for the **bayes** diagram, using the top leftmost box. These results are is given in the **Move** rows.

Both our static (Static) and incremental (Inc) version use the $A^\star$ algorithm to compute connector paths and give complete feedback. The incremental version computes the invisibility graph while the static one does not since this is not needed. We also give times for versions of the incremental algorithm which do not construct the invisibility graph (Inc-noInv) and use Dijkstra's shortest path algorithm rather than the $A^\star$ algorithm

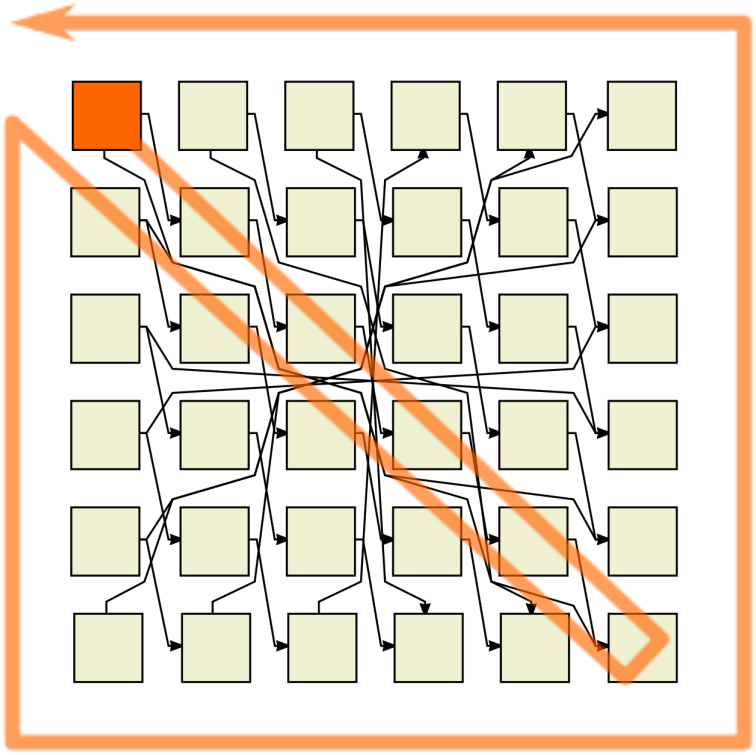


Figure 5.6: 6x6 Grid layout, showing the path taken through grid for **Move** experiment.

(Inc-noA*) for routing. In the **Move** sub-experiment, the incremental (Inc) version uses complete feedback. We also give times for the incremental algorithm using partial feedback (Inc-par) rather than complete feedback for the **Move** experiment.

The results are shown in Table 5.1, for grids of size 6, 8, 10 and 12, and **bayes**. A "—" indicates that the approach failed to complete the total experiment in three hours.

We can see from the table that the static version of our algorithm is considerably faster than Spline-O-Matic. The incremental versions are orders of magnitude faster than static algorithms. While the incremental version is usable for direct manipulation with complete feedback at least until **grid10** (and with difficulty at **grid12**), the static algorithms become unusable at **grid08**. The results show how important incremental recomputation is. The importance of the invisibility graph is clearly illustrated by the difference between Inc-noInv and Inc, improving visibility graph recomputation by orders of magnitude for **Delete** and **Move**. The $A^\star$ algorithm gives around 50% improvement in path re-routing. Partial feedback reduces the overhead of movement considerably, particularly as the number of connectors grows.

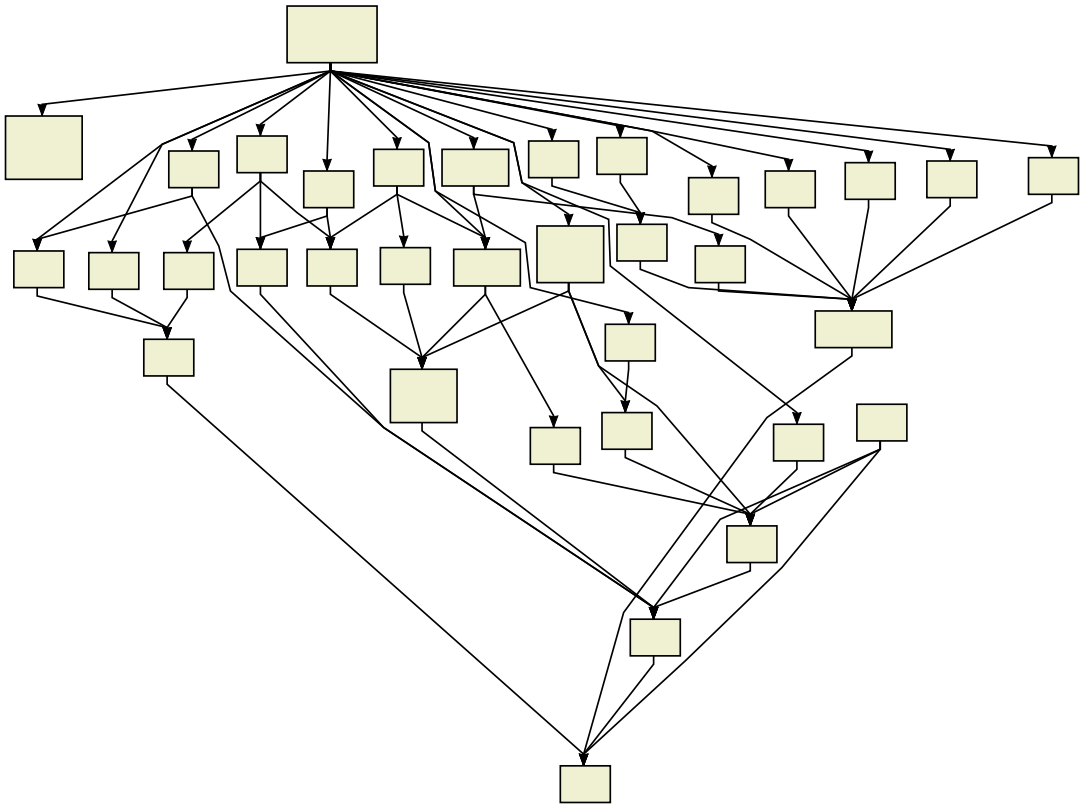


Figure 5.7: Layout of the **bayes** diagram

5.4 `libavoid`: Connector routing library

We separated out our initial implementation of the techniques described in Section 5.3 into a standalone C++ software library. This library, `libavoid`,[4] is available under the open-source GNU Lesser General Public License (LGPL), allowing it to be flexibly reused.

In addition to being used by our diagram editor, Dunnart, this library has been integrated into the popular open-source vector graphics editor, Inkscape. It has also been used by several other individuals (that we are aware of) for their own applications. One of these is for possible use in a future metabolic pathway viewer to be part of the Utopia toolset (Pettifer, Sinnott and Attwood, 2004).

5.4.1 `libavoid` interface

Here we present `libavoid`'s interface, by way of some basic example code.

First, we must create an instance of the router. Our program will probably need to have a separate instance of the router for each individual document it has open.

```
Avoid::Router *router = new Avoid::Router();
```

To add a shape (obstacle) to the router, we first create a `ShapeRef` by giving a polygon representing the convex hull boundary of the shape and an ID for the shape.

[4] libavoid, Michael Wybrow, `http://adaptagrams.sourceforge.net/`

		grid06		grid08		grid10		grid12		bayes	
Op	Algorithm	Vis	Paths	Vis	Paths	Vis	Paths	Vis	Paths	Vis	Paths
Add	SoM	152	198	752	881	2449	2831	6669	1166	122	284
	Static	40	67	154	313	475	1024	1209	1064	29	87
	Inc-noInv	0	13	8	90	15	304	14	761	0	1
	Inc-noA*	0	11	9	61	18	195	16	347	0	7
	Inc	0	1	9	20	18	58	16	138	0	0
Delete	SoM	146	185	724	853	2385	2779	6542	1149	110	269
	Static	40	64	153	296	463	1003	1186	1042	29	80
	Inc-noInv	53	16	266	87	1006	298	1146	749	77	3
	Inc-noA*	2	38	17	223	49	734	55	1331	7	9
	Inc	2	8	18	58	48	204	55	504	8	0
Move	SoM	149	188	742	863	—	—	—	—	114	282
	Static	31	69	156	310	461	1004	1214	1026	29	79
	Inc-noInv	43	15	230	80	950	289	1167	708	80	0
	Inc-noA*	0	25	12	102	34	261	37	449	7	8
	Inc	0	10	12	28	34	77	37	213	7	0
	Inc-par	0	3	10	7	26	20	28	11	0	3

Table 5.1: Average visibility graph and connector routing times (in msec.)

```
// Create the ShapeRef:
Avoid::Polygn shapePoly = Avoid::newPoly(3);
shapePoly.ps[0] = Avoid::Point(1, 1);
shapePoly.ps[1] = Avoid::Point(2.5, 1.5);
shapePoly.ps[2] = Avoid::Point(1.5, 2.5);
unsigned int shapeID = 1;
Avoid::ShapeRef *shapeRef = new Avoid::ShapeRef(router, shapeID, shapePoly);
// ShapeRef constructor makes a copy of polygon so we can free it:
Avoid::freePoly(shapePoly);
```

To actually add the shape to the router (and cause connectors it intersects to be rerouted) we do the following:

```
router->addShape(shapeRef);
```

Conversely, to delete a shape from the router (and reroute connectors that now have a better path) we can do the following. Note: `delShape()` frees the `ShapeRef` passed to it.

```
router->delShape(shapeRef);
```

To move or resize a shape already known by the router:

```
router->moveShape(shapeRef, &newPolygon);
```

In its default mode the router will queue multiple shape movements and perform the changes to the visibility graph in an optimised order. Thus we make several calls to `movShape` for different shapes and then tell the router to process the moves. This tends to be useful in interactive applications where the user may drag a group of shapes at once.

```
router->moveShape(shapeRef1, newPolygon1);
router->moveShape(shapeRef2, newPolygon2);
router->processMoves();
```

Alternatively, we can make the router perform the operations immediately when `movShape` is called if the following option has been set previously:

```
router->ConsolidateMoves = false;
```

To add a new connector to the router:

```
unsigned int connID = 2;
Avoid::Point srcPt(1.2, 0.5);
Avoid::Point dstPt(3, 4);
Avoid::ConnRef *connRef = new Avoid::ConnRef(router, connID, srcPt, dstPt);
connRef->updateEndPoint(Avoid::VertID::src, srcPt);
connRef->updateEndPoint(Avoid::VertID::tar, dstPt);
```

To remove a connector from the router:

```
connRef->removeFromGraph();
// Delete the shapeRef if its no longer needed.
delete connRef;
```

We can set a function to be called when the connector needs to be redrawn. When called, this function will be passed the pointer given as a second argument to `setCallback`:

```
void connCallback(void *ptr)
{
    Avoid::ConnRef *connRef = (Avoid::ConnRef *) ptr;

    printf("Connector %u needs to be rerouted!\n", connRef->id());
}

connRef->setCallback(connCallback, connRef);
```

The callback will be triggered by movement, addition and deletion of shapes. Adjusting the connector endpoints will not trigger it, since if we're altering the endpoints it's assumed that we probably want to control when the rerouting is performed. The following will check if a connector needs to be rerouted:

```
if (connRef->needsReroute()) ...
```

If you want to trigger the callback for a connector after moving its endpoints (or when it is first created) this can be done via:

```
connRef->handleInvalid();
```

To generate a new path for a connector we do:

```
connRef->generatePath();
Avoid::PolyLine route = connRef->route();
for (int i = 0; i < route.pn; ++i)
{
    printf("%f, %f\n", route.ps[i].x, route.ps[i].y);
}
```

We can update the endpoints of a connector with:

```
Avoid::Point dstPt(6, 3);
connRef->updateEndPoint(Avoid::VertID::tar, dstPt);
```

5.4.2 Inkscape integration

Inkscape[5] is an open-source vector graphics editor. It is primarily an illustration tool, with similar characteristics to Adobe Illustrator or CorelDRAW, though it is gaining more general diagramming features. It is widely used, with over 150,000 downloads a month.[6]

Using `libavoid`, we implemented a connector tool within Inkscape that supports auto-routing poly-line connectors. These connectors can be attached to shapes, that is, anything that is not an open path. When a shape is moved the connector will update itself to follow the shape. In addition, connectors will avoid objects marked as "avoided" and reroute if one of these objects is placed on their existing path, or is moved and frees up a better path.

Shapes can be marked as either "avoided" or "ignored" for connector routing. This is done via two toolbar buttons, available while in connector mode. The first of these marks all the objects in the current selection as avoided for connector routing. The second button marks all objects in the current selection as ignored.

Currently, only the bounding boxes of shapes are used for avoidance. The preferred approach would be to use each shape's convex hull. Unfortunately, Inkscape's `NR::ConvexHull` code only generates a rectangular bounding box. Once proper convex hull code is in the code-base, connector routing will be switched to use that.

While in connector mode, new connectors can be drawn by clicking and dragging from any empty point on the canvas. If you begin or end the connector over a connection point, then the connector will be attached to that shape. Connection points are currently shown at the centre (of the bounding box) of any shape, and are shown when the mouse cursor is hovered over the object in connector mode.

If you select a connector while in the connector mode, a handle will be shown for each of its endpoints. Dragging either of these will alter the endpoint of the connector. If the endpoint was attached to a shape then this connection will be removed. If this dragging action ends over a connection point, then that end of the connector will be attached to the shape. The dragged connectors will automatically reroute during this action.

[5] Inkscape 0.45.1, Inkscape developers, `http://www.inkscape.org/`

[6] `http://sourceforge.net/project/stats/index.php?group_id=93438&ugn=inkscape`

If an "avoided" shape is placed over an existing connector route, the connector will automatically reroute to avoid the shape. Also, if moving or deleting a shape frees up a better (currently meaning shorter) path then the connector will again be rerouted.

Moving a connector detaches it from the shapes it is connected to. Connectors moved as part of a selection will stay attached to other objects in the selection, rather than becoming detached from them.

By default, the Connector tool will not attach connectors to text objects. There is a checkbox in the connector tool preferences to control this setting.

The margins around avoided shapes can be adjusted via the "Spacing" control on the Connector toolbar.

5.4.3 Feedback

Informal feedback has been collected from `libavoid`'s use in both Dunnart and Inkscape. This feedback comes from around twenty-five different users, and was gathered from bug reports and feature requests on Inkscape's bug tracker website, as well as through e-mail sent to the author. The feedback from Inkscape users has been overwhelmingly positive. Many stated that the connector tool provided by `libavoid` has turned Inkscape into a usable diagramming tool. However, there were several requested features from users.

Inkscape's connectors attach only to the centre of shapes' bounding boxes. Users have suggested that user-specified connection points be implemented, where additional connection points could be added to shapes and be placed freely. There is no reason why this feature cannot be implemented—Dunnart shapes have their own connection points at different positions on their border, though these cannot be freely placed by the user. There are some questions about the user interface that should be used by the user to create these new connection points as well as subsequently editing them. Probably these points should be confined to be placed within the shape they are attachments for, and they should move relative to the width and height of the shape as it is scaled. We will likely add this feature to a future version of Inkscape.

A frequently requested addition to Inkscape's connector tool has been other connector styles, specifically curved connectors and orthogonal connectors. Several people stated that they had themselves manually added curved corners to the poly-line connectors produced by `libavoid`, to get a more visually pleasing diagram. We have a method for automatically drawing curved approximations of our poly-line connectors, described in the next section. This is implemented in Dunnart but not yet in Inkscape. Orthogonal connectors can obviously not be generated with the current style of visibility graph, though it might be possible to generate them efficiently with a similar visibility structure and techniques to maintain it. This is a subject for further investigation.

Another commonly requested feature, by both Inkscape and Dunnart users, has been user-specified bend points for connectors. Such a feature would allow users to select one or more specific points that a connector must be routed through. There is no technical difficulty in accomplishing this. The user-specified point can be added to the visibility graph and its route determined as if it were two connectors—one from the connector's

source point to the user-specified point, and another from this point to the connector's endpoint.

There are, however, questions about the desired behaviour of such user-specified bend points. Should they be able to be aligned with other objects? Should they drift or stay fixed? How are users planning to use them? For example, do users really just want to restrict a connector to route around the left side of a particular shape? In this case, maybe the specified point should actually be a vertex on a shape that would be moved as the shape moves. If so, the user may not want the connector to still route via that point on the shape if the shape is subsequently moved far away. User-specified connector bend points will likely be added to a future version of `libavoid` after further investigation into their potential use.

A final suggestion from Inkscape users was the ability to alter the padding distance around avoided shapes. This is already possible on a per-shape basis, but they requested it be available on a per-connector basis. This is obviously not possible with our current model, where every connector must use the same visibility graph with shape padding distances already encoded into it. It could be achieved using a separate visibility graph for each padding distance, though this would add considerable computational overhead.

5.5 Improved connector routing

In this section we demonstrate the flexibility of our connector routing technique by presenting some of the ways we have extended the original `libavoid` implementation to improve the routes it produces. These changes have been motivated by our own experience using `libavoid`() and from research into the aesthetic properties of connectors that are most important for diagram readability.

5.5.1 Capturing additional routing aesthetics

`libavoid` can produce more aesthetic connector routings by adding penalties to its cost function for undesirable properties. By default, the cost function considers only distance, resulting in routings with shortest length. The cost function can be altered to consider other properties of connector routes such as crossings or number of segments.

Cost function

The penalty for a connector route is simply the sum of the penalties for its segments where the penalty for an individual segment is given by:

$$cost = l + \alpha s + \frac{\beta a \log(a+1)}{10} + \gamma scc \tag{5.1}$$

where: α, β, γ are user specifiable penalties for, respectively, the segment penalty, the angle penalty and the crossing penalty, l is the length of the segment and a is the angle away from a straight line that this segment makes with the previous segment, scaled to the range $0 \leq a \leq 10$. Therefore $a = 0$ means the two segments make a straight line. If

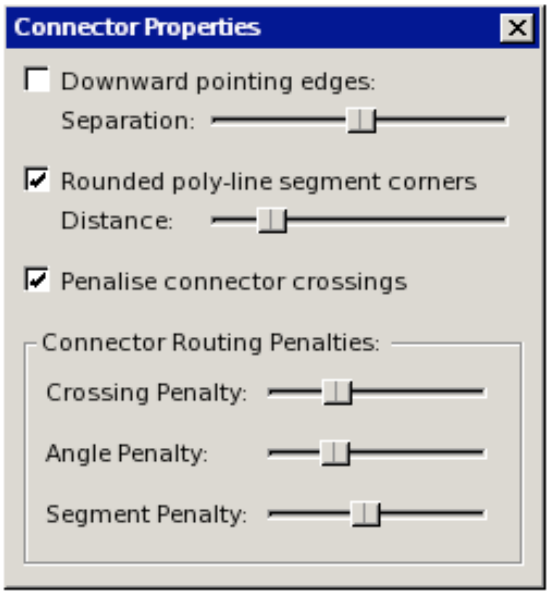


Figure 5.8: Dunnart's Connector Properties dialog, which allows interactive adjustment of routing penalties.

this is not the first segment and $a > 0$, then $s = 1$, otherwise $s = 0$. Finally, *scc* is the number of crossings for the segment. Computing *scc* is discussed below.

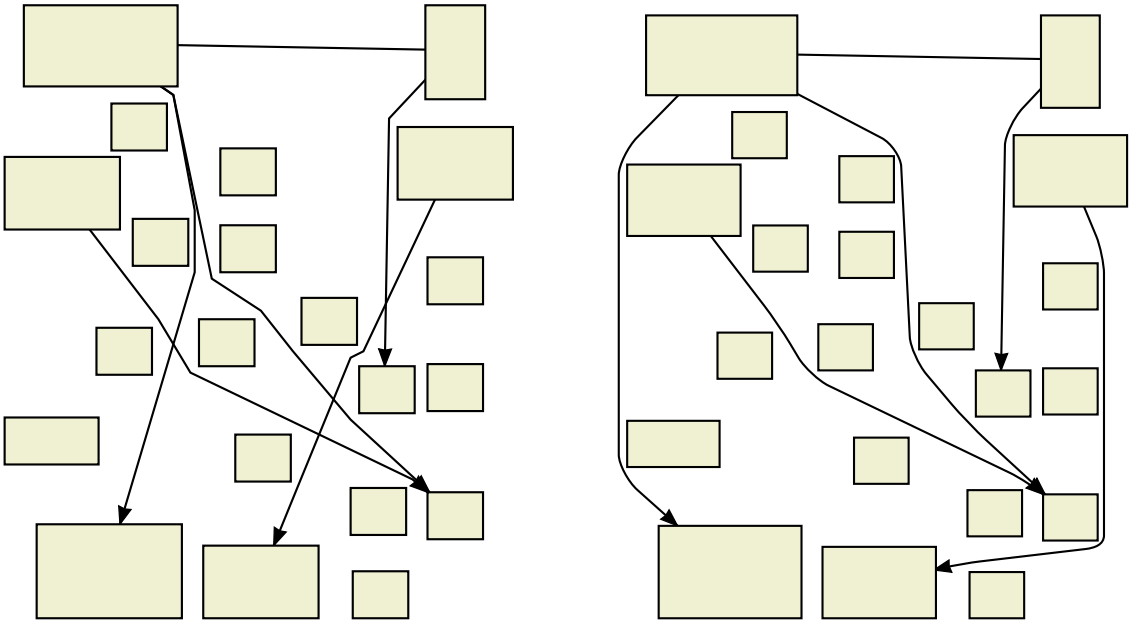
The penalty function incorporates three main features of poly-line connector routing that have been shown to affect user comprehension: connector length, number of bends and degree of bendiness (Ware et al., 2002). The ability to use a flexible penalty function allows us to adjust the initial routing to cater for our desired combination of aesthetic criteria and their trade-offs.

Currently we do not have any of the penalties enabled by default. The user can alter any of them through the penalty sliders in Dunnart's Connector Properties dialog, shown in Figure 5.8. The connector routes in the diagram are updated live as the penalties are altered through these sliders. In this way the user can immediately see the effect and easily find the penalty values most appropriate for the diagram.

Non-zero penalty values could be set by default, but while there has been some research showing comparisons between aesthetic trade-offs, these are not conclusive for all diagram types or all editing tasks. For example, Ware et al. (2002) show that roughly 38° of connector continuity is equivalent in cognitive cost to a single crossing, though this is for paths made up of multiple connectors, each a single straight line, where the user is determining the shortest path between two specified nodes.

we have considered an additional penalty on the direction of segments; this may be useful in a tree or similarly directed diagram to ensure that all connectors flow in the correct direction and don't loop back around other objects.

It should be noted that when using routing penalties we are no longer simply determining the shortest path through the visibility graph. As a result, the simple strategy described in Section 5.3.1 for reducing the set of connectors we must consider rerouting



(a) Initial connector routing (b) Penalised connector routing

Figure 5.9: Penalties can be added to the connector routing cost function to produce routings with more pleasing aesthetic properties, such as fewer segments, less bendiness or fewer crossings. As a further aesthetic improvement, corners of poly-lines can be drawn as smooth curves.

when a shape is deleted becomes less effective. The strategy remains useful while the majority of the path cost is due to distance, rather than routing penalties.

Reducing connector crossings

We have also investigated using a simple greedy heuristic to reduce the number of connector crossings present in the diagram. Each connector with crossings is considered once, in decreasing order of crossings. Again we use an $A^\star$ based-search to find the best route for each connector, though this time the routing cost includes a penalty for each crossing. The computations cost of routing each connector taking into account connector-crossings is currently $O(|S|(|E| + |V|) \log |V|)$ where $|E|$ is the number of edges in the visibility graph and $|S|$ the number of segments in the routed connectors.

Alternatively, a hill-climbing technique could be employed where the possible path improvements to all connectors with crossings are considered. The best choice is then re-routed and the process repeated. This may result in better initial connector layout, though the computational cost is prohibitive without doing some form of caching for path costs.

Finding poly-line connector crossings for a graph is not as simple as just determining the intersections between the segments of all connector paths. It is common for connectors to bend around the same shape corner or to share paths for part of their route, that is, running along the same paths in the visibility graph. In these cases we want to distinguish between situations where they cross and where they only touch or run parallel. We do

```
procedure crossingCount(A, B)
    crossings ← 0
    for each segment (a1, a2) in connector A do
        for each segment (b1, b2) in connector B do
            if (a1, a2) and (b1, b2) are the same line segment then
                # We notice crossings for shared paths when they diverge, so skip.
                continue
            else if (a2 = b2) or (a2 = b1) or (a1 = b2) then
                # The segments can share endpoints in four ways. We ignore three.
                continue
            else if a1 = b1 then
                    # The segments share an endpoint
                    a0 ← point on previous segment (a0, a1) of A if it exists
                    b0 ← point on previous segment (b0, b1) of B if it exists
                    if a2 = b0 then
                        # In terms of A, the shared path is ahead. Skip it.
                        continue
                    else if (a0 = b0) or (a0 = b2) then
                        # In terms of A, we are leaving a shared path.
                        if A and B are connected to the same node then
                            # Don't want to the count case where connectors route from or to a common endpoint.
                            continue
                        endif
                        # Note: the shared portion of the path can go in the forwards or backwards direction along B.
                        bL ← last shared point on B (will be b0 or b2)
                        endSide ← cornerSide(a0, a1, a2, bL)
                        Scan backwards (or forwards) through A and B until reaching the beginning of the shared path.
                        aS ← point on A leading into the shared path.
                        bS ← point on B leading into the shared path.
                        aS2 ← the first point along the shared path.
                        aS3 ← the second point along the shared path.
                        startSide ← result of cornerSide(aS, aS2, aS3, bS)
                        if startSide ≠ endSide then
                            # The shared path contains a crossing.
                            increment crossings
                        endif
                    endif
            else
                if segments (a1, a2) and (b1, b2) intersect then
                    # Standard line segment intersection.
                    increment crossings
                endif
            endif
        endfor
    endfor
    return crossings

procedure cornerSide(c1, c2, c3, p)
    # c1, c2, c3 is a line C with two segments.
    if point p is to the left of line C then
        return 1
    else if point p is to the right of line C then
        return -1
    endif
    return 0
```

Figure 5.10: Poly-line connector crossing detection algorithm.

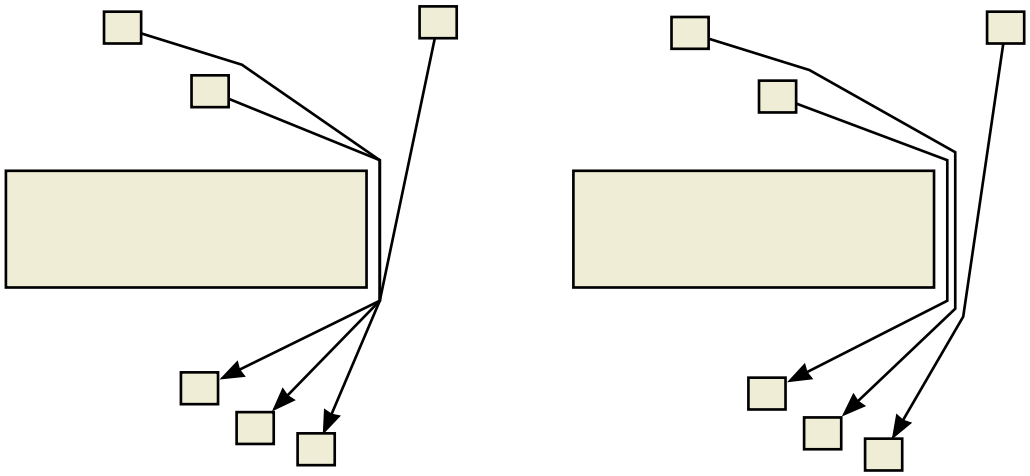


Figure 5.11: An exaggerated example of the nudging we perform on shared connector bend points.

this by comparing the order of connectors entering and leaving shared paths or common bend points. The algorithm used for this purpose is described in Figure 5.10.

Since the length of the shared path has no effect on whether the two connectors cross, we can treat bend points equivalently to shared paths. We start by looking for cases where the segments of two connectors have a single shared endpoint. This is the beginning of a shared path or a common bend point. Such bend points always pass around the corner of a node, so we determine the order of connectors entering the shared path at this point by finding which connector runs closest to the node. From here we follow the common segments along the shared path until they diverge again. We then determine the order of connectors leaving the shared path, taking into account features of the bends such as the winding directions. If the two orders are different then we can tell that the connectors cross along the shared path, rather than running parallel.

Setting a very high penalty for crossings will produce routings with few crossings but this is not always ideal. By reducing the penalty we produce more pleasing routings, where crossings will be avoided where possible, but not at the expense of other desired aesthetic criteria. We may wish to allow a connector crossing if removing it would result in a very long connector path or many extra segments.

5.5.2 Nudging shared paths

Using the information from the connector crossing detection algorithm, line segments can be adjusted to slightly separate connectors routed around the same corner of a shape. This involves nudging the bend points of connectors along shared paths, and at the points they cross or touch, as shown in Figure 5.11. To do this a sorted order for each of these points and shared paths is kept when determining crossings. This nudging step is useful for making the diagram easier to comprehend, since the user will be more able to determine the paths of connectors that share a common sub-route.

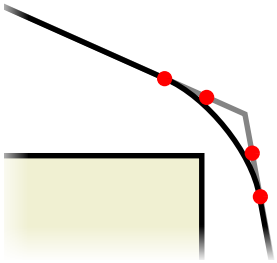


Figure 5.12: A poly-line connector drawn with a Bézier curve at the corner between segments. The control points for the Bézier curve are shown in red.

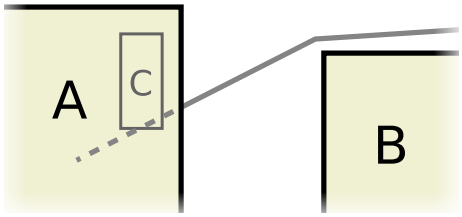


Figure 5.13: A connector path from Shape A to the corner of Shape B. If Shape C is present then it will block this visibility edge, making the path invalid. Even though the path is only drawn from the edge of Shape A, it is computed from the shape's centre.

5.5.3 Approximating curved connectors

Several Inkscape users have expressed a desire to draw poly-line connectors with curved bends. We do this in Dunnart by smoothing the corners of the poly-line using Bézier curves (Bartels, Beatty and Barsky, 1998). Each segment is shortened at each end by a fixed distance n and a Bézier curve is inserted at the corner with control points at the two shortened segment endpoints and at an equal distance $(n/2)$ between these points and the original corner point, as shown in Figure 5.12.

Since the curve will run inside the original corner, as long as n is fairly small the curve will be in the buffer space around shapes and will not cause the connector to intersect any shapes. In any case, the global curving amount n is adjustable via a slider in the interface as shown in Figure 5.8, allowing it to be interactively adjusted by the user. Figure 5.9(b) shows an example Dunnart diagram where poly-line connectors are drawn with curved corners.

5.5.4 Containment and cluster routing

When using connectors attached to the centre of shapes, or any connection point inside the perimeter of the attached shape, there are particular cases that can result in strange routing. Obviously when calculating visibility for these "contained" connection points, the containing shape must be ignored or else the point will effectively have no visibility and paths will be unable to escape from the shape. Also, since the connector is only drawn from the edge of the shape, we don't want to consider objects inside the shape as

obstacles that obscure this visibility edge, see Figure 5.13. We address this issue by only considering the portion of the visibility edge outside of the shape (i.e. the drawn portion) when determining collisions with obstacles.

As an aside, consider that some types of diagrams have the requirement that a connector flows from a particular face of a shape, for example, its right side. Such routing can be achieved simply by ignoring just that single face of the shape for visibility purposes but leaving the remaining faces as obstacles.

A similar problem is routing connectors in a diagram with clusters. Clusters are objects that contain other diagram objects, or even whole sub-diagrams. Clusters are often drawn as a rectangular or convex polygonal boundary around the contained objects, for example, see Figure 1.3. We would like connectors outside of the cluster to route around clusters as if the clusters were normal shapes. We would like connectors within the cluster to be routed so that they do not leave the cluster. Connectors that connect objects within the cluster with objects outside the cluster should be able to cross the cluster boundary once.

While this has not yet been implemented in `libavoid`, the desired cluster routing behaviour can be obtained by making the following changes to the routing techniques described earlier. First, differentiate between normal shapes and cluster shapes within `libavoid`. Second, for each connector endpoint, keep a sorted list of the clusters it is contained within, ordered from closest to furthest away. Third, when computing visibility edges, count edges as visible if they are obscured only by edges forming cluster boundaries, but store with them a sorted list of clusters they cross. Fourth, when computing the path of a connector, determine the set of cluster boundaries it must route through by finding the closest common containing cluster for each endpoint. This gives an order of cluster boundaries that the connector must pass through and can be used to determine whether the extra visibility edges should be attempted for this route. The connector must route out through all lower clusters to the common cluster (which may even be the page itself at the highest level) and then back down through the lower clusters to the other endpoint.

This technique could also be used for the original problem of attached endpoints being contained within shapes that might also contain obstacles. Unfortunately this turns every shape with an active centre connection point into a cluster. Consequently, this would lead to many more edges being placed in the visibility graph, each likely to have a large lists of shapes (clusters) that blocks it. This approach also means extra housekeeping and complexity in maintaining the visibility graph from additions and deletions of both shapes and connectors.

5.6 Conclusions

While most existing diagram editors provide some form of automatic connector routing, they generally don't cater well for users due to the ad hoc manner in which connectors are updated and unpredictable routes produced. Also, many do not take into account obstacles and thus do nothing to prevent shapes and connectors from overlapping.

In this chapter we have presented a fast incremental algorithm for maintaining a visibility graph for use in interactive environments. This allows us to generate object-avoiding

poly-line connectors that update automatically as obstacles are added to the diagram. These connectors are automatically improved as better routes become available due to removal of shapes from the diagram. These algorithms have been designed to support user interaction. They offer a fast partial feedback mode with less distracting interactive dragging of shapes.

Our technique produces shortest path routes for connectors. The routing is deterministic: starting with the same diagram layout will always result in the same connector routes. We believe that this results in connector routes being predictable and easy for users to understand. The algorithm is flexible enough to allow several useful extensions. The user can add penalties to control particular routing features, allowing more aesthetically pleasing routings to be produced. Two drawing modifications—shared path nudging and curved connector corners—increase comprehensibility of the generated routings.

The routing algorithm and extensions are implemented as the open-source software library `libavoid`, already integrated into a popular vector graphics editor.

Future work could improve clustered routing support in `libavoid` as well as investigating techniques for incremental routing of orthogonal object-avoiding connectors. Usability evaluation of the connector tool's use could focus on questions of how user may want to further influence connector paths, such as via user-specifiable bend points.

Chapter 6

Interactive network layout

6.1 Introduction

Network diagrams are a popular method for graphically communicating relational data. Some widely known examples of network diagrams are social networks, flow charts, organisational charts, mind maps, family trees, UML diagrams, circuit diagrams and biological networks. Tidy layout is important for network diagrams since features of layout such as overlapping objects or connector crossings impact negatively on the networks readability. Producing good layout of such diagrams is not easy; it is tedious to author manually and computationally intractable.

As previously discussed, there has been decades of research into techniques and algorithms for generating good network layouts as part of the field of graph drawing (Di Battista et al., 1999). There has also been investigation of incremental methods for the dynamic graph layout of changing graphs (Branke, 2001). However, the use of these graph drawing techniques for interactive diagramming applications where the layout is a collaboration with the user, is largely unexplored. Most interactive diagramming tools provide very little support for automatic layout of network diagrams. Where they do, they provide tools that make a single pass over the diagram when run and produce a completely new layout. These tools largely ignore modifications that the user has manually made to the layout of the diagram.

As we saw in Section 2.4.7, graph drawing algorithms have been added to Microsoft Visio via Tom Sawyer's Layout Assistant add-on. Also, the demonstration editor yEd includes several types of automatic network layout, as does the commercial diagram editor OmniGraffle. In each of these cases the graph layout components are not well integrated into the rest of the editor and feel like they have been added as an afterthought. Part of the problem is that standard approaches to graph layout are not well suited for use in interactive applications. This is true for once-off techniques that take the structure of the graph, some algorithmic parameters and produce a new layout for the graph. While individual algorithms may capture certain kinds of drawing conventions in their layout, they lack the flexibility to be able to encode the range of differing styles that a user may wish to apply to their diagrams. More importantly, they are unable to be used in a collaborative sense; they do not take into account manual position changes made by the

user or they have no notion of—and thus can't maintain—user-specified constraints of the kind we have explored in previous chapters.

In this chapter we investigate the use of *constrained stress majorization* (Dwyer, Koren and Marriott, 2006) for providing well-integrated graph layout for interactive diagram editors. Constrained stress majorization extends the stress majorization approach to force-directed layout by allowing *separation constraints* in each dimension.[1] These separation constraints allow us to represent user-specified placement constraints such as alignment and distribution, as well as representing stylistic constraints to capture various common diagram drawing conventions. In addition, constrained stress majorization is both incremental and has robust convergence behaviour, making it an ideal candidate for providing persistent layout in an interactive setting.

We present a new interaction model, called *continuous network layout*, for integrating automatic graph layout into diagramming tools. In this model separation constraints are used to represent stylistic constraints (e.g. non-overlap and directed edges) and placement constraints (e.g. alignment and distribution). The graph layout engine runs continuously during user interaction, improving the rest of the layout in response to changes while maintaining constraints that have been placed on the diagram. The user can further improve the automatically generated layout by manipulating objects, for example, to prevent connector crossings.

The continuous network layout model is a natural combination of previous research into constraint-based placement tools in diagram editors and recent research into graph drawing. It provides an interactive form of layout where existing user-specified placement constraints and styles are maintained. In addition, this model has the added benefit of allowing the user to suggest new starting positions for the layout algorithm and visually recognise and free the algorithm from local minima. This allows a collaborative approach to graph layout involving algorithmic optimisation techniques but guided and assisted by the diagram author.

In Section 6.2 we provide a summary of related research into graph drawing. Section 6.3 presents a model for continuous semi-supervised network layout for interactive applications, as well as describing the IPSep-CoLa algorithm we use for constrained graph layout. Section 6.4 describes new user-specifiable placement tools implemented with IPSep-CoLa, Section 6.5 describes a few stylistic constraints, and Section 6.6 details Dunnart's example structural styles. Section 6.7 examines the issue of reporting constraint state as well as conflicts to the user. Section 6.8 discusses feedback from the use of this new model for several application domains.

The continuous network layout model uses a novel approach to diagram authoring which is significantly different from existing techniques. It is still in its infancy and has not yet experienced wide exposure—as such, there may be unrecognised usability problems. Nevertheless, this chapter discusses the key issues required to make it practical and shows evidence of its usefulness for several real-world diagramming applications. Formal usability

[1] Separation constraints have the form $u + d \leq v$ or $u + d = v$ where u and v are variables representing horizontal or vertical position and d is a constant giving the minimum separation required between u and v.

testing of the model would be extremely valuable. We plan to conduct some evaluation ourselves, and suggest that this should be an important avenue for future work.

6.2 Background

Researchers and practitioners in various fields have been arranging diagrams automatically using physical "mass-and-spring" models since at least 1965 (Fisk and Isett, 1965). Eades (1984) popularised the technique for graph drawing with the *spring embedder*. The spring embedder model treats nodes in the diagram as steel rings with edges represented as springs. It also adds additional repulsive forces between each pair of nodes. Over a number of iterations these forces pull the nodes to a stable configuration.

Kruskal and Seery (1980) and Kamada and Kawai (1989) introduced similar models formalising the force-directed approach. They add the concept of an ideal distance between non-neighbouring nodes proportional to the shortest path between them. The basic *stress* function that they attempt to minimise for each connected component is:

$$stress(X) = \sum_{i<j} w_{ij}(||X_i - X_j|| - d_{ij})^2 \tag{6.1}$$

where for each pair of objects i and j in the connected component, d_{ij} gives an ideal separation between i and j, $w_{ij} = \frac{1}{d_{ij}^{-2}}$ is used as a normalisation constant and X is a $n \times 2$ matrix of positions for all nodes, where 2 is the dimensionality of the drawing and n is the number of objects. This type of force-directed layout results in a graph where connected nodes are close together and weakly connected nodes are spread far apart from each other, see Figure 6.10(a).

Another graph layout technique, popular for drawing directed graphs in a layered fashion is that of Sugiyama, Tagawa and Toda (1981). The "Sugiyama" layout algorithm generates drawings of hierarchical (or directed acyclic) graphs using a four step process. The first step consists of some preprocessing to remove cycles from the input graph if necessary, turning it into a proper hierarchy. Also, edges that span more than one level in the hierarchy are given dummy vertices at each intermediate level so that every edge spans only a single level. In the second step, the nodes at each level are reordered to minimise edge crossings between neighbouring levels. The third step adjusts horizontal positions of nodes to minimise edge lengths. The fourth step is to draw the resultant graph. An example of Sugiyama layout can be seen in Figure 6.10(b).

The previous techniques were developed for once-off layout of unchanging graphs. *Dynamic graph layout* (Brandes and Wagner, 1997; Branke, 2001) is largely concerned with stable re-layout of changing graphs and interactive navigation of large graphs. Many of the models for interaction involve a user making structural changes to graphs with an existing static layout algorithm used repeatedly to provide a new layout. The research has often focused on modifications to these algorithms to have them provide more dynamic stability and thus better preservation of the user's *mental map* (Eades, Lai, Misue and Sugiyama, 1991; Misue, Eades, Lai and Sugiyama, 1995).

An example of this kind of dynamic graph layout research is Bridgeman, Fanto, Garg, Tamassia and Vismara's (1997) InteractiveGiotto, an interactive version of an orthogonal graph layout algorithm, Giotto, based on the network flow approach to bend minimisation (Tamassia, Battista and Batini, 1988). They modify Giotto to preserve crossings, the embedding of the graph (the circular order of edges around vertices), and number of connector corners. The layout still needs to be manually invoked by the user after they have made modifications to the graph.

Our approach of adding constraints to a graph layout algorithm to make it more stable and usable in an interactive situation is not new. Böhringer and Paulisch (1990) use the same approach, recognising the problems with not preserving user- or application-specified constraints as well as the issues with existing algorithms discarding information about the current layout when run. They talk about the importance of maintaining *dynamic stability*, which is the concept of minimising the difference between successive layouts, and *structural stability*, which is concerned with meeting user-specified layout constraints. They apply these concepts to Sugiyama's layered layout algorithm, adding constraints to maintain dynamic and structural stability.

GLIDE (Ryall et al., 1997), discussed earlier in Section 2.3, is a diagram editor with user-specifiable structural constraints (such as alignment, clustering or symmetry). These Visual Organisation Features (VOFs) are implemented using spring forces, and maintained during diagram editing. GLIDE adds node-node and edge-node forces by default to prevent overlap of diagram components. While GLIDE is based on a force-directed model, it doesn't do any general graph layout on the diagram, other than via explicitly created VOFs. Also, since constraints in GLIDE are implemented via springs, they may not always be fully satisfied, causing confusion for the user.

Our approach has similar aims to dynamic graph layout methods but ultimately shares more with research into collaborative graph layout tools, like the GDHints system. GDHints (do Nascimento and Eades, 2002) is an interactive system based on the Sugiyama layout algorithm. It is built on the idea of *interactive optimisation* (do Nascimento and Eades, 2005), where the user can interact with the optimisation engine to improve the layout and escape local minima by providing *user hints*. These user hints fall into three classes. The first class of hints are adjustment of objectives and constraints. The user has the ability to alter the objective function as well as graphically adding or removing constraints to encode domain information and preferences. The second type of hint is directing the focus of an optimisation method. The user can select a portion of the graph and run the optimisation method on just that. The third hint type are manual changes. Basically, the user can make other changes to the initial graph, such as directly placing a node, to positively influence the outcome of the optimisation.

Not all previous approaches maintain user preferences such as the positions of objects, or satisfy user-specified constraints. Those that do offer these features—necessary for interactive applications—may be unable to handle certain styles (like downward pointing connectors) or they place constraints of their own on the diagram due to they underlying layout technique. For example, methods based on the Sugiyama framework impose rigid

and often arbitrary constraints on the layout: all nodes must lie on regularly spaced layers and all edges must span a layer (whether they are directed or not). Our proposed method is highly flexible while allowing specific direction from the user. It can model the style of many previous graph drawing algorithms without imposing any rigid layout constraints of its own.

6.3 Continuous network layout

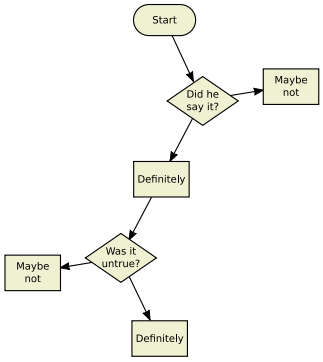
The *continuous network layout* model allows creation of better layouts via collaboration. In this model a graph layout engine performs general optimisation[2] while the user is able to visually recognise and correct undesirable properties in the layout through direct manipulation and addition of placement relationships. For example, the user may drag a shape to a different location to prevent a couple of edge crossings, and in doing so help the optimisation algorithm escape from a local minimum and find a better solution.

The continuous network layout approach results is a vastly different interaction model than we have been used to in previous chapters. Here, the editor is free to rearrange the entire layout of the diagram in response to any change made by the user. The model offers considerable power in that it supports user-specified placement constraints such as alignment and distribution. It also allows the user to set structural styles or stylistic constraints that automatically apply certain constraints to the entire diagram to produce layouts of a certain style. It supports inequality constraints as well as equalities, allowing non-overlap and minimum separation relationships.

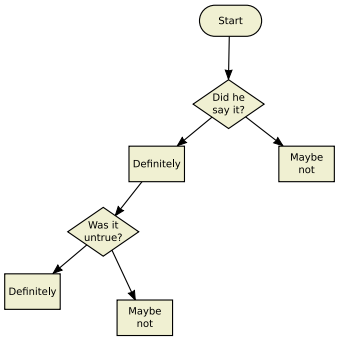
Finding a good layout for a network is a computationally difficult task. We recognise that algorithms are very good at certain kinds of layout improvement. We also recognise that they often fall short of expectations since users have their own aesthetic preferences and quickly detect what they see as visual flaws in layout. The continuous network model allows interactive collaboration between user and layout software to produce the best layout. The user is able to specify structural, placement and style constraints. Within the limits of these constraints, the layout algorithm is able to optimise the readability of the diagram, enhancing the structural readability by placing strongly connected objects close to each other, separating weakly connected objects, making connectors as straight and short as possible. The user can interact with the layout to provide hints and guide the optimisation. In the model, the layout can enforce strict structural layout conventions useful for certain types of network diagrams. Where an existing layout is given, it can be beautified without altering its topology by adding *topology preserving* constraints. These constraints prevent shapes from crossing connectors, so all connector crossings will be preserved with no additional crossings created. This model is designed to provide much more usable and useful automatic network layout than existing diagram editors.

Figures 6.1–6.3 illustrate the construction of an example network using the continuous network layout model. The example demonstrates structural style—flow layout—being

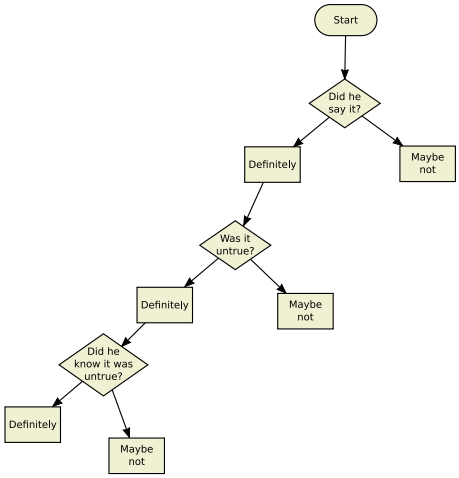
[2]Here, "general optimisation" refers to improvement to the diagram in terms of force-directed graph layout; moving nodes so that connectors have an ideal length, ensuring objects are not overlapping and bringing strongly connected objects close together.



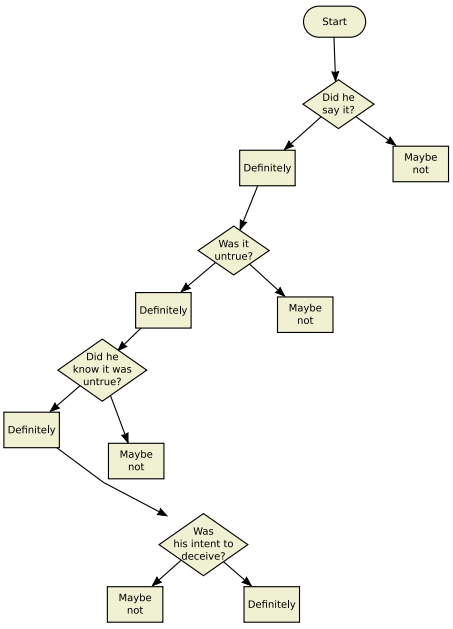
(a) The layout engine runs continuously during diagram construction to satisfy constraints and optimise the aesthetic criteria.



(b) Here, downward flow layout has been selected by the user as the structural style for the diagram. Shapes will be arranged so that connectors point downwards.

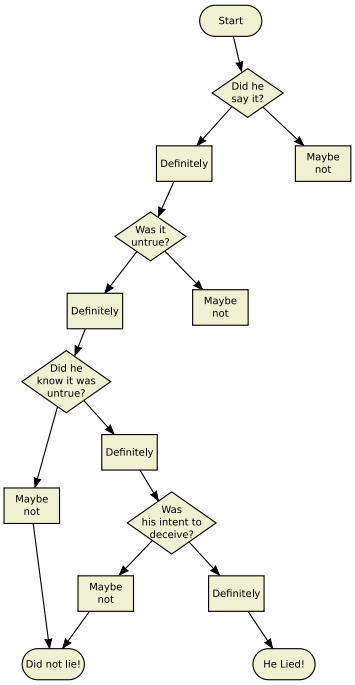


(c) As the diagram grows, the layout engine maintains its structural style as well as enforcing stylistic constraints such as non-overlap between shapes.

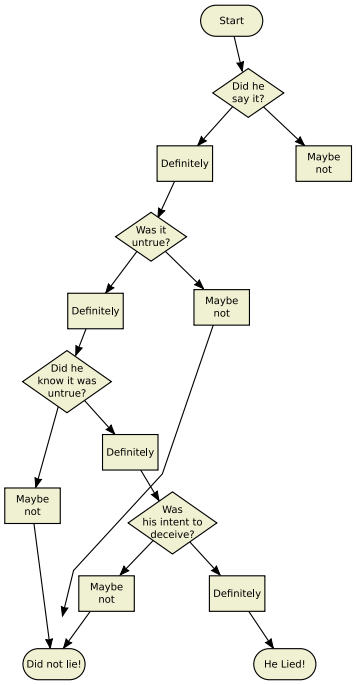


(d) While connectors are created, they are continuously routed to avoid crossings with shapes.

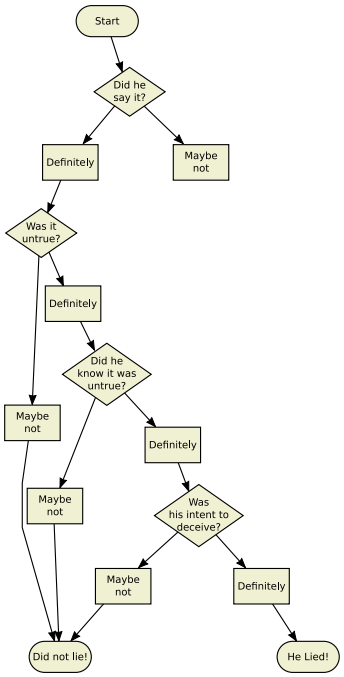
Figure 6.1: Constructing a network diagram in Dunnart using continuous network layout. "Did he lie?" flowchart originally from `http://sean.gleeson.us/2005/11/07/`



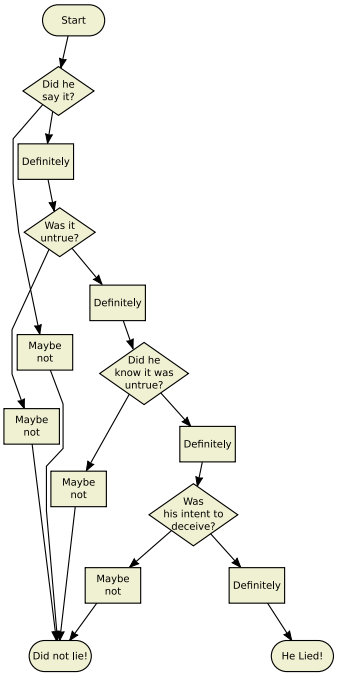
(a) The remaining shapes are added to the diagram.



(b) A connection is made between two shapes. The automatic routing during creation of connectors is shown here.

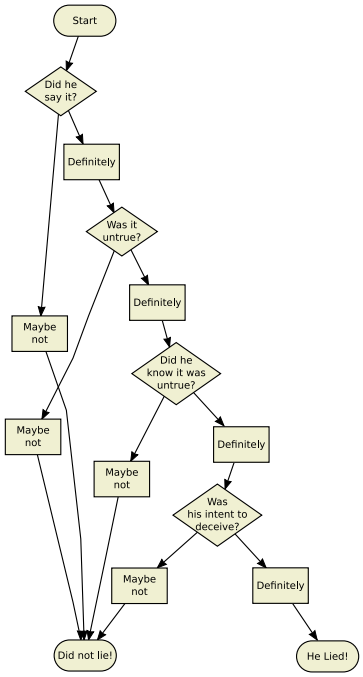


(c) When the connector is finalised, the network layout rearranges the shapes to clean up the diagram.

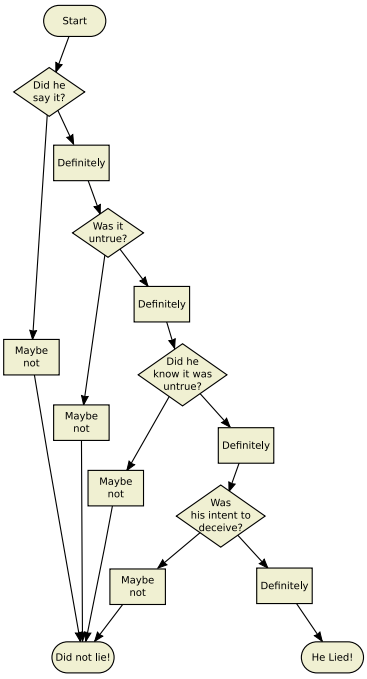


(d) After the last connector is added to the diagram, we can see the automatic network layout results in an undesired (and unnecessary) connector crossing.

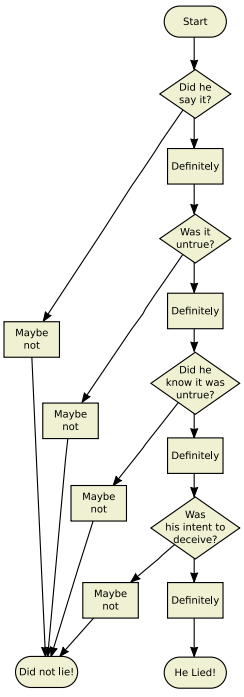
Figure 6.2: Constructing a network diagram in Dunnart using continuous network layout. (Continued.)



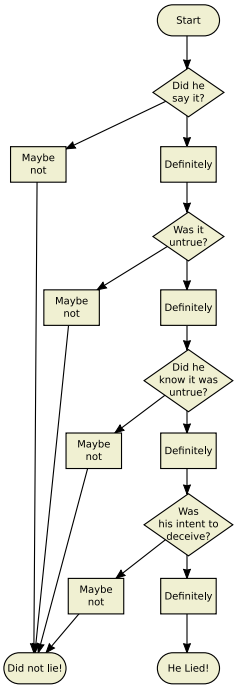
(a) Multi-segment connectors are shortened and straightened after enabling connector straightening constraints. These constraints also serve to maintain the topology of the diagram throughout further editing.



(b) The user interrupts the layout by holding down the ALT key. They then alter the topology to get rid of the connector crossing by picking up one of the "maybe not" nodes and placing it at the left of the diagram.



(c) The user draws attention to a critical path through the diagram by selecting the nodes involved and creating a vertical alignment relationship.



(d) The Diagram is further neatened by adding horizontal alignment constraints for each of the "definitely/maybe not" decision choices.

Figure 6.3: Constructing a network diagram in Dunnart using continuous network layout. (Continued.)

enforced throughout the construction. It shows topology preserving features of the continuous network layout model (Figure 6.3(a)), as well as the user using the ALT key modifier to manually alter the topology of the diagram to reduce connector crossings (Figure 6.3(b)). Finally, it shows how user-specified placement constraints such as an alignment relationship are integrated with the technique and can be used by the author to draw attention to features of the diagram (Figure 6.3(c)) or just to enforce personal aesthetic preferences (Figure 6.3(d)).

To implement this model we require a flexible, incremental graph layout algorithm that handles constraints. Equality constraints are needed for our existing placement tools. Inequality constraints are also necessary to encode containment and non-overlap behaviour.

6.3.1 IPSep-CoLa: Constrained stress majorization

Our work on interactive graph layout is built upon Dwyer, Koren and Marriott's (2006) IPSep-CoLa algorithm. IPSep-CoLa provides constrained force-directed layout. Like other force-directed layout approaches, it works by finding an embedding of the graph in 2D space that minimises some continuous goal function. The goal function (or *stress function*) used by IPSep-CoLa is the same as that of Kamada and Kawai (1989), given earlier as Equation 6.1.

In standard stress majorization the value of the stress function is reduced by alternately minimising quadratic forms in the horizontal and vertical dimensions that bound the stress function, see Gansner, Koren and North (2004)). Constrained stress majorization does the same except that the bounding stress function is solved at each iteration subject to a set of separation constraints. To do this, IPSep-CoLa uses the gradient projection algorithm detailed in Dwyer, Koren and Marriott (2006). The advantage of gradient projection is that it is quite fast and inherently incremental, especially when the active separation constraints from the previous problem form a good approximation to those for the current problem.

IPSep-CoLa is a good choice for implementing interactive network layout for a couple of reasons. It gives the benefits of force-directed layout methods: placing strongly connected objects close together and separating weakly connected objects. It offers separation constraints which can be used to specify drawing styles and placement relationships. Finally, as well as being incremental, it gains the benefits of stress majorization, namely efficiency and better convergence than Kamada and Kawai (1989).

While constrained stress majorization offers a powerful and flexible approach to network layout, it is not without its limitations. Because it solves the optimisation problem in two dimensions, we are limited to creating constraints that operate in the x and y dimensions. We can't easily handle creation of an alignment guideline that has any other orientation other than horizontal or vertical. Likewise, we cannot handle non-overlap of convex shapes, we are forced to approximate this by applying non-overlap constraints to the bounding boxes of shapes.

An implementation of the IPSep-CoLa algorithm is available in the open-source C++ software library `libcola` as part of the AdaptaGrams project.[3] This library was integrated with Dunnart to provide the new placement tools and interactive network layout.

6.3.2 Architecture

The layout engine, a `libcola` instance, runs in a separate thread to the rest of Dunnart. This way Dunnart itself can remain responsive at all times, even while the layout engine is working. The main thread handles all interaction with the user, as well as "repainting" the canvas at appropriate times, as shown in Figure 6.4. In theory the layout thread can be running continuously, but this tends to only happen while the user is continuously dragging. When there is no user interaction the layout thread will usually converge quickly for small graphs, settling on a stable solution after a couple of seconds.[4] The layout thread will then be put to sleep.

When the user moves a shape or multiple shapes, Dunnart fulfils that action and then stores a list of the moved shapes and their new positions. It then checks if the layout thread is running. If it is, it sets a flag to let the layout thread know there are some new position changes from Dunnart. If the layout thread has finished (converged at a stable layout) then it sends a signal to the layout thread to wake it up.

After each iteration of the graph layout process IPSep-CoLa will call a pre-registered callback function, allowing us a chance to update canvas objects with new positions from the layout. At this stage we first check if the layout has been interrupted by Dunnart—if the user has done something in the main thread to cancel the layout or invalidate it. If not, we generate a list containing a new position for each diagram object as reported by the layout thread. Next we check if Dunnart has finished handling the last of these updates from the layout thread. If it hasn't, we wait. If it has, we generate a new event (much like a mouse movement or key press event) that states there are updates from the layout thread waiting to be processed. Mutual Exclusion (Mutex) techniques are used throughout the code to protect shared data structures (like the diagram object position list) that might be accessed by both Dunnart and the layout thread.

The main Dunnart thread sees these update events as part of its standard event processing, that is, layout events will be interleaved with mouse movement events while the user drags shapes. Dunnart handles these layout events by moving each of the diagram objects and then redrawing the canvas. This update need not trigger any further work other than connector routing since all other constraints in the diagram are captured within the IPSep-CoLa model.

Undo and Redo operations in Dunnart are complicated slightly by the continuous layout model. An important difference when using this model is that user actions are not always atomic. That is, the complete results of an action may not occur until a couple of seconds after the original action. Also, they may be interrupted by the user and not occur at all. When the user undoes an action in Dunnart, they are returned to the layout

[3] AdaptaGrams, Tim Dwyer, `http://adaptagrams.sourceforge.net/`

[4] `libcola` actually takes much less time to compute a stable solution, but we slow it down by animating all the intermediate steps graphically to help preserve the user's mental map of the diagram.

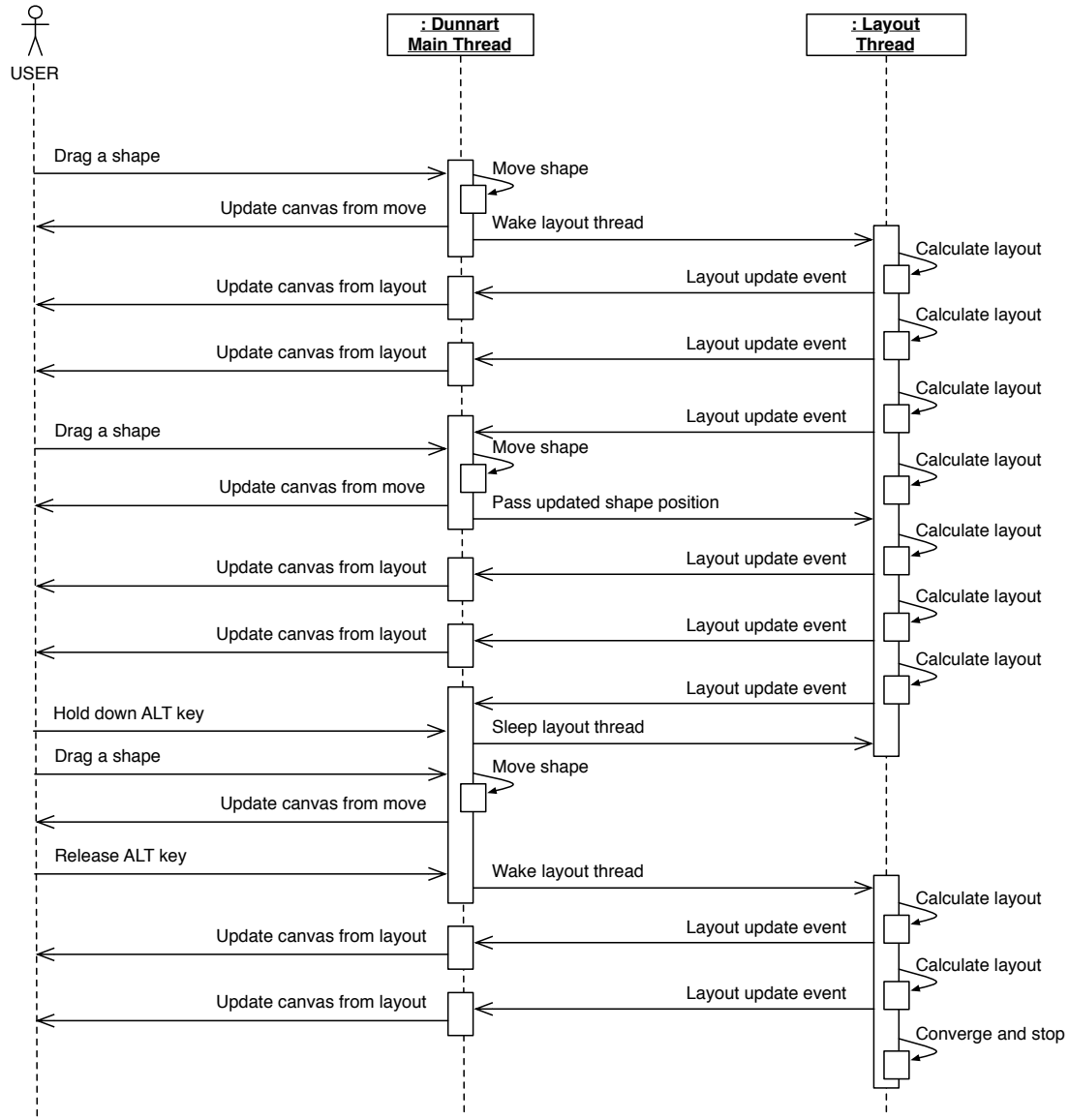


Figure 6.4: Sequence diagram showing interaction between Dunnart's main thread and the graph layout thread. Graph layout runs in the background, continuously passing back layout updates while Dunnart remains responsive for user interaction. The user can pause the automatic network layout by holding down the ALT key.

of the diagram at the point they began the original action. To do this, we allow a single undo scope to collect movement events right up until the next undo scope is begun by the next user action. Even if the layout had originally been running, the layout thread is always stopped following an undo action. The motivation for this decision is that the user is using undo so that they can return to an earlier point and make a different action. If the layout changes quickly (by itself) after the undo action, it will be hard for the user to evaluate the state of the diagram and make their choice of action before the diagram changes again.

6.3.3 User interaction

Dragging shapes is handled by invisibly locking those shapes, effectively specifying a constraint to keep them at exactly their current position. As the user drags a shape it is moved relative to the mouse cursor and then locked at that position for the purposes of layout until the dragging specifies a new position.

During dragging of shapes the layout engine will run continuously as new positions for the selected shapes are passed to it. Dunnart can handle over 20 layout updates a second, resulting in video-like animation for small graphs. Coupled with the incremental nature of the algorithm, this animation enables us to smoothly show layout changes, thus preserving the user's mental map of the diagram. When the user stops dragging the layout engine tends to converge to a locally optimal solution after a couple of seconds and then goes to sleep.

A nice side-effect of our optimisation method is that differences between subsequent layout iterations reduce as the layout converges—that is, the stress is reduced quickly at first and then more slowly. This resultant slow-down of objects before stopping is beneficial for a user who is tracking them by eye. Moving objects in the real world usually slow down before coming to a stop. Fundamental animation principles—which have also been applied to animation in user interfaces—advocate this slow-down during movement, referred to as *slow-in* and *slow-out* (Lasseter, 1987; Chang and Ungar, 1993). This technique aims to improve the realism of moving objects by making their behaviour similar to that of physical objects. While our layout model exhibits the slow-out behaviour after movement it does not have slow-in behaviour—shapes may move a large amount on the first layout iteration. We could simulate slow-in behaviour through animation, by drawing the shapes at intermediate positions during their movement. Future usability experiments could test the value of such animation techniques for continuous network layout.

The behaviour of non-selected objects in the diagram moving automatically could be considered problematic in situations where the user wants to click on a shape that is being automatically moved by the layout engine. In practice we have not observed this to be a problem. Shapes quickly slow down after a dragging operation is completed, and users are able to select them easily if they are only moving slightly.

A more problematic example is that of the user who wants to attach a shape to a guideline, but the layout engine moves the guideline away as the shape is dragged towards it. Another issue occurs when the user wants to draw a connector to a shape that is being

moved away. To solve these problems we allow the user to hold down the ALT key at any time to "pause" the layout engine. As shown in Figure 6.4, if ALT is pressed, the layout engine discards the updated layout it had generated and goes to sleep. For the user, this causes all automatic network layout movement to cease. The objects being dragged are updated still, as are connectors attached to these objects. When ALT is released the layout thread is sent a signal waking it up and automatic network layout and constraint satisfaction continues in the background.

By turning off Automatic Graph Layout, Dunnart can be used for diagramming without any automatic network layout. The layout engine is then used only to satisfy placement relationships like alignment and distribution, and maintain stylistic constraints such as non-overlap.

6.3.4 Edge straightening for topology preservation

Bertault (1999) showed that in force-directed models repulsive forces could be applied between nodes and their projection point on edges to prevent nodes from passing through edges, thus preserving the topology of the graph during layout. However, Bertault's technique was presented only for graphs with straight-line edges and point-size nodes.

In Dunnart we have an option called "straighten connectors". This uses a process described in (Dwyer, Marriott and Wybrow, 2007) which improves the layout by moving nodes and edge bend points so that edges are straightened and made more uniform in length. To prevent creation of new overlaps or connector crossings it automatically creates separation constraints between shapes, connectors and other connectors. These constraints preserve the topology of the diagram during further layout.

The edge straightening technique requires an initial layout including node positions and poly-line routes for edges. Since the edge straightening preserves the crossing properties of the layout, it is important that the initial layout has no overlap between shapes and connectors and that shared bend points between connectors have been nudged apart to minimise crossings. In Dunnart we can provide this layout via the following steps. First, we position the shapes in the diagram with a force-directed layout using IPSep-CoLa. We apply non-overlap constraints, but ignore edge routing. Next, we route poly-line connectors for the resultant diagram using `libavoid` and making use of the extensions for reducing crossings and nudging common bend points apart, described earlier in Sections 5.5.1 and 5.5.2.

The actual straightening is achieved by attempting to move connector bend points to their projection point on the line made by their accompanying segments. Consider a bend point $b = (x_b, y_b)$ on a line from $p_1 = (x_1, y_1)$ to $p_2 = (x_2, y_2)$. The minimal change to the position of b which straightens the line is to move b to its projection p_b on to the line $\overline{p_1 p_2}$. We let t_b be distance of p_b along $p_1\vec{p}_2$, that is, $p_b = p_1 + t_b(p_2 - p_1)$ where $t_b = \frac{\vec{p_1 b} \cdot p_1\vec{p}_2}{||p_1\vec{p}_2||^2}$ (from the dot product rule for scalar projection) Since all paths have minimal length and bends we have that the projection point must lie between p_1 and p_2, i.e. that $0 \leq t_b \leq 1$.

So for an edge $(v_i, v_j) \in E$ with one bend at position b we can straighten the edge when moving vertices and bend points horizontally by minimising $f(x_i, x_j, x_b) = (x_b -$

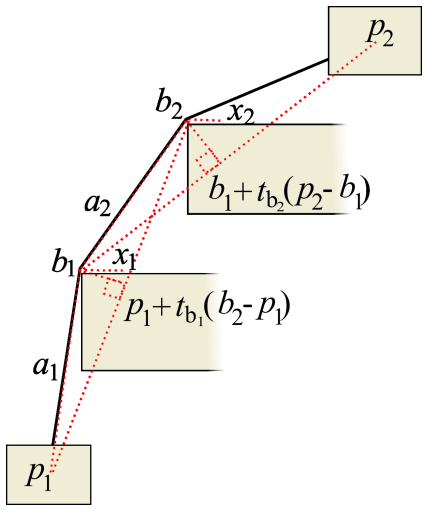


Figure 6.5: The bend b_1 will be straightened to its projection point on the line $\overline{p_1 b_2}$, e.g. in the x-dimension, to x_1. Bend b_2 will be similarly straightened towards the line $\overline{b_1 p_2}$. The potential bend points a_1 and a_2 will be straightened to the line segment between actual bend points or the ends of the line, i.e. a_1 will be straightened to the line segment $\overline{p_1 b_1}$.

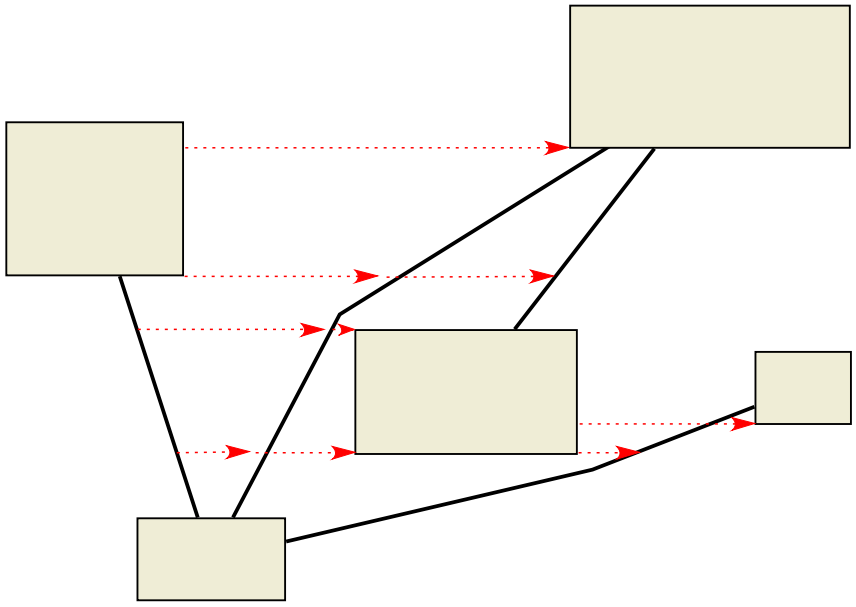


Figure 6.6: Example showing the separation constraints (dashed arrows) required to prevent the creation of new node/node and node/edge crossings when moving nodes horizontally.

$(1-t_b)x_i - t_b x_j)^2$. If an edge is routed through multiple bend points $b[1], b[2], ..., b[n]$ we can straighten all bends by minimising:

$$f(x_i, x_b[1], x_b[2]) + \sum_{k=2}^{n-1} f(x_{b[k-1]}, x_{b[k]}, x_{b[k+1]}) + f(x_{b[n-1]}, x_{b[n]}, x_j)$$

and similarly, when placing points vertically, we will straighten edges by minimising an equivalent set of expressions over y. This is illustrated in Figure 6.5.

To prevent the bend points moving across connectors or shapes and creating crossings, separation constraints are added between bend points, shapes and potential bend points on other connectors. Figure 6.6 shows the constraints generated when moving nodes horizontally. Notice that additional *potential* bend points are introduced in some straight edge segments. These are created horizontally or vertically in line with shape corners or existing bend points so there is a point that can be subject to the separation constraints. To prevent these potential bend points becoming active bend points, they are strongly weighted to lie on the straight line between corresponding active bend points.

Edge straightening is useful for a couple of reasons. The straightening itself is useful in beautifying the diagram: it reduces the angle between connector segments and removes unnecessary bend points (and thus segments) from connectors. The guarantee of connector crossings not being introduced is important; the straightening will improve the layout rather than degrade it. Also, the topology-preserving aspect is useful in an interactive context. The user may place a shape in a particular location of the diagram and want to beautify the diagram while preventing the shape from drifting back to its ideal force-directed position. Importantly, since the straightening is implemented as additional stresses and constraints on top of the IPSep-CoLa algorithm, we can still preserve existing user-specified placement constraints such as alignment and distribution during the straightening process.

6.4 User-specified placement constraints

The integration of `libcola` into Dunnart involved replacing the QOCA constraint solver. QOCA was being used to solve constraints representing placement relationships such as alignment and distribution. As discussed in the previous section, `libcola` can handle these constraints as part of its force-directed layout. In fact, by having `libcola` ignore all connectors in the diagram it will not do any network layout optimisation. Instead, it will compute the closest feasible solution from a given set of separation constraints. In this way `libcola` can be used purely to provide placement tools without any kind of automatic network layout.

Dunnart's alignment and distribution tools were rewritten to use equality constraints in IPSep-CoLa. Functionally, the new versions act the same as those implemented in QOCA, as described in Chapter 4. Their behaviour and appearance has not changed—the constraints are just being handled by `libcola` instead of QOCA.

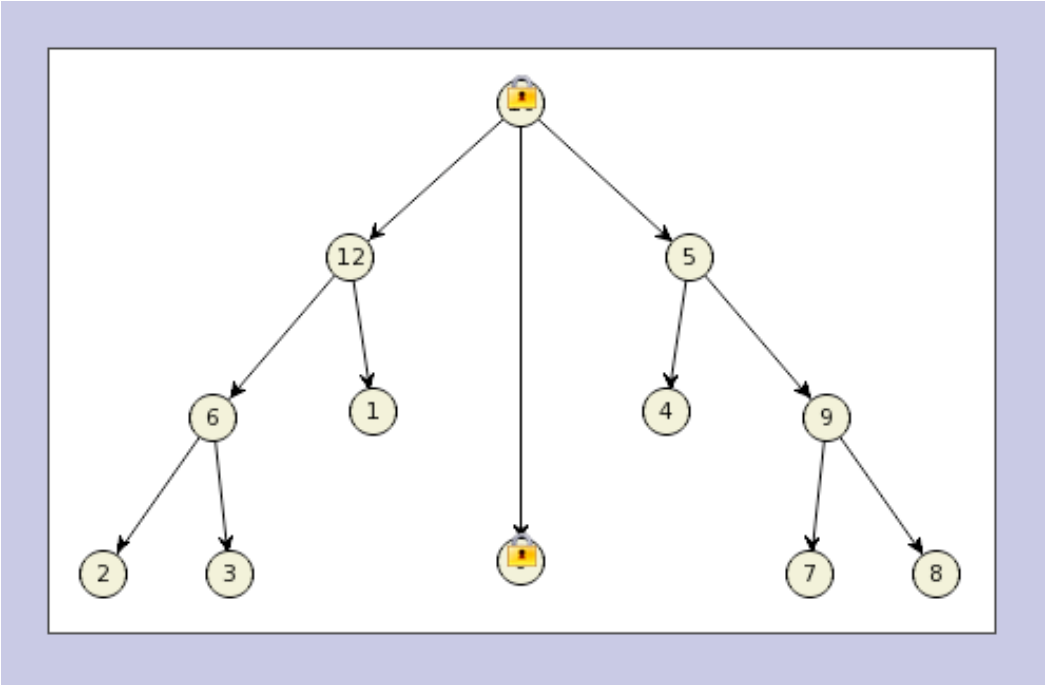


Figure 6.7: A diagram with locked shapes (marked with padlock icons). These shapes are held in their current position and will not be moved by the network layout or placement tools. The user is still able to drag them to position them manually.

As the constraint solving is now done by a background thread, objects moved by the solver can lag slightly behind the objects being dragged. This was an implementation choice; we could wait to move the dragged objects until the layout has computed new positions for the other objects, but then the dragged objects may appear to lag behind the mouse position. In any case, the lag is very minor and not strongly apparent.

Diagram authors sometimes want to keep objects on the page separated by a minimum distance, or they may just want to preserve the horizontal or vertical ordering for some objects. Separation constraints can be used to provide exactly this behaviour. Therefore, we created a new user-specified placement tool for minimum separations. We also added an anchoring tool that can be used by the user to lock the position of a shape, as requested by participants in the first study. This can be useful for preventing specific shapes from being moved by automatic network layout.

Both these tools have an on-screen representation that is compatible with the indicator visualisations such as fading and Information Mode, as discussed in Chapter 4. We describe these new tools here.

6.4.1 Anchoring

We have added user-specifiable anchoring constraints that allow the position of shapes to be locked. The user selects one or more shapes and then clicks the “Lock/Unlock Shape Positions” toolbar button to mark those shapes as locked. While locked, these shapes will be constrained to their current position. They will not be moved by any of the placement

tools or automatic network layout, but can still be moved manually by the user. If a locked shape is dragged it becomes unlocked during the movement operation and is then locked at its new position.

Locked shapes are displayed with a padlock icon in their upper right-hand corner, as shown in Figure 6.7. Locked shapes can be unlocked by selecting them and clicking the "Lock/Unlock Shape Positions" toolbar button.

Locked shapes are frozen in both the x and y dimensions. Since this is done with two separate constraints, we could easily allow shapes to be locked in just one dimension. However, this would require a more descriptive on-screen representation, and the behaviour can already be accomplished—with a clear on-screen representation—by attaching the shape to a locked guideline. The shape will be able to slide along the guideline but will remain locked in the other dimension.

6.4.2 Separations

Separation relationships define a minimum separation between objects. They are useful for spacing objects out or keeping them in relative order on the page. Separations are similar to distributions: they can be applied to a selection of two or more objects, they operate on guidelines, and applying them to a set of shapes creates guidelines and attaches the shapes to them. Separation relationships are created by selecting a set of shapes or guidelines and clicking the "Keep Shapes Separated" button on the toolbar. This opens a dialog box where the user can select horizontal or vertical separation, as well as the initial separation distance.

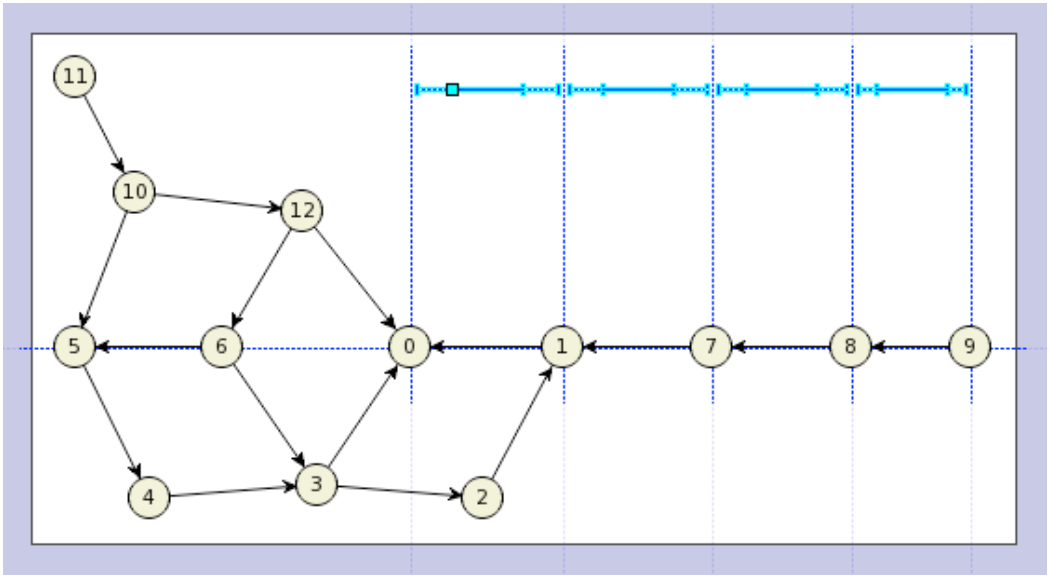
Internally, a variable s is used to represent the minimum separation between two adjacent guidelines ($g1$ and $g2$) in the distribution. An inequality constraint is created between each pair of adjacent guidelines to constrain their distance apart to be equal to or greater than the minimum separation value:

$$(g1 - g2) >= s$$

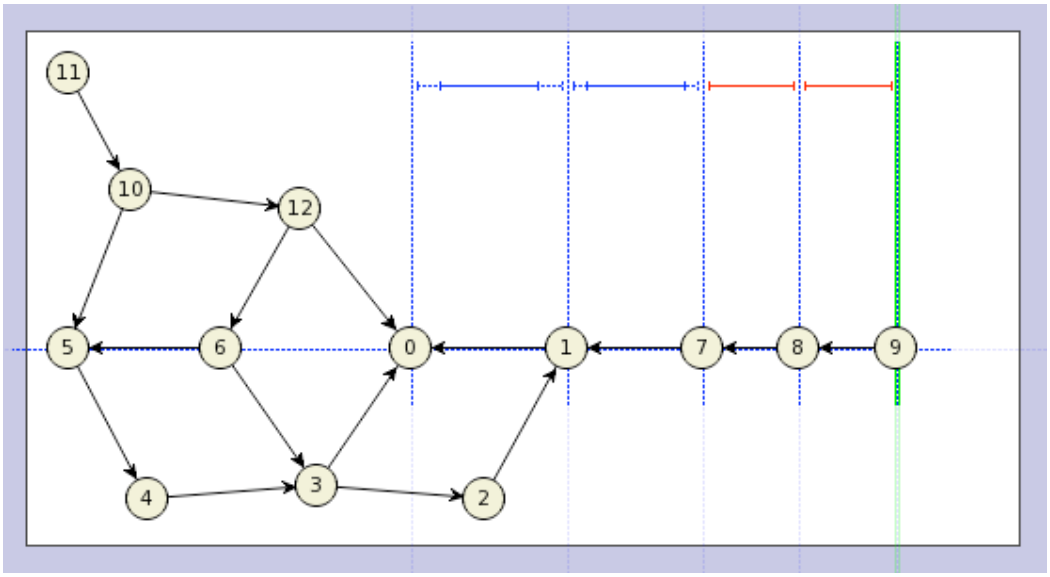
Once created, a separation relationship has an indicator object as its visual representation. When a separation indicator is selected, it has a handle that can be dragged to resize the minimum separation distance between objects in the separation relationship, as shown in Figure 6.8(a). The separation indicator shows the relationship between separated objects as well as the minimum separation distance. When two objects in the separation relationship are pushed together to their minimum separation distance and the constraint becomes active (at equality), this length of the separation indicator is highlighted, as seen in Figure 6.8(b). Deleting a separation indicator removes the separation relationship.

6.5 Stylistic constraints

Stylistic constraints are a form of layout constraints that can be easily applied to the entire diagram at once. When the user enables a particular layout style, the layout engine



(a) A resize handle is shown while the separation indicator is selected



(b) The separation indicator highlights objects at their minimum separation

Figure 6.8: A diagram with a separation relationship. (a) While the separation indicator is selected a resize handle can be dragged to modify the minimum separation distance. (b) As the rightmost guideline is dragged to the left, the separation indicator highlights where objects are pushed together to their minimum separation distance.

automatically generates all the necessary constraints to fulfil that style for the entire diagram. Examples of stylistic constraints are non-overlap, page containment and downward pointing connectors.

While all stylistic constraints could be applied to objects individually, like placement tools, they would then require their own interface and on-screen representation to notify the user of their presence. We felt that these constraints would be more useful if presented to the user as a style that could be toggled on or off for all objects at once.

6.5.1 Non-overlap

The user can enable non-overlap constraints for the entire diagram via a checkbox in the layout properties dialog. An accompanying slider controls the minimum distance between shapes. By default this distance is set to 10 pixels, which always allows room for connectors to be routed between shapes.

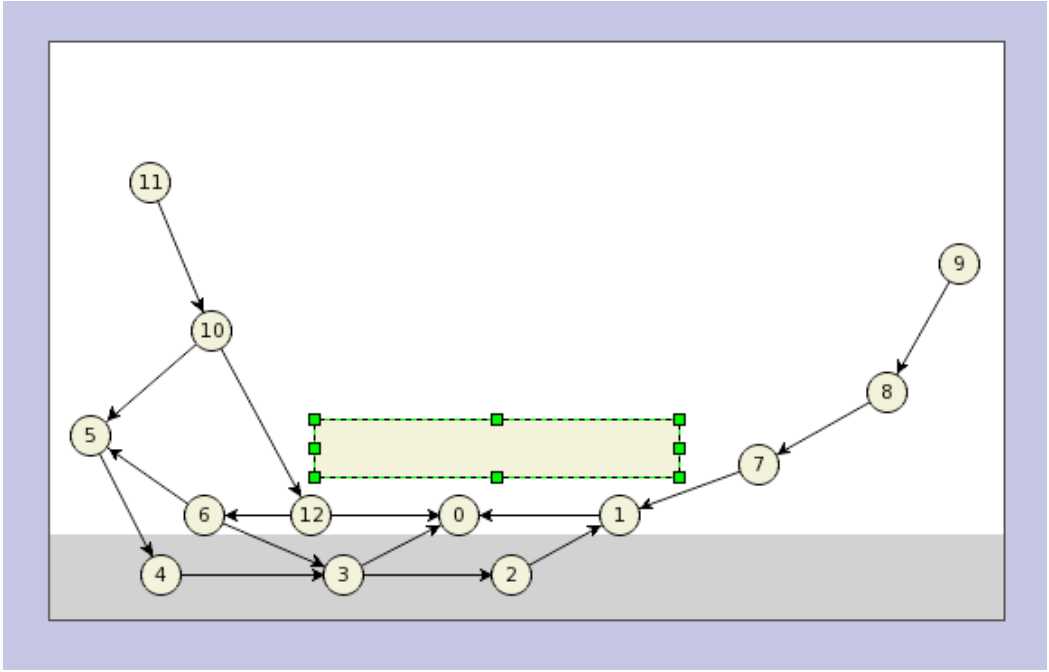
Since IPSep-CoLa only solves a single dimension at once, we need not generate non-overlap constraints between every pair of shapes. Thus, non-overlap constraints are generated at each iteration for the current dimension using the fast scan-line algorithm given by Dwyer, Marriott and Stuckey (2006). In the case of the x-dimension this works by moving a scan-line vertically down the diagram and keeping track of the objects and container boundaries that are "open" at that vertical position. In essence when an object boundary is first encountered a left-of constraint is placed between it and its closest neighbour to the right and a right-of constraint to the neighbour to the left, ensuring non-overlap in the horizontal direction.

As with the other stylistic constraints, there is no visible on-screen representation for non-overlap constraints. Their presence is immediately obvious while interacting with the diagram—dragging a shape around causes other shapes to be pushed around as non-overlap constraints prevent them from overlapping. We have found this style of interaction to be quite natural, and have not witnessed any problems from users who were confused by the constraints or a lack of visible indicators. This is most likely because the interaction produced by the constraints conforms to the familiarity and predictability principles; the objects on-screen behave like objects in the real-world and are therefore easily understood.

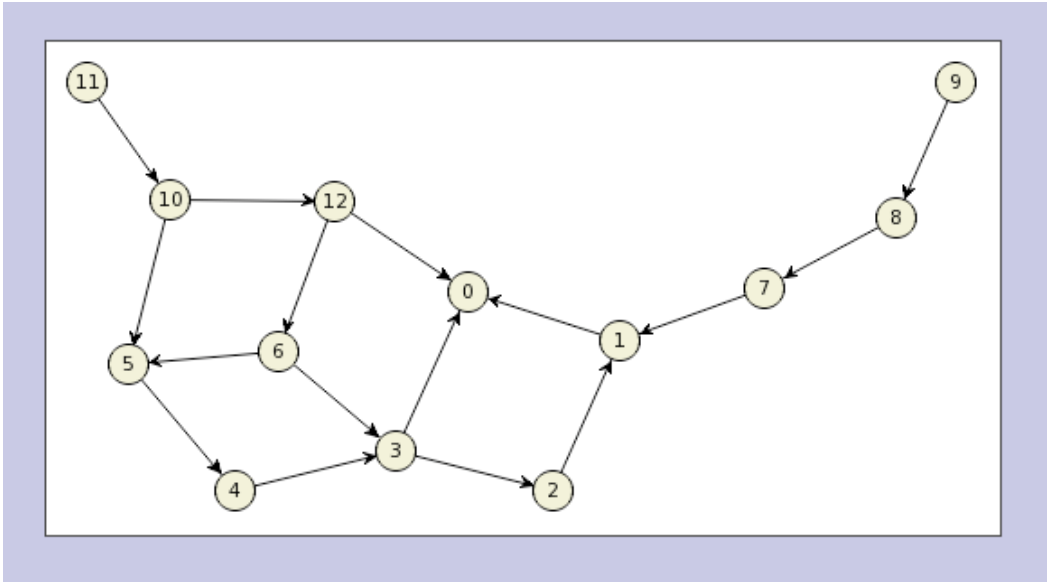
6.5.2 Page containment

Page containment constraints keep the entire diagram within the bounds of the page where possible. This behaviour is useful for a user who wants to arrange the diagram into a page of a certain size for printing. In Dunnart we have made page containment constraints a style that can be toggled on or off for the entire diagram via a checkbox in the layout options dialog.

Page containment constraints are implemented using two dummy variables in each dimension. That is, the position of the page top and bottom are used in the y dimension. Each shape in the diagram is then constrained to lie between the positions of these two



(a) Diagram forced outside of the page boundary



(b) Diagram arranged to fit within the page boundary

Figure 6.9: A diagram with page boundary constraints, causing the diagram to be kept within the page boundary. If non-overlap constraints are enabled and a shape is pushed into other objects while dragging, the diagram will not be able to fit within the page and the page will "balloon out" to accommodate those objects. When the dragged shape is removed, the diagram will move back within the page.

variable using separation constraints.[5] These dummy variables are given ideal positions reflecting the real page size and dimensions, but they will be moved if other constraints in the diagram require it. In this way, the page boundary has an elastic property where it can balloon out when necessary, for example, if a user should constrain the diagram to the page bounds and then drag a shape outside the page area.

Previous approaches have variously added extra forces to all nodes to keep them close to the centre of the page (Frick, Ludwig and Mehldau, 1995), added extra forces to keep nodes from getting too close to the boundaries of the page (Davidson and Harel, 1996), or just moved nodes back to the closest point on the boundary of the page after the layout is complete (Fruchterman and Reingold, 1991). In contrast to the force-based approaches, our method does not affect the layout if the diagram would fit within the page area.

In Dunnart, page containment constraints do not really have an on-screen representation except when effectively unsatisfied; when the dummy page edge variables are not at their ideal positions. In this case the page "balloons out" as shown in Figure 6.9(a). When space becomes available the shapes will move back onto the page, as in Figure 6.9(b). This makes it very clear to the user that this stylistic constraint is present. When dragging a shape, that shape is locked to the position of the mouse cursor. The user is always able to drag a shape outside the page and view this ballooning effect. We have found this to be a clear indicator of the page containment constraints. When all objects are within the page there is no visual indicator, but this is fine because the constraints are slack or not in use. Effectively we are just visualising the constraints when they are tight or cannot be fully satisfied.

Sticky nodes

While non-overlap constraints are enabled, shapes being dragged by the user will push other shapes out of the way. The dragged shapes act like a bulldozer, permanently moving objects they run into. Because this behaviour may be undesirable in many situations, Dunnart has an option called "Sticky nodes", which adds a small force attracting nodes to their starting position from the beginning of the movement operation. The amount of global stickiness can also be adjusted. The effect of this option is that nodes are still pushed around by a shape being dragged while non-overlap is turned on, but when that shape is no longer pushing them they will drift slowly (or quickly with a stronger stickiness) back to their original position from the beginning of the drag operation.

Setting a high value for global stickiness will improve dynamic stability when making structural changes to a graph since nodes will not move far from their starting positions. This achieves a similar result to Diehl and Görg (2002) who use a layout adjustment technique and simulated annealing to prevent nodes moving much during successive iterations for simple spring embedders.

[5] If non-overlap constraints are enabled, then only an object with no neighbour between it and the boundary is given this constraint, since non-overlap constraints will keep other objects inside if their neighbour is constrained to be inside.

6.5.3 Downward pointing connectors

When downward pointing connectors are enabled, separation constraints are added between each pair of shapes (*src*, *dst*) connected by a directed connector so that all connectors flow downward:

$$dst.y >= src.y + sep$$

Here, *sep* is a global separation distance, computed based on a percentage of the ideal edge length for standard force-directed layout. This percentage modifier defaults to 0.5 and can be adjusted interactively by the user via a slider in the Layout Properties dialog. See Figures 6.1–6.3 for an example of downward pointing edge constraints in action.

Sugiyama and Misue (1995) previously showed how directed edges could be treated like needles in a magnetic field to achieve a similar effect in the force-directed layout model. This is not so flexible as using constraints and tends to result in the graph becoming more bunched up because all directed edges are attempting to point straight down.

Dunnart currently allows only downward-pointing directed edge constraints, but it could trivially be extended to allow directed edge constraints in any of the four primary compass directions.

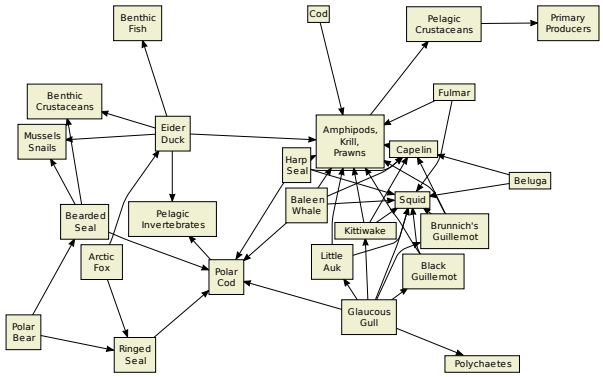
6.6 Structural layout styles

Structural layout styles prescribe a set of automatically generated constraints based on graph structure. They can provide a procedure to calculate an initial layout, which may use either constrained force-directed layout from IPSep-CoLa or an external generic graph layout algorithm. They must specify a set of constraints to maintain the desired features of the generated layout.

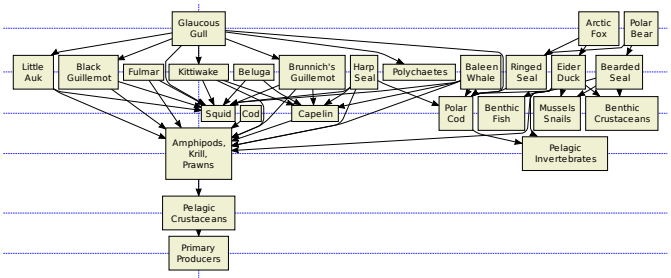
In this way, existing generic graph layout algorithms can easily be used to produce specialist drawing styles that can then be maintained and modified by the user within the continuous network layout model. Presently, Dunnart employs three structural styles, as shown in Figure 6.10.

- **Organic-style layout.** Organic layout is quite simple, effectively being force-directed layout with no additional style constraints other than non-overlap. When the user activates Organic-style layout three steps occur. First, constrained stress majorization is performed using the current position of shapes as a starting point. It takes into account user-specified placement constraints, but non-overlap and containment constraints are ignored so that the graph can more freely untangle and is less likely to get caught in undesirable local minima. Second, constrained stress majorization with non-overlap and containment constraints is performed. Finally, object-avoiding poly-line connector routing is performed using `libavoid`.

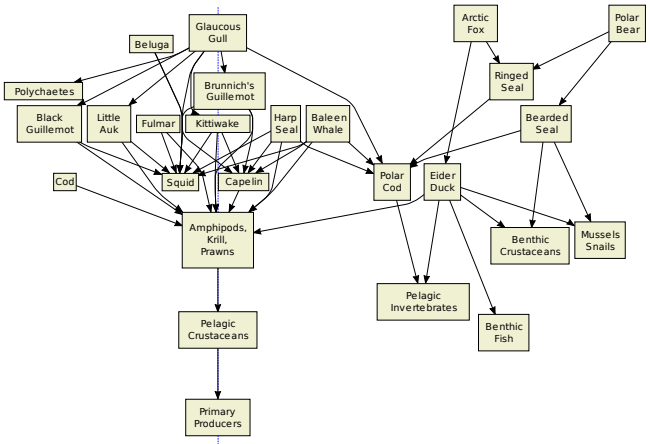
- **Flow-style layout.** Flow layout is handled identically to Organic-style layout except that the tool first adds downward pointing connector constraints.



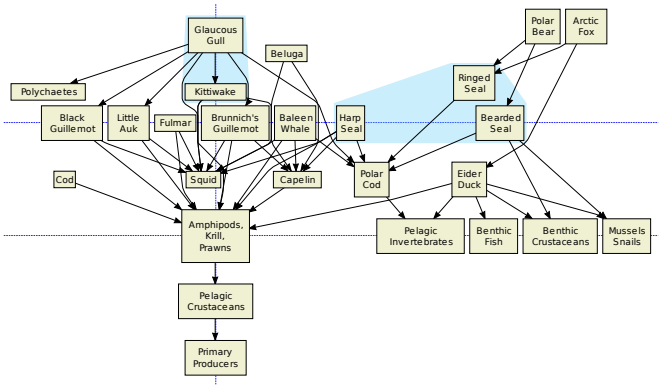
(a) Organic layout style



(b) Layered layout style



(c) Flow layout style



(d) Final layout

(a) The author initially calls the structural layout tool with *organic layout* style. Note how various automatic refinement constraints such as non-overlap of nodes keep the layout tidy.

(b) The author tries the *layered* layout style. This generates a set of horizontal alignment constraints with separation constraints between them. The separation constraints keep the layers in order and enforce a minimum spacing (adjustable by the user) between layers. The author has added a vertical alignment constraint to several nodes to improve the layout.

(c) Unhappy with the result, the author now tries the *flow* layout style. This generates style constraints requiring that directed edges be downward pointing. The minimum separation between nodes connected by directed edges can be adjusted live via a slider. Note that the vertical alignment constraint was maintained when switching layout styles.

(d) Reasonably happy, the author now fine-tunes the layout. They first add two new horizontal alignment constraints. They then cluster together three seal species and two types of gull. As a result the position of several nodes and edges are automatically updated to remove overlap with the clusters. Finally, the author repositions the "Beluga" and "Arctic Fox" nodes to remove some edge crossings.

Figure 6.10: Example of interactive network layout with Dunnart. Network data from the Many Eyes "Arctic food chain" visualisation, `http://www.many-eyes.com/`.

- **Layered-style layout.** Layered layout makes use of the Graphviz[6] library to produce an initial layered digraph layout. Graphviz determines a layer for each object in the network, the ordering of objects on each layer and a routing for connectors through the layers which minimises crossings. An alignment placement constraint is generated for each layer and a separation constraint between each pair of layers keeps them a minimum distance apart and preserves the layer order. Currently, existing user-specified placement constraints are ignored in this style since they can conflict with the level and ordering constraints specified by Graphviz.

Many structural styles can easily be handled by the model, requiring only that they can be expressed using separation constraints and/or via a modified optimisation function. Candidate structural styles that could be added to Dunnart in the future are orthogonal layout and circular layout (where the graph is divided into structural partitions and each partition is arranged separately as a circle).

Some structural layout styles that are based on force-directed layout will automatically maintain their structure during further editing. Others, such as Layered-style layout that provide their own routings or place certain nodes in unbalanced positions need to have their topology preserved. In this situation Dunnart uses the edge straightening technique described in Section 6.3.4. By taking the connector routing produced by the structural style into account when generating edge straightening constraints Dunnart is able to adjust the layout of the resulting diagram while maintaining its topology.

6.7 Showing constraint status

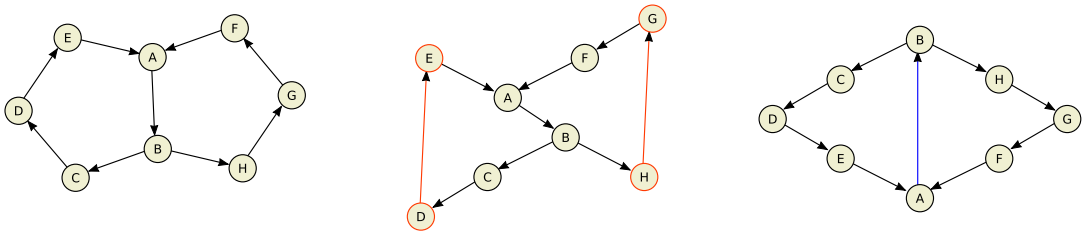
An important concern in constraint-based systems is visualising the state of constraints to the user. An on-screen representation of constraint-based relationships clearly identifies them to the user. A constraint may be tight,[7] slack, or unsatisfiable due to conflicts, and its changing status will impact on its behaviour. Hence, the user must be aware of constraint status, but at a high-level—they should not need to understand implementation or behaviour of the underlying constraints or solver.

Tools that are implemented with equality constraints, such as alignments and distributions, affect the diagram at all times. Their constraints are always tight. We visualise them via a permanent on-screen indicator. In Chapter 4, we have talked about high-level placement relationships being active when they are causing other objects in the diagram to be moved. We highlight this distinction to the user by slowly fading out inactive placement relationship indicators over time, and bringing them back to full visibility when they are active.

On the other hand, tools implemented with separation constraints may not always have an effect on the diagram. These constraints are tight when at equality; non-overlap constraints do nothing until two shapes are pushed together, and then one must be moved

[6] Graphviz, AT&T Research, `http://www.graphviz.org`

[7] The literature refers to constraints as *active* or *inactive*. We instead refer to these states as *tight* or *slack* to avoid confusion when discussing placement tools that are themselves "active" when exerting an effect on the diagram.



(a) (b) (c)

Figure 6.11: (a) A directed graph with cycles. (b) When downward pointing connector constraints are applied to the graph graph, IPSep-CoLa randomly selects a constraint—here shown in red—to drop to make each cycle satisfiable. (c) The user will often make better choices, so they are allowed to mark individual connectors (shown in blue) so that downward pointing connector constraints are omitted for them.

out of the way to prevent overlap. We show whether constraints are tight in different ways for different tools. For separations, when two shapes are pushed together to their minimum spacing, the relevant span of the separation indicator is highlighted to show it is keeping the objects apart.

Page boundary constraints become tight when a shape is pushed against the edge of the page. If that shape can be pushed back onto the page by the page boundary then we don't show any visualisation, though we could do this to show the user that an object is pushing against the boundary. If the object cannot be pushed back within the standard page boundary, then we show the ballooned-out page boundary as well as its ideal position. This shows the user that these constraints are tight, gives a sense of stress and an elastic feel to the page edge.

We do not visualise non-overlap constraints whether they are tight or slack. We are able to do this since the action is like real-world interaction of solid objects and the active state of non-overlap can be immediately determined when objects are indirectly moved as a result of direct manipulation. Again, it would be trivial to add a temporary indicator for collisions but this seems unnecessary given the effect is immediately visible to the user. Ideally, this assumption will be tested by a future usability experiment.

Downward pointing edge constraints currently have no indicator to show they are tight, except the resulting movement of nodes in the diagram. During construction they have a significant effect on layout, so are not easily forgotten. At other times they may not be so obvious to the user, and would probably benefit from a visualisation indicating their activity. We have not yet done this, but plan to in the future.

When a system generates a large amount of constraints and allows the user to specify more, unsatisfiable constraints may sometimes occur. Unsatisfiable constraints can be caused by constraint cycles or conflicting equality constraints. An example conflict would occur if two shapes, joined by a directed connector, were horizontally aligned and then downward pointing connector constraints were enabled. To be satisfied, the downward pointing connector constraint needs the two shapes to have a minimum vertical separation between them, but the horizontal alignment specifies that they have no vertical separation.

In a graph containing cycles of directed edges, some downward pointing connector constraints will be unsatisfiable. Since not all edges in a cycle can point downward; there will always be at least one conflict. There are several possible approaches we can take to deal with this problem:

1. Find the *maximum acyclic subgraph* omit constraints for edges that are not in the subgraph. Though the maximum acyclic subgraph problem has been shown to be NP-hard (Karp, 1972), there are several heuristic approaches that give good results in practice, see Bastert and Matuszewski (2001).

2. Find *strongly connected components* (SCC) of the graph, and omit constraints for all edges in an SCC. A strongly connected component is a subgraph in which each pair of vertices have a path to each other. SCCs can be computed in linear time (Mehlhorn, 1984).

3. Highlight SCCs and allow the user to choose which edge to omit constraints for.

4. Allow the solver to detect infeasible constraints and disable them as they are found.

IPSep-CoLa takes the fourth approach, but this is not ideal since it results in a random and sometimes sub-optimal choice of constraints to omit, see Figure 6.11. The particular connector chosen to break the cycle can have a large effect on the subsequent readability of the diagram, hence it would be preferable to let the user make the choice. This is especially true where one edge may be selected to break multiple cycles. For this reason Dunnart includes an algorithm to detect cyclic connectors (Simpson, 2006). These are then visualised so that the user may easily choose which connectors to omit downward pointing connector constraints for.

6.7.1 Future work

As stated earlier, the layout thread is halted whilst the user is ALT-dragging. ALT is also used as a modifier key to allow the user to drag constrained objects free of their placement relationships, as described in Section 4.2.1. Since constraints for placement tools are now maintained by the layout thread, dragging a guideline with ALT held down—to remove it from a distribution—will not immediately move the shapes attached to it, and they will jump to align with it when ALT is released and the layout thread restarts. In this situation it would be preferable for layout and stylistic constraints to be halted, but for placement constraints to still be solved. We plan to implement this change in the future. Further work could involve usability evaluation to determine the mos useful behaviour for the tools in this situation.

Currently, cycles of directed connectors in Dunnart are visualised by highlighting the connectors and shapes involved to the user in-place—that is, wherever they are located in the current layout. It would possibly be more useful to the user to see a structural view of the cycles, say via a force-directed layout. This could be shown along with the canvas view, with some visual cue to aid the user in recognising the same object across the two views.

Automatic layout could be used to quickly allow the user to see the results of choosing a particular connector to omit constraints for. We could automatically run a force-directed layout without non-overlap constraints and connector straightening, followed by another with non-overlap constraints and then straightening (if they were being used). This process would allow the diagram to properly rearrange itself, which would not be possible with non-overlap or straighten constraints preventing shapes from sliding past each other.

When IPSep-CoLa encounters a cycle of conflicting constraints it disables one of the constraints in the cycle and continues. When it completes its iteration and returns a layout it reports all the conflicting constraints as well as which ones were dropped. At the moment, Dunnart uses this information only to highlight placement indicators or diagram objects involved in conflicting constraints. It would be much more desirable to show some sort of overlay for the conflicts, where the users could be shown say a list of constraints involved in the conflict, in a high-level form, for example, "alignment" and "downward pointing connector". They could then move their mouse over each of these in the list and have the corresponding shapes or placement indicator be highlighted. In this way they could hopefully understand the conflict and resolve it.

We have built into Dunnart a Heads-Up Display (canvas overlay) feature. The layout engine can return special dummy objects in the position list that represent a visual notification. These are then drawn onto the canvas by Dunnart for a set amount of time. This feature will allow us to create more detailed visualisations for conflicts, such as the menu mentioned above.

Currently, IPSep-CoLa returns only the low-level separation constraints, rather than relationship type for the constraint such as "alignment" or "downward pointing edge". This is because a graph layout layer in Dunnart converts the high-level relationships down to separation constraints before passing them to `libcola`. To provide better constraint conflict resolution to the user we really need to be able to report the conflicts as high-level constraints. An easy solution to this would be for each constraint in `libcola` to be assigned an ID. Dunnart could then keep track of which constraint IDs correspond to which placement relationship or layout style.

Possibly we could also assign priorities to certain constraints, so that when IPSep-CoLa encountered a cycle of conflicting constraints it could selectively drop the least important one, rather than an arbitrary one. Some work would have to be done to determine the priorities for different kinds of constraints.

6.8 Discussion and case studies

In this section we discuss informal observations and feedback from users of the continuous network layout model in Dunnart. We also present three case studies exploring the use of constraints and continuous network layout in the important application domains of software engineering and biology. For each case study we collaborated with a domain expert to determine the requirements for drawing a certain class of diagram. In each, we examine how easily continuous network layout with constraints can be used to produce

the required style of diagram, as well discussing any necessary modifications to Dunnart to produce them.

Here, we again present informal user feedback rather than the formal usability evaluation of the kind given in the early chapters. The reason for this is purely that the continuous network layout model is new, unique, and in a prototype form. The observations and user feedback given in this section can be thought of as the groundwork for formal usability testing that should follow as future work.

6.8.1 General observations and discussion

These observations arise from demonstrations of the continuous network layout model to various individuals (approximately thirty people), both visiting academics and colleagues at conferences and workshops. During demonstrations, we noted positive and negative reactions of users to the system, as well as any feedback or difficulties encountered.

In general, we have found the continuous layout model to provide a usable and comfortable environment for creation and improvement of diagrams or graphs. We find it gives the user adequate freedom to encode aesthetic preferences and drawing conventions. It provides enough interaction and control to feel like a collaborative process guided by the user.

We found separation placement relationships to be useful for network layout as well as in general diagramming. Users understood their purpose and behaviour and did not have any issues interacting with their on-screen representation. This is not altogether unexpected, since their representation and behaviour is quite similar to the distribution tool.

In our observations we have found the anchoring tool is only minimally used in general editing. This may be because what users really want is a way to lock shapes together relative to each other, so they can all be moved but effectively their relative layout is finalised. This tends to be the case when the user lays out a portion of the graph in a certain way to a stage they are happy with and then wishes to preserve it while laying out other parts of the graph in different ways. The tool is still very useful for certain cases but this might be an argument for moving it to context menus of shapes and developing a more useful general purpose layout locking tool.

From our observations and informal feedback, users were comfortable using the stylistic constraints of non-overlap, page containment and directed edges. Users had no problems understanding the behaviour of non-overlap constraints, even though there was no on-screen indicator for their presence. Since the behaviour of non-overlap constraints during interaction is analogous to physical shapes in the real world, users were able to quickly comprehend and work with this style enabled.

We similarly found page boundary constraints to be well understood by users. Drawing the page "ballooning out" when the page boundaries could not be placed at their ideal positions was an adequate indicator of the increased stress and forces being exerted on unconstrained shapes to bring them back within the bounds of the page.

We found downward pointing connector constraints to be well understood by users when they had enabled them themselves or used them during the diagram construction phase. We observed confusion by users working with pre-prepared diagrams, where they would move a shape and several other shapes would move to stay below each other. This likely occurs because it may not be obvious from looking at a static diagram that it was constructed with downward pointing connectors constraints. Also, this type of constraint is not particularly common and might be a foreign concept to the user. We often found that confusion occurred because the user didn't attribute the unexpected movement to downward pointing connector constraints. Once they determined, or were instructed, that these were the culprits, they turned them off (or adjusted their actions accordingly) and carried on without issue. We plan to add an overlay visualisation that shows up for tight downward pointing connector constraints—when they move unselected shapes—so the user can better understand the reason for the movement.

In general, users were comfortable working with the force-directed layout and did not have issues with the rest of the layout updating in the background as they made modifications. There are certainly cases where users do not want this automatic network layout happening, and the ALT key and toolbar option to disable automatic network layout satisfactorily allowed the users to disable it when desired. The topology preserving aspect of the edge straightening beautification tool has been found to be very useful. We have found the force-directed layout to be used at an early stage to find a basically good layout, and then users will apply placement constraints and use beautification to apply slight improvements to the layout. It is also invaluable when working with a layout produced by an external layout algorithm, especially if it supplies its own routings for connectors.

6.8.2 Case study: Biological networks

In collaboration with Dr Falk Schreiber[8] at the Leibniz Institute of Plant Genetics and Crop Plant Research Gatersleben (IPK), Germany, we investigated the use of constraint-based network layout for biological networks. We investigated two common biological networks, metabolic pathways and gene regulatory networks, and found we were able to capture their characteristic drawing conventions using constrained stress majorization within our continuous network layout model.

Metabolic Pathways

A metabolic pathway is a sequence of chemical reactions that occur within cells of living organisms as part of the process of metabolism. When drawn as a diagram, nodes represent metabolites (substances that take part or are produced during metabolic processes) and edges represent reactions that transform them.

In the past such metabolic pathways have been arranged with different methods such as combinations of linear, tree and circular or layered layout, modified at the algorithm

[8]Dr Falk Screiber, `http://bic-gh.ipk-gatersleben.de/~schreibe/`

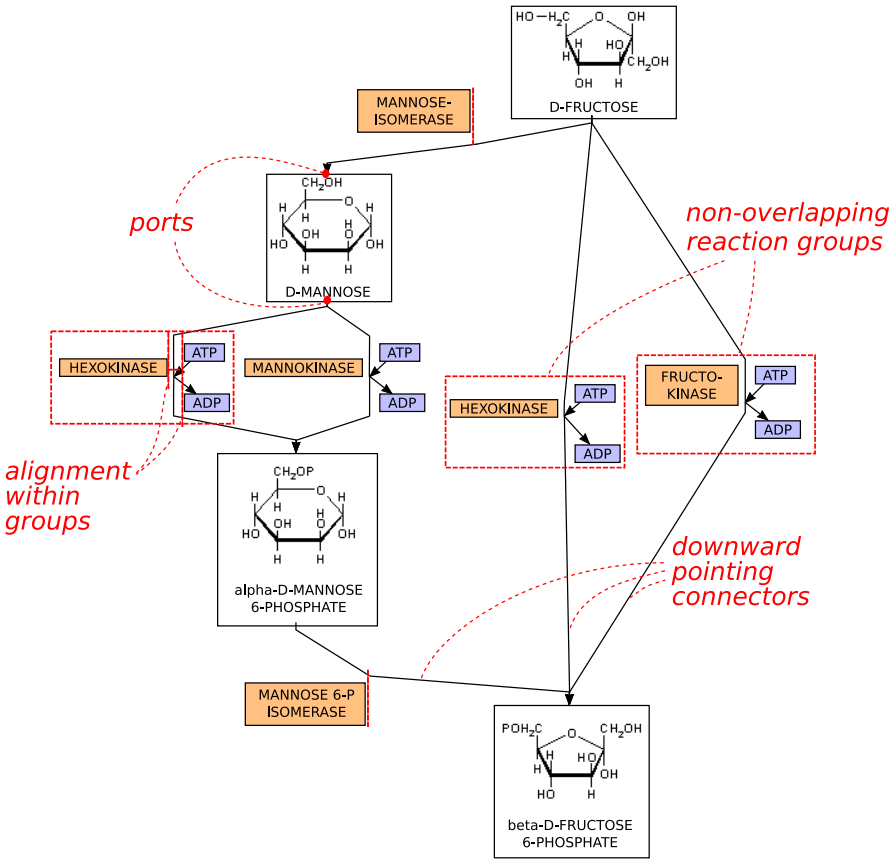


Figure 6.12: Basic use of constraints to capture the drawing conventions required for a small metabolic pathway.

level to handle nodes of highly irregular height and to ensure the correct style for the reaction subgraphs (Becker and Rojas, 2001; Karp and Paley, 1994; Schreiber, 2001).

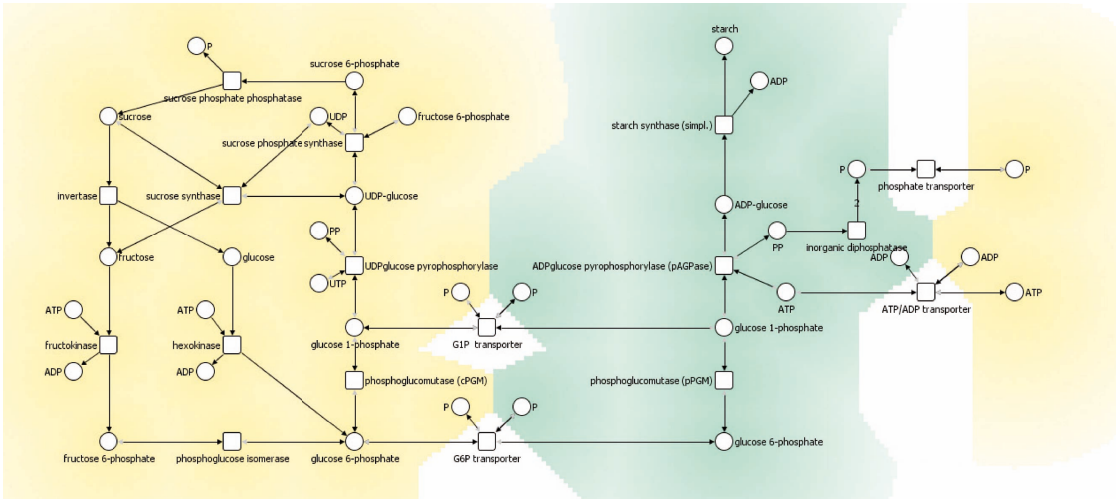
Figure 6.12 shows a metabolic pathway with a fairly straightforward use of constraints to capture standard drawing conventions.

Separation constraints are used to ensure node placement so that edges point downward to indicate the usual direction of reaction. In addition, dummy *port* nodes have been defined and constrained to lie on the top or bottom of the metabolite nodes. Incoming edges for each metabolite are then connected to the top port and outgoing edges to the bottom.

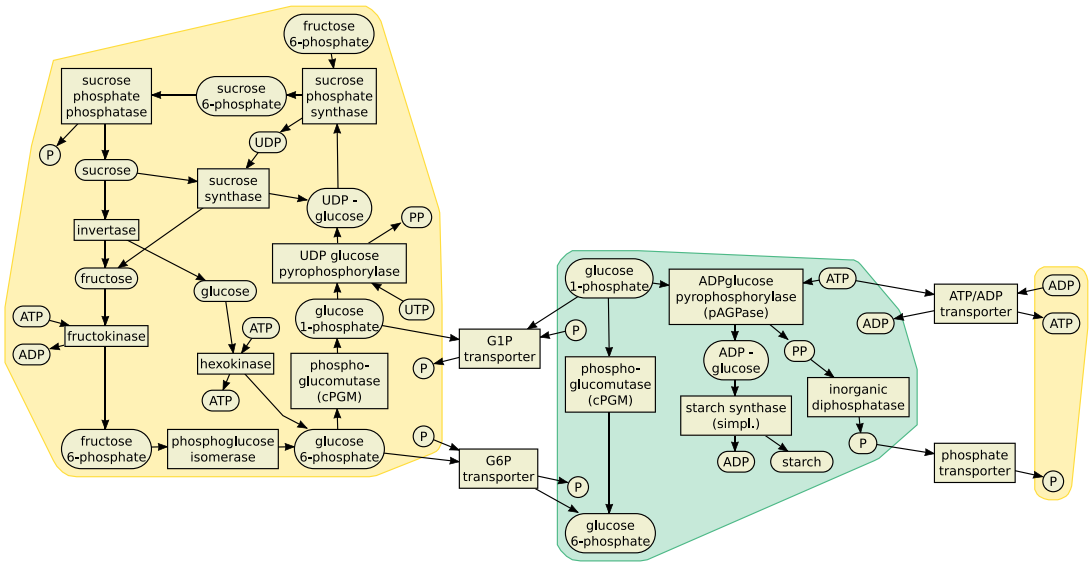
The co-substances associated with each reaction form small subgraphs around dummy nodes in reaction edges. Co-substances for a particular reaction are vertically aligned and then the reaction subgraph is enclosed in a rectangular boundary to prevent overlaps.

Figure 6.13(b) shows a larger metabolic pathway that demonstrates a more sophisticated use of constraints. Figure 6.13(a) shows a version of the same metabolic pathway, hand-drawn by a biologist. Such hand-drawn diagrams are currently all that is offered by many online repositories such as the KEGG (Kanehisa, Goto, Hattori, Aoki-Kinoshita, Itoh, Kawashima, Katayama, Araki and Hirakawa, 2006) or MetaCrop[9] databases.

[9]Figure 6.13(a) is from the MetaCrop database: `http://metacrop.ipk-gatersleben.de/`

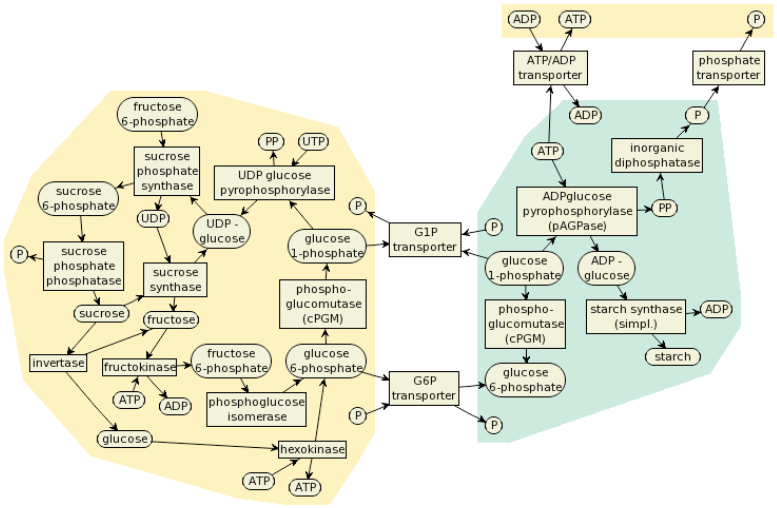


(a) Hand drawn diagram

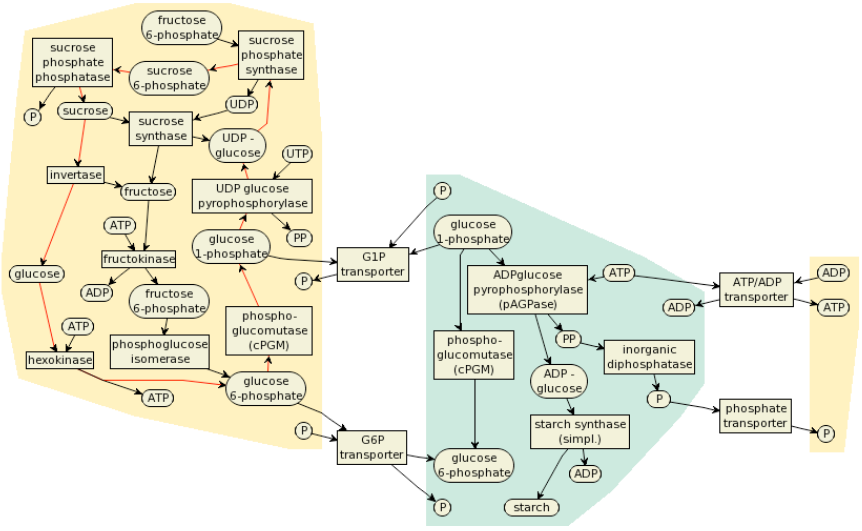


(b) Diagram drawn in Dunnart using continous network layout

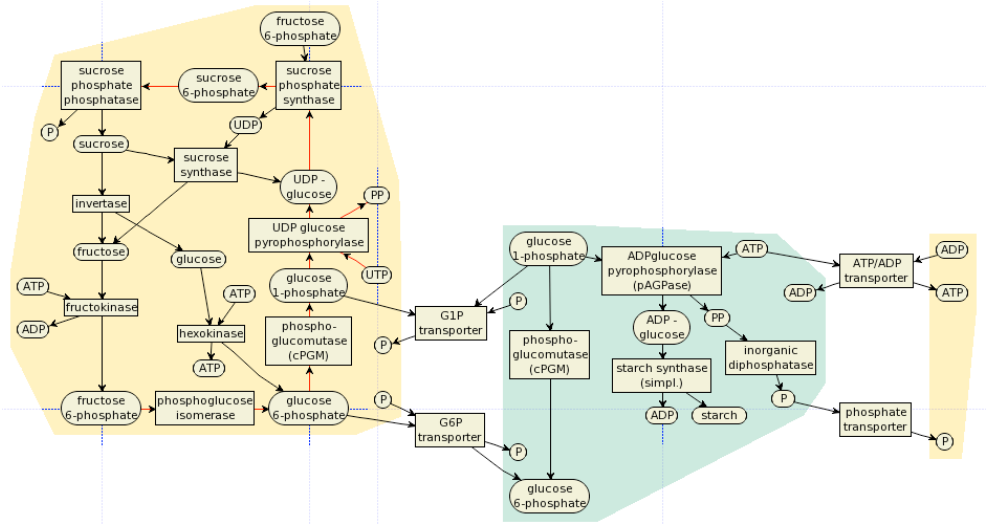
Figure 6.13: Comparison between a diagram hand-drawn by a biologist and the same diagram arranged in Dunnart using continuous network layout.



(a) Initial force-directed layout with non-overlap constraints



(b) Downward pointing connector constraints have been added



(c) Alignment relationships are added and edge straightening further improves the layout

Figure 6.14: Screenshots showing the use of continuous network layout to produce the metabolic pathway diagram shown in Figure 6.13(b). The user has manually marked the red connectors to have downward pointing connector constraints omitted for them.

The metabolic pathway in Figure 6.13(b) was drawn in Dunnart using continuous network layout, see Figure 6.14. Initially, in Figure 6.14(a), the author imports the initial diagram and lays it out with just a force-directed layout with non-overlap constraints enabled. Next, in Figure 6.14(b), the author enabled downward pointing connector constraints. They then mark all the objects in the cycle of connectors at the left of the diagram (marked in red) so that they have downward pointing constraints omitted for them. This way, the layout engine is able to arrange these objects more freely while preserving downward pointing constraints for the rest of the diagram. Finally, in Figure 6.14(c), the user adds six alignment relationships to neatly align several sets of shapes.

Interestingly, the manually prepared diagram in Figure 6.13(a) displays a number of characteristics that previous automatic graph drawing systems have not been able to capture. These features, such as guaranteed non-overlapping clusters of shapes, alignment, left-of and right-of constraints, and downward pointing edges are easily handled by constrained stress majorization as part of our continuous network layout model.

It is worth noting that most of the constraints used in Figure 6.13(b) could easily be automatically generated from the definition in a biological database. This further simplifies the layout process and speeds up the initial stages of layout. Because the user can still override or replace any of the generated constraints there is no loss of control.

To produce the metabolic pathway diagrams, we required that connectors could be attached to ports on the edges of shapes. The behaviour of these ports was easily achieved in `libcola` by setting an equality constraint in each dimension to constrain the ports to their correct position on shapes.

Another requirement of metabolic pathway drawings was that we support containment, so that reaction groups could be guaranteed not to overlap. These were implemented with the same technique as non-overlap constraints. The rectangular bounding box of each reaction group was treated like a normal object in the diagram for the purposes of non-overlap. The objects in each reaction group were constrained to be inside the reaction group's bounding box with the same set of constraints as are used for page boundaries, except that the dummy variables that defined the edges of the bounding box were given a weak attractive force to keep the boundary tight around the contained objects, for example, the left and right edge of the bounding box were attracted to each other.

The final new requirement for metabolic pathways, were minimum-containment convex cluster objects, as shown in Figure 6.13(b). Their boundary was initially computed as the convex hull of the objects contained within the cluster. In the layout engine, the boundary had weak edge straightening techniques applied to it, so that it attempted to minimise its length and pull inwards. In this way, clusters were always kept tight around their contained objects, even as these moved about as a result of layout.

We can see that our model has the power and flexibility to represent the complex drawing conventions of metabolic pathways, and the user still has a large amount of freedom to affect the layout or just alter a single drawing convention—something that is not possible with prior automatic drawing approaches.

Gene Regulatory Networks

Gene Regulatory Networks (GRNs) control the speed at which DNA in a cell expresses genes. The components of such a network are DNA segments that interact with each other and other substances in the cell. Like metabolic pathways, gene regulatory networks also contain downward pointing edges. Until now, the standard approach to draw them has been the Sugiyama layered graph drawing method.

Figure 6.15 shows a gene regulatory framework drawn with our technique and with a layered graph drawing method. Our method gives a much more compact layout and layout with much fewer edge crossings that we believe better shows the network structure. This is because we do not require nodes to be strictly arranged on (artificial) layers, which leads to very wide drawings (the drawing in Figure 6.15(a) has to be scaled to quite a large size before the labels can become readable) and many edge crossings. Also, we are able to use intelligent edge routing as described in Chapter 5 to further reduce crossings and then the constrained edge straightening technique described earlier to minimize bends.

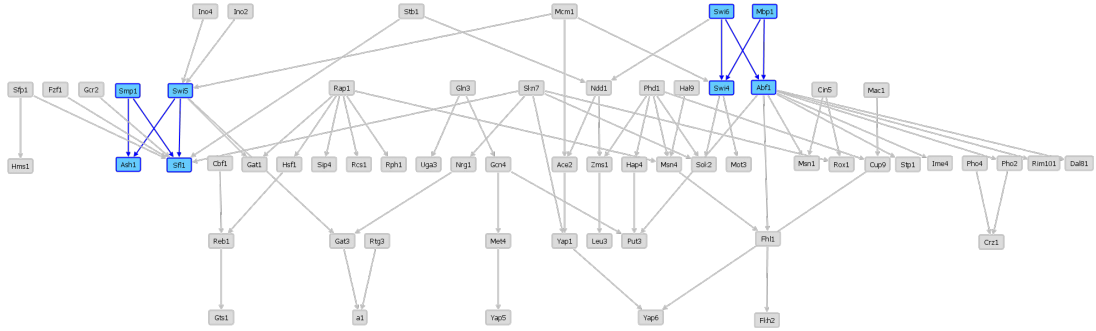
An active area of interest in bioinformatics is applying network analysis to biological networks to automatically identify structures which may be of interest. Constrained layout allows us to emphasise the results of such analysis in the visualisation. For example Figure 6.15 shows a result from a network motif investigation (Milo, Shen-Orr, Itzkovitz, Kashtan, Chklovskii and Alon, 2002) on a gene-regulatory network. In the two drawings, two bi-fan motifs are highlighted and each drawn in the same way. In Figure 6.15(a), the conventional layered drawing algorithm has been extended to draw motifs. In Figure 6.15(b), the same is accomplished using our method but with constraints used to ensure that both motifs are drawn in the same way. No special algorithmic changes were required to draw each motif the same way, merely a constraint definition as part of the input. The constraints used to achieve this for each bi-fan motif were four alignment relationships and two minimum separation relationships, as shown in Figure 6.16.

6.8.3 Case study: UML diagrams

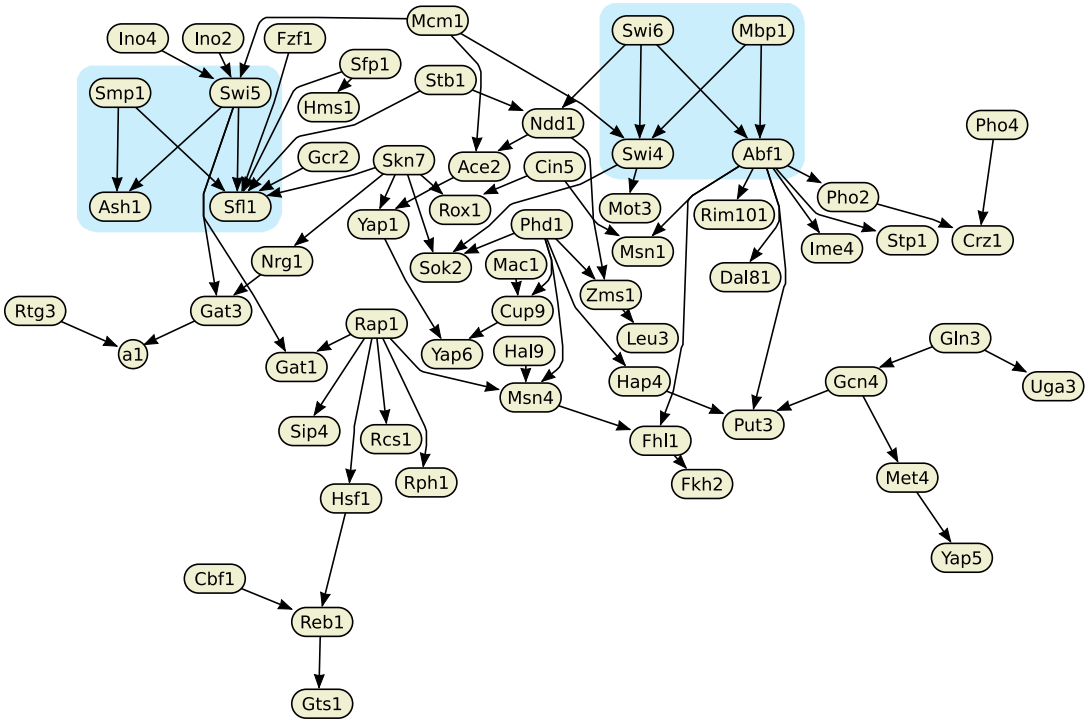
We have investigated the use of constraints and continuous network layout for construction and exploration of UML class diagrams. This work was in collaboration with honours student Michael Woodward at University of Melbourne.

Since classes in UML diagrams show all properties of a class, they can become quite large if the class has many attributes or if its method prototypes are very long. Not all of this detail is useful to the diagram author or reader at all times, thus some of it can be abbreviated or temporarily omitted. There is opportunity to show classes at different sizes and levels of detail. For example, the lowest level of detail might omit all attributes and methods completely, showing only the class name. At the next level only the names of the methods and attributes are shown. The next level of detail reveals the types of attributes and the return types of methods. Next the names of method arguments are revealed. Finally the types for method arguments are also shown.

We added to Dunnart the ability to adjust the level of detail by hovering the mouse cursor over a class and scrolling the mouse wheel. Scrolling up raises the level of detail,



(a) Layered layout



(b) Constrained stress majorization

Figure 6.15: Two drawings of a gene regulatory network with two bi-fan motifs highlighted and drawn similarly. The first drawing was produced using a standard layered graph drawing method. In such a method, special cases must be added to the underlying algorithm such that all instances of a particular motif are drawn the same way. Using continuous network layout in Dunnart, similar relative placement of nodes within motifs is ensured using alignment and minimum separation constraints. Furthermore, without the strict layering—using only separation constraints to force connectors to point downwards—a more compact layout with readable labels and fewer crossings is achieved (77 crossings in (a) compared to 11 in (b)).

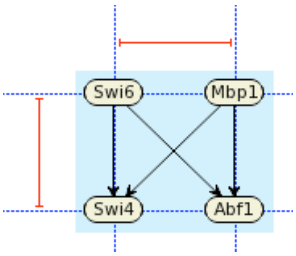


Figure 6.16: A bi-fan motif with constraint indicators showing the alignment and minimum separation relationships necessary to draw it in the desired style.

scrolling down lowers it. This is termed *semantic zooming* after the similar interaction in the early multiscale interfaces Pad (Perlin and Fox, 1993), Pad++ (Bederson and Hollan, 1994) and AGE (Sengupta, Kimura and Apte, 1994).

The idea is that once the user has semantic zooming for classes available to them they will mostly work with classes collapsed down to a small size so they can better visualise the structure of the system. Then, when they are interested in particular classes they can zoom those and other classes to a higher level of detail and work with them at that size.

This means they will be frequently adjusting the sizes of classes while authoring or interpreting the diagram. Obviously, raising the level of detail for a class by semantically zooming it causes the class to grow significantly in size. Non-overlap constraints become very useful in this situation since they move classes that would otherwise be obscured out of the way. This works fairly well. With sticky nodes enabled, classes are moved out of the way of the expanding class, but the placement of the rest of the layout is not affected much. When shapes are collapsed, they drift slowly back to their ideal position.

A diagram with a high degree of orthogonality is one where most shapes are horizontally or vertically aligned, leading to lines of centre-aligned objects at right angles. In this way, the diagram will have more of a grid-like arrangement and connectors will usually be vertical or horizontal lines. It has been shown that *orthogonality* is one of the most highly preferable aesthetic criteria for UML diagrams (Purchase et al., 2002). Conveniently, alignment relationships can be set up in Dunnart to enforce orthogonality. When the user semantically zooms classes, the non-overlap and alignment constraints cause neighbouring nodes to spread out either horizontally or vertically, as seen in Figure 6.17. This both preserves the orthogonality of the diagram and makes the movement more predictable. Page boundary constraints or force-directed layout ensures classes return back to a logical position when other classes are collapsed and more space is made available.

Woodward (2007) developed a proximity-based alignment algorithm for automatically creating the alignment relationships necessary to make a UML diagram orthogonal. When run, it looks at the structure of the graph, as well as the current positions of classes and tries to place them orthogonally, aligned with their neighbours where appropriate, which attempting to prevent any additional crossings as well as minimise the movement of classes from their original position. This works well with our model, since the user can manually adjust the alignment relationships where they felt the automatic procedure did not do an optimal job.

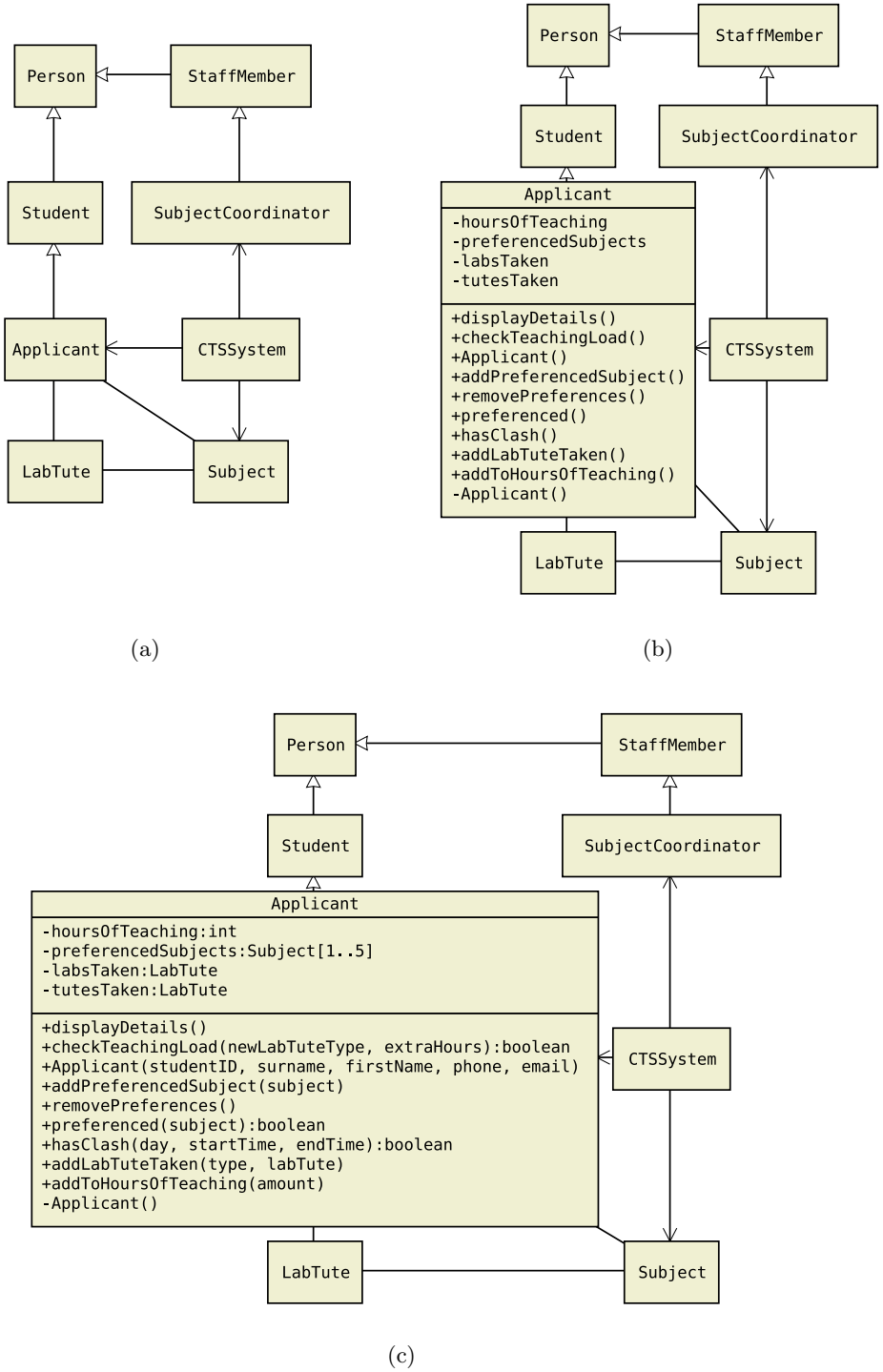


Figure 6.17: Semantic zooming of the Applicant class. Alignment and non-overlap constraints automatically preserve orthogonality of the diagram as the class is expanded.

Connectors in UML diagrams represent different relationships between classes. These relationships may have a text label shown at each end of the connector near where they attach to classes, representing multiplicities (e.g. see Figure 6.18). They may also have a label shown near the connector's centre, denoting its role. Ideally these connector labels should be placed close to their ideal position, except when this would cause a label to overlap another object. We can represent this behaviour for labels within our model with a two step layout process. In the first step layout is done as normal, but labels are completely ignored. The second step is to lock the position of diagram shapes and execute layout again with non-overlap constraints and labels initially place at their ideal position. This layout operation will move labels as necessary to present overlap, and edge straightening constraints and forces can be used to make sure the moved labels do not overlap connectors.

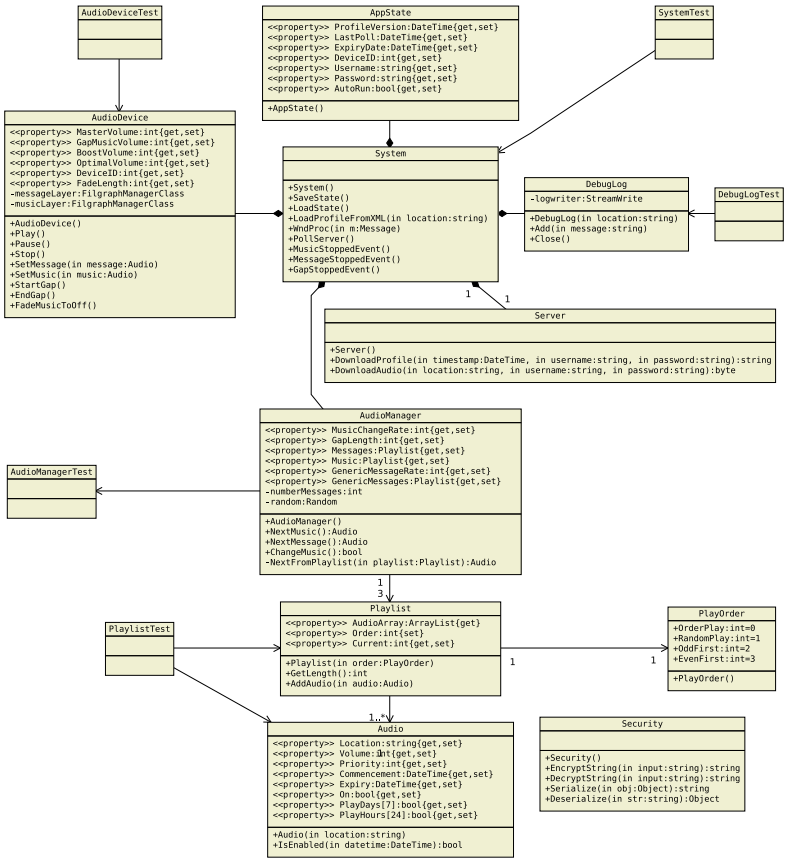
The continuous network layout model is also useful if the user wants to prepare the orthogonalised UML diagram for printing. Typically they will want to print it with classes shown at their maximum level of detail. To do this they can take their orthogonalised diagram, turn on page boundary constraints and then expand all classes to show their full level of detail. Probably some classes will be wider or higher than others and the user will likely have to make a couple of alterations to the alignment relationships to best fit the diagram on the page. The flexibility offered by the layout model is important in this case, since the user can easily halt the layout, drag classes off one guideline and attach them to another in a more ideal position. The automatic network layout will then continue, ensuring classes don't overlap and pushing classes together to best fit within the page area. See Figure 6.18 for a visual example of this kind of interaction.

6.8.4 Case study: Protein topology diagrams

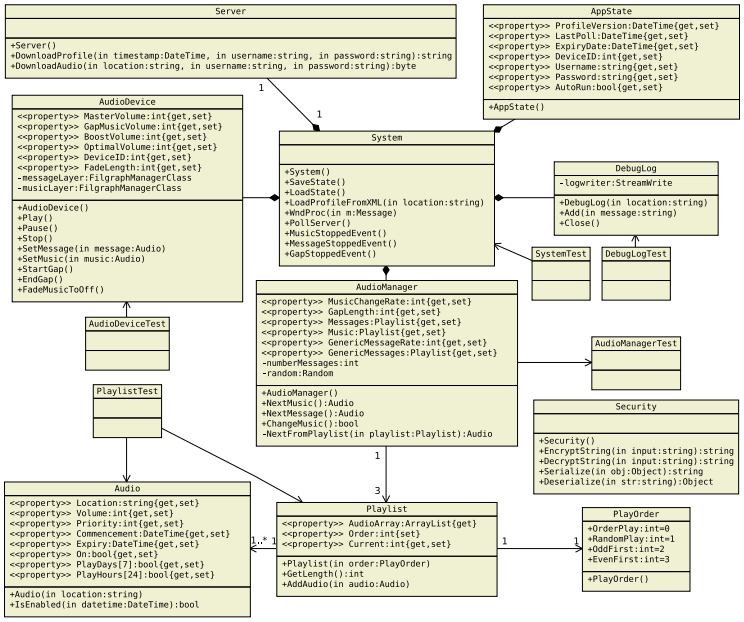
With Alex Stivala, a Masters student at the University of Melbourne, we have investigated the use of constraints and continuous network layout for assisting in the generation of protein topology cartoons.

A protein is an organic compound consisting of a linear chain of amino acids joined together via peptide bonds. In its natural state each protein will fold into a unique 3-dimensional structure. This structure is characterised by several regularly repeating sub-structures formed by weak hydrogen bonds: *α-helices* and *β-sheets*. An α-helix occurs when a section of the strand forms a right-handed coil, resembling a spring. A β-sheet is made up of β-strands—an extended stretch of around 5–10 amino acids—joined along their length by hydrogen bonds to form a sheet, often itself twisted. Biochemists refer to the arrangement of α-helices and β-sheets for a protein as its *secondary structure*. This secondary structure is useful to illustrate the correspondence between a protein's one-dimensional amino acid sequence and its three-dimensional folded structure.

Protein topology cartoons are a type of diagram that show secondary structure for proteins. They display α-helices as cylinders or circles, and β-strands are arrows or triangles. β-sheets are shown by the placement of β-strands next to each other, sometimes with a bounding shape drawn around them. The rest of the amino acid strand is just shown



(a) UML diagram after expanding all classes to their highest level of detail



(b) Compact layout suitable for printing, after manually repositioning several classes

Figure 6.18: (a) An orthogonally constrained UML diagram at its full level of detail for printing. (b) The user can arrange the diagram more compactly by holding ALT to break alignment constraints and by manually repositioning several shapes.

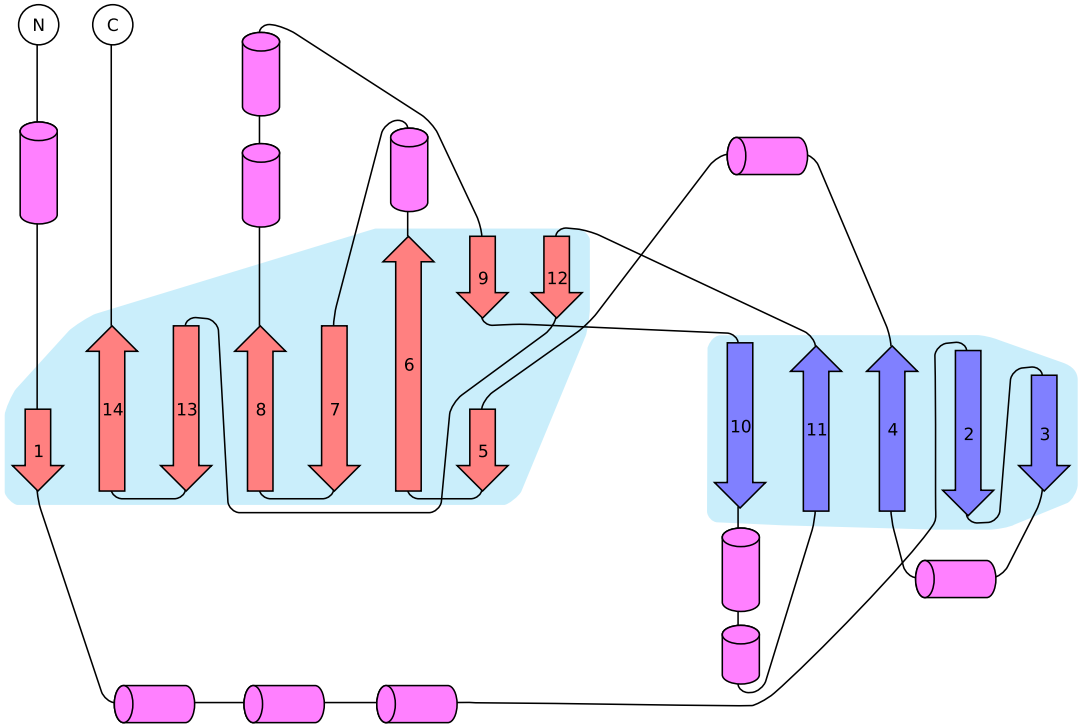


Figure 6.19: Protein topology diagram for the protein α_1-antitrypsin, drawn in Dunnart.

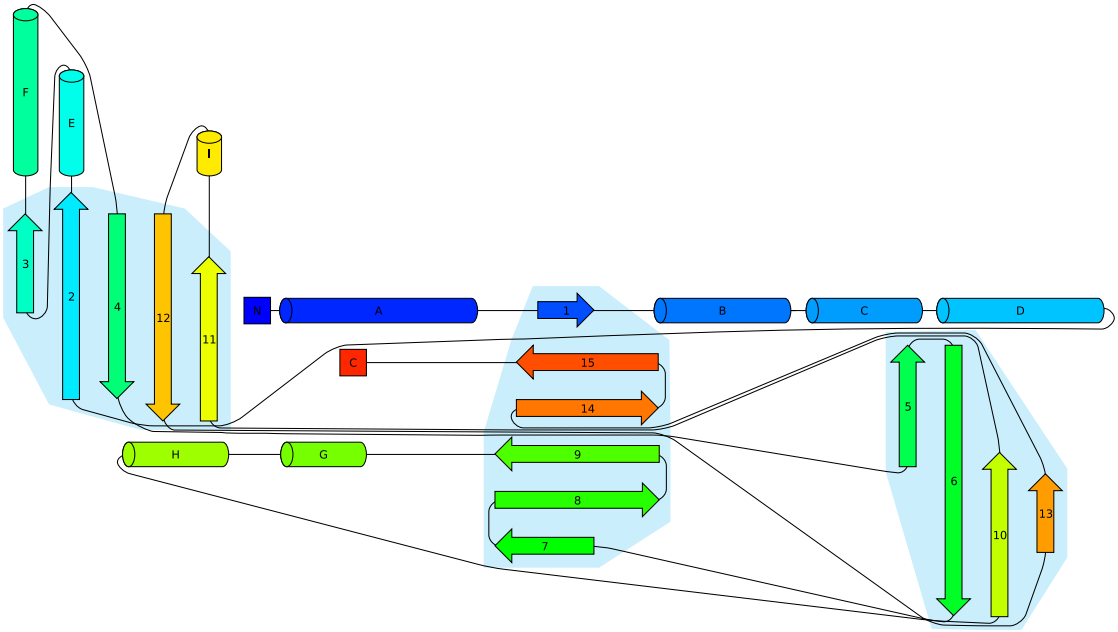


Figure 6.20: Another protein topology diagram for the protein α_1-antitrypsin, drawn in Dunnart. This diagram shows a different interpretation of the protein's structure, based on a manually edited description specified in the Protein Data Bank (PDB). This protein has the PDB identifier "1QLP".

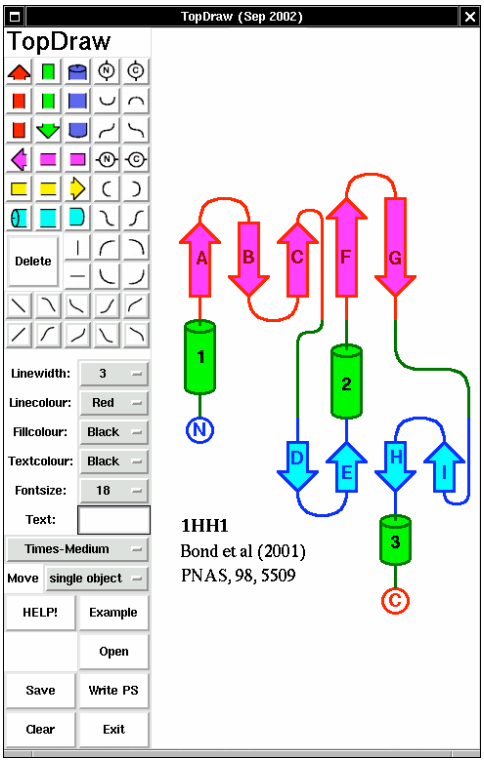


Figure 6.21: TopDraw, a sketchpad for drawing protein topology cartoons.

as connectors between these components. Figure 6.19 shows a such a diagram for the structure of the protein α_1-antitrypsin (Elliott, Pei, Dafforn and Lomas, 2000) generated in Dunnart. The two groups of arrows in the figure denote two distinct β-sheets. Figure 6.20 shows a different structural interpretation of α_1-antitrypsin, manually specified by a structural biologist and then drawn in Dunnart.

The folded structure for a protein can be accurately determined via a process called X-ray crystallography, where the protein is crystallised and X-rays are projected through it. The diffraction pattern of X-rays scattered by elections can be used to determine the exact positions of each individual atom in the protein. These solved structures are freely available in a database called the Protein Data Bank (PDB).[10] The information in the PDB is then used to generate various visualisations of proteins.

To draw protein topology diagrams the secondary structure of the protein must be determined, that is, α-helices and β-sheets need to be identified. This can be done with an automated process such as DSSP (Kabsch and Sander, 1983) or STRIDE (Frishman and Argos, 1995). There are also some automatic drawing processes; for example, TOPS (Flores, Moss and Thornton, 1994), but these do not usually produce ideal results. For this reason, many protein topology cartoons in literature have been drawn manually in the specialist sketchpad TopDraw (Bond, 2003), sometimes using the output of software like TOPS as a starting point. In TopDraw the user must manually place all diagram components, see Figure 6.21.

[10]RCSB Protein Data Bank, `http://www.pdb.org/`

We have investigated using STRIDE for secondary structure assignment, and using this information to generate a set of constraints reflecting the drawing conventions for protein topology cartoons. The actual layout can be done together by Dunnart and the user, while distribution constraints are used to preserve the spacing of β-strands in β-sheets, exact spacing constraints (a distribution between just two shapes) are used to position β-stands relative to each other in the other dimension, and alignments are used generally to keep β-stands and α-helices neatly in line. The results of these constraints can be seen in Figure 6.19.

An added benefit under our model is that the user is free to override the generated constraints, such as splitting a β-sheet into two separate β-sheets. This is highly relevant since automatic secondary structure assignment techniques like DSSP and STRIDE do not always agree with each other or with the opinion of a structural biochemist determining the features themselves. Therefore it is a common situation for the biochemist to want to adjust the generated structure of the diagram as well as the layout. With our model they have this flexibility.

6.9 Conclusions

We have seen that many popular pieces of diagramming software lack support for any kind of automatic network layout. Those that do provide network layout often have only tools that do a single complete re-layout of the diagram each time they are invoked, discarding alterations made by the user. While there has been specific research on dynamic graph layout, these techniques often only extend existing algorithms with the aim of increasing their dynamic stability. They focus on preservation of the user's mental map but many fail to allow true user interaction that can influence the produced layout.

In this chapter we have presented the continuous network layout model for integrating automatic graph layout into diagramming tools. We demonstrated how separation constraints can be used to represent global stylistic constraints as well as manually specified placement constraints.

In the continuous network layout model the graph layout engine runs continuously during user interaction. It constantly improves the layout in response to structural changes or manipulation of diagram objects. Throughout this process the layout engine maintains the constraints that have been placed on the diagram. This saves the user from repeated actions that would otherwise be required to enforce drawing conventions or aesthetic preferences. They can instead delegate the responsibility for maintaining these relationships to the software.

The continuous network layout model is flexible, allowing external graph layout techniques to be used to provide the initial layouts and define a set of separation constraints. Edge straightening techniques can then be used to perform small beautifications on a layout while preserving its topology. The user is given additional freedom to improve the layout by manually making adjustments to the topology. These changes can then be preserved throughout further editing.

Chapter 7

Conclusions

Diagrams are important and widely used. Good diagram layout depends on several factors: Drawing conventions, aesthetics and personal taste. Manually arranging components without any assistant is tedious and frustrating for the user. Many of the geometric relationships that a user would want to represent can be expressed as constraints. For example, connector endpoints attached to a shape, or two objects in alignment with each other.

This thesis has investigated the use of constraint-based approaches to improve the usability of diagramming authoring software. This chapter gives a summary of the findings and contributions. It discusses the research results from each of the problem areas examined: placement tools, connector routing and interactive network layout. Ideas for further work have been previously outlined in individual chapters, but are again summarised here.

7.1 Constraint usage in diagram editors

Chapter 2 has shown that the application of constraints for graphical editing purposes has been widely studied. Research systems vary greatly: they utilise different constraint technologies and their tools have vastly different behaviour and interfaces.

We saw that the focus of this research has been on multi-way constraint solving technologies that can be used to persistently maintain geometric relationships. These techniques are not utilised by popular diagramming software. Instead, such editors offer single-effect tools, or sometimes tools based on one-way constraints. Single-effect tools just move objects and result in a viscous system where the user is required to reuse them repeatedly to maintain their desired layout. One-way constraint-based tools maintain relationships for the user, but have their own usability issues; breaking easily and suffering from hidden dependencies.

We noted the main reasons why tools from research systems have not made the jump into popular commercial and open source diagram editors. The primary concern is that there has been almost no formal evaluation involving the usability of the prototype systems or the general worth of constraint-based tools for diagramming. As a direct result, there is no clear consensus on the form that constraint-based tools should take—that is, their behaviour and the interface to the user. The secondary concern is the complexity in

implementing constraint solvers that are fast and flexible enough for use in interactive applications.

7.2 Placement tools in Visio

In Chapter 3, we designed and conducted a usability study of constraint-based placement tools in Microsoft Visio. We compared Visio's native once-off and one-way alignment and distribution placement tools with multi-way versions. We implemented a set of multi-way constraint-based placement tools as an add-on for Visio, using the QOCA constraint solver.

The study examined the relative usefulness of different kinds of constraint-based placement tools for the general task of constructing and editing diagrams. Its results support the hypothesis that placement tools based on one-way constraints have usability issues and that multi-way constraint-based placement tools offer significant benefit over one-way constraint-based tools.

They results showed persistent placement tools based on one-way constraints offer no significant advantage over the simple, once-off tools offered by nearly all diagram editors. Multi-way constraint-based tools were not found to offer a statistically significant advantage over once-off tools in all tasks, though in tasks requiring alignment and distribution they consistently resulted in faster average completion times and fewer errors in the final diagram.

In analysing the problems encountered by participants, we determined that some of the usability issues in Visio's one-way constraint-based tools were a result of the Visio implementation, rather than an intrinsic limitation of one-way constraints. The major issues were tools unnecessarily breaking one-way constraints even when they didn't conflict, and Visio only providing delayed feedback for movement operations once they were completed.

7.3 Placement tools in Dunnart

Chapter 4 described the design of new, more usable multi-way constraint-based placement tools, along with several more usability studies. The tools were implemented within Dunnart, a new prototype constraint-based diagram editor. Dunnart was written from scratch to provide a platform for constraint-based diagramming. It also contains instrumentation features useful for conducting usability experiments.

The first study was a formal experiment comparing the usability of one-way and multi-way constraint-based alignment and distribution tools in diagram editors. The results of which provide strong support for our hypothesis that multi-way constraint-based placement tools are more usable than one-way constraint-based placement tools.

This experiment also investigated the impact of visual feedback provided during direct manipulation. We tested delayed feedback in which the position of objects connected by constraints to the selected objects being moved is not updated until the user has completed the action. We compared this with immediate feedback in which the user sees all changes to the diagram immediately during direct manipulation.

While immediate feedback provided some obvious benefits and was favoured by users, it unexpectedly appeared to slow users down. While this slowdown was not statistically significant, it is interesting since it conflicts with the general belief that immediate feedback is purely beneficial.

This experiment also highlighted two significant usability issues with our multi-way constraint-based placement tools. Firstly, constraint indicators could obscure the diagram and cause clutter. Secondly, complicated systems of constraints were frequently difficult for users to understand. To address these problems we again revised the tools, adding visibility controls for indicators as well as a constraint information mode.

A follow-up user study was conducted examining the usefulness of these additions. We found that the visibility controls, and specifically fading of indicators while they are inactive down to a minimum fade level, provided an effective solution for the clutter problem. The study showed information mode to be a beneficial addition, but not the complete answer to the issue of user comprehension.

While these studies were conducted specifically for constraint-based tools for alignment and distribution, we believe the findings extend to other similar interactive constraint-based tools.

7.4 Connector routing

In Chapter 5 we presented a fast incremental algorithm for maintaining a visibility graph for connector routing in interactive applications. We show how to compute object-avoiding poly-line connectors that update automatically as obstacles are added to the diagram. These connectors are automatically improved as better routes become available due to removal of shapes from the diagram. The algorithms are designed to support user interaction and offer a fast partial feedback mode that results in less distracting interactive dragging of shapes.

By default, the connector routes are predictably computed as the shortest path. However, the presented approach has a lot of flexibility, allowing various extensions that enable the user to control penalties for particular routing features, allowing more aesthetically pleasing routings to be produced. Two drawing modifications—shared path nudging and curved connector corners—can increase the comprehensibility of the generated routings.

The routing algorithm and extensions are implemented as the open-source software library `libavoid`, already integrated into the popular Inkscape vector graphics editor.

7.5 Interactive network layout

Chapter 6 presented the continuous network layout model for integrating automatic graph layout into diagramming tools. We demonstrated how separation constraints can be used to represent global stylistic constraints as well as manually specified placement constraints.

In the continuous network layout model the graph layout engine runs continuously during user interaction. This is a vastly different model of interaction for the user—the system constantly improves the layout in response to structural changes or manipulation

of diagram objects. Throughout this process the layout engine maintains the constraints that have been placed on the diagram. This saves the user from repeated actions that would otherwise be required to enforce drawing conventions or aesthetic preferences. They can instead delegate the responsibility for maintaining these relationships to the software.

The continuous network layout model is flexible, allowing external graph layout techniques to be used to provide the initial layouts and define a set of separation constraints. Edge straightening techniques can then be used to perform small beautifications on a layout while preserving its topology. The user is given additional freedom to improve the layout by manually making adjustments to the topology. These changes can then be preserved throughout further editing.

7.6 Dunnart constraint-based diagram editor

Throughout the previous chapters, we have seen all of the research ideas fully implemented in the constraint-based diagram editor, Dunnart. This provides working and usable examples of all the techniques presented in this thesis.

Dunnart is not just a prototype system, but is a complete, truly usable constraint-based diagram editor. It was used to generate over two-thirds of the diagrams in this thesis, and is already being used for various projects involving diagram layout. Through the opinions and experience of those who have worked with Dunnart, it has provided important feedback and evaluation of design of constraint-based layout and placement tools.

Dunnart is currently available as a free download,[1] and the complete source code will shortly be made freely available under an open source software license.

7.7 Future directions

While this thesis has answered some important questions, many more arise from the suggested use of new interfaces and interaction models for constraint-based diagramming. This section outlines some suggestions for future research in each of the areas explored by this thesis.

Placement constraints

For placement relationships, further research could explore the usefulness of automatic inference of constraints for setting up placement relationships. Automatic inference was frequently requested by study participants. Of course, the difficulty here is accurately generating relationships that match users' expectations.

Attention could be focused on exploring techniques for improving the user's understanding of complex systems of constraints, supplementing the information mode idea

[1] Dunnart, Michael Wybrow, `http://www.csse.monash.edu.au/~mwybrow/dunnart/`
This website also includes some video demonstrating placement tools, continuous network layout and other selected features of Dunnart described in this thesis.

or via a completely different method. In general, this is a difficult and long recognised problem that still requires considerable work.

The learnability of the placement tools could be tested via further usability experiments, to get a more accurate sense of how easily users can begin using them effectively.

Further usability testing could examine the interface for minimum separation relationships to verify that the separation indicators clearly convey the type of relationship and allow easy manipulation of it. While the interface for separation relationships is based on the successful interface for distributions, they were developed after the formal usability studies and have thus not received as much attention and feedback from users.

Currently, we only provide alignment, distribution and separation in either the x or y dimension. It would be worth investigating constraint solving techniques that would allow us to provide these tools in any direction. Several people who have viewed our placement tools have asked if we could provide alignment guidelines in other orientations.

Connector routing

Future research in the area of connector routing could improve clustered routing support in `libavoid`. A simple approach to this is to detect and penalise connector crossings with cluster boundaries as we do for connector crossings, using the same shared path crossing detection.

Usability evaluation of the connector tool in Dunnart could be carried out, focusing on questions of how users may want to influence the chosen connector path, such as via user-specifiable bend points. Also, developing a way for users to specify the initial routing for new connectors could be very useful when coupled with the topology preserving aspect of our continuous network layout. One approach to this would be to allow the user to thread connectors around the diagram like a piece of elastic, having them wind around obstacles, but allowing them to unwind when the user backtracks.

A major area for future research would be to create an equivalently flexible and incremental technique for routing of orthogonal object-avoiding connectors. Orthogonal connectors are important for certain types diagrams and tend to be available in popular diagram editors, automatic routing for such connectors has been frequently requested by users who have seen our automatic poly-line routing.

Continuous network layout

As we have seen, continuous network layout provides a powerful tool to aid in diagram layout, however its interaction model is quite unique and will not be familiar to many users. As a result, this layout model could benefit from detailed, formal usability evaluation in many areas. Experiments could test if the visualisations for stylistic constraints are adequate or require more on-screen indicators. Research could test if it is worth highlighting collisions of non-overlapping objects during interaction, or testing a visualisation to indicate the interaction of downward pointing connectors. Experiments could look at other constraint comprehension and conflict visualisation questions, such determining an

ideal interface for debugging cycles of connectors. The value of animation techniques during layout updates could be examined, as could appropriate behaviour for placement tools while being ALT-dragged.

It would be beneficial to examine the use of continuous network layout for authoring various other diagram types. Also, the use of continuous network layout on examples from different domains may uncover still more useful types of constraints—user-specified or stylistic—that could be worth providing.

Finally, further work could investigate methods for working with large network diagrams. For example, the biological sciences domains have very large networks which they want to explore and sometimes arrange. Some of our constraint-based layout techniques could be useful during exploration or for arranging subsets of these large graphs.

7.8 Closing remarks

Diagrams are everywhere and increasing in use. They are usually authored manually by people using a diagram editor. Existing diagramming software provide tools for construction but very little support for layout. Thus, users are currently forced to tediously maintain placement relationships in diagrams by repeatedly using single-effect layout primitives.

This thesis has demonstrated the use of constraint-based techniques for providing placement tools, connector routing and automatic network layout for diagram editors. Each of these tools treats layout features as constraints that the user directly or indirectly specifies. The responsibility for maintaining these relationships is then delegated to the system. The tools are very flexible and allow the user to alter or override them at any time.

The constraint-based layout tools have been developed using user-centred design methodologies. They have all been fully implemented in the constraint-based diagram editor, Dunnart. This has allowed evaluation of the tools, with formal and informal feedback gathered and taken into account throughout the design process. These approaches have been found to offer users a more powerful and usable environment for authoring diagrams than has previously been available.

Though it is difficult to see in the still images on these pages, Dunnart and the techniques presented in this thesis offer a fundamentally different approach to diagram layout. This revolutionary interaction model uses constraints to continuously maintain desired aesthetic properties of the diagram. This saves the user a large amount of effort, while still giving them a great deal of flexibility to control the layout and the precise placement of all objects in the diagram. Constraints in the system are presented to the user through a high-level interface as placement tools and layout styles with familiar and predictable behaviour. This allows the constraint-base tools in Dunnart to be easily learned and used, even by those with little diagramming experience and no knowledge of constraints.

Vita

Publications arising from this thesis include:

Wybrow, M., Marriott, K., McIver, L. and Stuckey, P. J. (2003). The usefulness of constraints for diagram editing. In S. Viller and P. Wyeth (eds), *Proceedings of the 2003 Australasian Computer Human Interaction Conference (OZCHI),* Information Environments Program, University of Queensland, Queensland, Australia, pp. 192–201.

Wybrow, M., Marriott, K. and Stuckey, P. J. (2006). Incremental connector routing. In P. Healy and N. S. Nikolov (eds), *Proceedings of the 13th International Symposium on Graph Drawing (GD'05),* Vol. 3843 of Lecture Notes in Computer Science, Springer, pp. 446–457.

Dwyer, T., Marriott, K. and Wybrow, M. (2007). Integrating edge routing into force-directed layout. In M. Kaufmann and D. Wagner (eds), *Proceedings of the 14th International Symposium on Graph Drawing (GD'06),* Vol. 4372 of Lecture Notes in Computer Science, Springer, pp. 8–19.

Wybrow, M., Marriott, K., McIver, L. and Stuckey, P. J. (2008). Comparing usability of one-way and multi-way constraints for diagram editing. *ACM Transactions on Computer-Human Interaction (TOCHI)* 14(4): 19:1–38.

Dwyer, T., Marriott, K., Schreiber, F., Stuckey, P.J., Woodward, M., Wybrow, M. (2008). Exploration of networks using overview+detail with constraint-based cooperative layout. In *IEEE Transactions on Visualization and Computer Graphics (InfoVis 2008),* to appear.

Dwyer, T., Marriott, K., and Wybrow, M. (2008). Dunnart: A constraint-based network diagram authoring tool. In *Proceedings of the 16th International Symposium on Graph Drawing (GD'08),* submitted.

Dwyer, T., Marriott, K., and Wybrow, M. (2008). Topology Preserving constrained graph layout. In *Proceedings of the 16th International Symposium on Graph Drawing (GD'08),* submitted.

Permanent Address: Clayton School of Information Technology
Monash University
Australia

This thesis was typeset with LaTeX 2$_\varepsilon$[2] by the author.

[2] 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

Asano, T., Asano, T., Guibas, L., Hershberger, J. and Imai, H. (1986). Visibility of disjoint polygons, *Algorithmica* **1**(1): 49–63.

Badros, G. J. (2000). *Extending Interactive Graphical Applications with Constraints*, PhD thesis, Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350.

Bartels, R. H., Beatty, J. C. and Barsky, B. A. (1998). Bézier curves, *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*, Morgan Kaufmann, San Francisco, CA, USA, chapter 10, pp. 211–245.

Bastert, O. and Matuszewski, C. (2001). Layered drawings of digraphs, pp. 87–120.

Becker, M. Y. and Rojas, I. (2001). A graph layout algorithm for drawing metabolic pathways, *Bioinformatics* **17**(5): 461–467.

Bederson, B. B. and Hollan, J. D. (1994). Pad++: a zooming graphical interface for exploring alternate interface physics, *UIST '94: Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, New York, NY, USA, pp. 17–26.

Ben-Moshe, B., Hall-Holt, O., Katz, M. J. and Mitchell, J. S. B. (2004). Computing the visibility graph of points within a polygon, *SCG '04: Proceedings of the 20th Annual Symposium on Computational Geometry*, ACM Press, New York, NY, USA, pp. 27–35.

Bertault, F. (1999). A force-directed algorithm that preserves edge crossing properties, *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, Vol. 1731 of *Lecture Notes in Computer Science*, Springer, pp. 351–358.

Bertin, J. (1983). *Semiology of graphics*, University of Wisconsin Press. Translated by William J. Berg.

Bier, E. A. and Stone, M. C. (1986). Snap-dragging, *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, pp. 233–240.

Blackwell, A. F., Britton, C., Cox, A. L., Green, T. R. G., Gurr, C. A., Kadoda, G. F., Kutar, M., Loomes, M., Nehaniv, C. L., Petre, M., Roast, C., Roe, C., Wong, A. and

Young, R. M. (2001). Cognitive dimensions of notations: Design tools for cognitive technology, *CT '01: Proceedings of the 4th International Conference on Cognitive Technology*, Springer, London, UK, pp. 325–341.

Böhringer, K.-F. and Paulisch, F. N. (1990). Using constraints to achieve stability in automatic graph layout algorithms, *CHI'90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, pp. 43–51.

Bond, C. S. (2003). Topdraw: a sketchpad for protein structure topology cartoons, *Bioinformatics* **19**(2): 311–312.

Borning, A., Marriott, K., Stuckey, P. and Xiao, Y. (1997). Solving linear arithmetic constraints for user interface applications, *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 87–96.

Brandes, U. and Wagner, D. (1997). A bayesian paradigm for dynamic graph layout, *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, Vol. 1353 of *Lecture Notes in Computer Science*, Springer, pp. 236–247.

Branke, J. (2001). Dynamic graph drawing, *in* M. Kaufmann and D. Wagner (eds), *Drawing Graphs: Methods and Models*, Vol. 2025 of *Lecture Notes in Computer Science*, Springer, London, UK, chapter 9, pp. 228–246.

Brennan, R. J. (1975). An algorithm for automatic line routing on schematic drawings, *Proceedings of the 12th Design Automation Conference*, pp. 324–330.

Bridgeman, S. S., Fanto, J., Garg, A., Tamassia, R. and Vismara, L. (1997). InteractiveGiotto: An algorithm for interactive orthogonal graph drawing, *GD '97: Proceedings of the 5th International Symposium on Graph Drawing*, Vol. 1353 of *Lecture Notes in Computer Science*, Springer, pp. 303–308.

Card, S. K., Mackinlay, J. D. and Shneiderman, B. (1999). Information visualization, *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1–34.

Chandler, P. and Sweller, J. (1991). Cognitive load theory and the format of instruction, *Cognition and Instruction* **8**(4): 293–332.

Chang, B.-W. and Ungar, D. (1993). Animation: from cartoons to the user interface, *UIST '93: Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, New York, NY, USA, pp. 45–55.

Chok, S. S. and Marriott, K. (1998). Automatic construction of intelligent diagram editors, *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 185–194.

Davidson, R. and Harel, D. (1996). Drawing graphs nicely using simulated annealing, *ACM Transactions on Graphics* **15**(4): 301–331.

Di Battista, G., Eades, P., Tamassia, R. and Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Inc.

Diehl, S. and Görg, C. (2002). Graphs, they are changing—dynamic graph drawing for a sequence of graphs, *in* M. T. Goodrich and S. G. Kobourov (eds), *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, Vol. 2528 of *Lecture Notes in Computer Science*, Springer, pp. 23–30.

Dijkstra, E. W. (1959). A note on two problems in connection with graphs, *Numerische Mathematik* pp. 269–271.

Dix, A., Finlay, J., Abowd, G. D. and Beale, R. (2003). *Human-Computer Interaction (3rd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

do Nascimento, H. A. D. and Eades, P. (2002). User hints for directed graph drawing, *GD '01: Revised Papers from the 9th International Symposium on Graph Drawing*, Springer, London, UK, pp. 205–219.

do Nascimento, H. A. D. and Eades, P. (2005). User hints: a framework for interactive optimization, *Future Generation Computer Systems* **21**(7): 1177–1191.

Dwyer, T., Koren, Y. and Marriott, K. (2006). IPSep-CoLa: An incremental procedure for separation constraint layout of graphs, *IEEE Transactions on Visualization and Computer Graphics* **12**(5): 821–828.

Dwyer, T., Marriott, K. and Stuckey, P. (2006). Fast node overlap removal, *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, Vol. 3843 of *Lecture Notes in Computer Science*, Springer, pp. 153–164.

Dwyer, T., Marriott, K. and Wybrow, M. (2007). Integrating edge routing into force-directed layout, *in* M. Kaufmann and D. Wagner (eds), *Proceedings of the 14th International Symposium on Graph Drawing (GD'06)*, Vol. 4372 of *Lecture Notes in Computer Science*, Springer, pp. 8–19.

Eades, P. (1984). A heuristic for graph drawing, *Congressus Numerantium* **42**: 149–160.

Eades, P., Lai, W., Misue, K. and Sugiyama, K. (1991). Preserving the mental map of a diagram, *Proceedings of Compugraphics '91*, pp. 34–343.

Elliott, P. R., Pei, X. Y., Dafforn, T. R. and Lomas, D. A. (2000). Topography of a 2.0 å structure of α_1-antitrypsin reveals targets for rational drug design to prevent conformational disease, *Protein Science* **9**: 1274–1281.

Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C. and Woodhull, G. (2003). Graphviz and Dynagraph—Static and dynamic graph drawing tools, *in* M. Jünger and P. Mutzel (eds), *Graph Drawing Software*, Springer, Berlin, pp. 127–148.

Fisk, C. J. and Isett, D. D. (1965). ACCEL: Automated Circuit Card Etching Layout, *DAC'65: Proceedings of the SHARE Design Automation Project*, ACM Press, pp. 9.1–9.31.

Fletcher, J. D. and Tobias, S. (2001). The multimedia principle, *in* R. E. Mayer (ed.), *Multimedia Learning*, Cambridge University Press, New York, NY, USA, chapter 7, pp. 117–134.

Flores, T. P., Moss, D. S. and Thornton, J. M. (1994). An algorithm for automatically generating protein topology cartoons, *Protein Engineering* **7**: 31–37.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34**(3): 596–615.

Freeman-Benson, B. N., Maloney, J. and Borning, A. (1990). An incremental constraint solver, *Communications of the ACM* **33**(1): 54–63.

Frick, A., Ludwig, A. and Mehldau, H. (1995). A fast adaptive layout algorithm for undirected graphs, *GD '94: Proceedings of the DIMACS International Workshop on Graph Drawing*, Springer-Verlag, London, UK, pp. 388–403.

Frishman, D. and Argos, P. (1995). Knowledge-based protein secondary structure assignment, *Proteins: Structure, Function, and Genetics* **23**: 566–579.

Fruchterman, T. M. J. and Reingold, E. M. (1991). Graph drawing by force-directed placement, *Software—Practice and Experience* **21**(11): 1129–1164.

Fudos, I. (1995). *Geometric Constraint Solving*, PhD thesis, Purdue University, Department of Computer Sciences.

Gansner, E., Koren, Y. and North, S. (2004). Graph drawing by stress majorization, *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, Vol. 3383 of *Lecture Notes in Computer Science*, Springer, pp. 239–250.

Ghosh, S. K. and Mount, D. M. (1991). An output-sensitive algorithm for computing visibility, *SIAM Journal on Computing* **20**(5): 888–910.

Gleicher, M. (1992). Integrating constraints and direct manipulation, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, ACM Press, pp. 171–174.

Gleicher, M. (1993). A graphics toolkit based on differential constraints, *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 109–120.

Gleicher, M. and Witkin, A. (1994). Drawing with constraints, *The Visual Computer: International Journal of Computer Graphics* **11**(1): 39–51.

Green, T. R. G. (1989). Cognitive dimensions of notations, *in* A. Sutcliffe and L. Macaulay (eds), *Proceedings of the 5th Conference of the British Computer Society, Human-Computer Interaction Specialist Group - People and Computers V*, Cambridge University Press, New York, NY, USA, pp. 443–460.

Green, T. R. G. (1991). Describing information artifacts with cognitive dimensions and structure maps, *in* D. Diaper and N. Hammond (eds), *Proceedings of the 6th Conference of the British Computer Society, Human-Computer Interaction Specialist Group—People and Computers VI—Usability Now!*, Cambridge University Press, New York, NY, USA, pp. 297–315.

Green, T. R. G. and Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework, *Journal of Visual Languages and Computing* **7**(2): 131–174.

Harris, R. L. (1999). *Information Graphics: A Comprehensive Illustrated Reference*, Oxford University Press, Inc., New York, NY, USA.

Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems Science and Cybernetics (SSC-4)* **2**: 100–107.

He, W. and Marriott, K. (1998). Constrained graph layout, *Constraints* **3**(4): 289–314.

Heydon, A. and Nelson, G. (1994). The Juno-2 constraint-based drawing editor, *Technical Report 131a*, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.
`http://ftp.digital.com/pub/Digital/SRC/research-reports/SRC-131a.ps.gz`

Heyns, W., Sansen, W. and Beke, H. (1980). A line-expansion algorithm for the general routing problem with a guaranteed solution, *Proceedings of the 17th Design Automation Conference*, pp. 243–249.

Hill, R. D. (1993). The Rendezvous constraint maintenance system, *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 225–234.

Horn, R. E. (1999a). Information design: Emergence of a new profession, *in* R. Jacobson (ed.), *Information Design*, The MIT Press, chapter 2, pp. 15–33.

Horn, R. E. (1999b). *Visual Language: Global Communication for the 21st Century*, MacroVU, Inc., Bainbridge Island, WA, USA.

Horton, W. (1991). *Illustrating Computer Documentation: the Art of Presenting Information Graphically on Paper and Online*, John Wiley & Sons, Inc., New York, NY, USA.

Hower, W. and Graf, W. H. (1996). A bibliographical survey of constraint-based approaches to CAD, graphics, layout, visualization, and related topics, *Knowledge-Based Systems* **9**: 449–464.

Hurst, N., Marriott, K. and Moulder, P. (2003). Cobweb: a COnstraint-Based WEB browser, *ACSC '03: Proceedings of the 26th Australasian Conference on Computer Science*, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 247–254.

Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H. (1997). Interactive beautification: a technique for rapid geometric design, *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 105–114.

Jacobs, C., Li, W., Schrier, E., Bargeron, D. and Salesin, D. (2004). Adaptive document layout, *Communications of the ACM* **47**(8): 60–66.

Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features, *Biopolymers* **22**: 2577–2637.

Kamada, T. (1989). *Visualizing Abstract Objects and Relations*, World Scientific Publishing Co., Inc., River Edge, NJ, USA.

Kamada, T. and Kawai, S. (1989). An algorithm for drawing general undirected graphs, *Information Processing Letters* **31**: 7–15.

Kanehisa, M., Goto, S., Hattori, M., Aoki-Kinoshita, K. F., Itoh, M., Kawashima, S., Katayama, T., Araki, M. and Hirakawa, M. (2006). From genomics to chemical genomics: new developments in KEGG, *Nucleic Acids Research* **34**: D354–357.

Karp, P. D. and Paley, S. M. (1994). Automated drawing of metabolic pathways, *in* H. Lim, C. Cantor and R. Bobbins (eds), *Proceedings of the International Conference on Bioinformatics and Genome Research*, pp. 225–238.

Karp, R. M. (1972). Reducibility among combinatorial problems, *in* R. Miller and J. Thatcher (eds), *Complexity of Computer Computations*, Plenum Press, pp. 85–103.

Knuth, D. E. (1979). *TEX and METAFONT*, Digital Press.

Kramer, G. (1992). A geometric constraint engine, *Artificial Intelligence* **58**(1–3): 327–360.

Kruskal, J. B. and Seery, J. B. (1980). Designing network diagrams, *Proceedings of the 1st General Conference on Social Graphics*, U.S. Department of the Census, pp. 22–50.

Kurlander, D. and Feiner, S. (1993). Inferring constraints from multiple snapshots, *ACM Transactions on Graphics (TOG)* **12**(4): 277–304.

Lasseter, J. (1987). Principles of traditional animation applied to 3d computer animation, *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, pp. 35–44.

Lee, C. Y. (1961). An algorithm for path connections and its applications, *IRE Transactions on Electronic Computers* **EC-10**(2): 346–365.

Lee, D.-T. (1978). *Proximity and reachability in the plane.*, PhD thesis, Department of Electrical Engineering, University of Illinois, Urbana, IL.

Marks, J. and Reiter, E. (1990). Avoiding unwanted conversational implicatures in text and graphics, *AAAI '90: Proceedings of the 8th National Conference on Artificial Intelligence*, The MIT Press, pp. 450–456.

Marriott, K. and Chok, S. S. (2002). QOCA: A constraint solving toolkit for interactive graphical applications, *Constraints* **7**(3-4): 229–254.

Mayer, R. E. (2001a). Principles for reducing extraneous processing in multimedia learning: Coherence, signaling, redundancy, spatial contiguity, and temporal contiguity principles, in *Multimedia Learning* (Mayer, 2001b), chapter 12, pp. 183–200.

Mayer, R. E. (ed.) (2001b). *Multimedia Learning*, Cambridge University Press, New York, NY, USA.

McCormack, C., Marriott, K. and Meyer, B. (2004). Adaptive layout using one-way constraints in SVG, *Proceedings of the 3rd Annual Conference on Scalable Vector Graphics (SVG Open)*.
`http://www.svgopen.org/2004/papers/ConstraintSVG`

McDonald, J. A., Stuetzle, W. and Buja, A. (1990). Painting multiple views of complex objects, *Proceedings of the 1990 ACM Conference on Object-Oriented Programming: Systems, Languages, and Applications and the European Conference on Object-Oriented Programming*, Ottawa, Canada, pp. 245–257.

Mehlhorn, K. (1984). *Graph algorithms and NP-completeness*, Springer, New York, NY, USA.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D. and Alon, U. (2002). Network motifs: Simple building blocks of complex networks, *Science* **298**(5594): 824–827.

Miriyala, K., Hornick, S. W. and Tamassia, R. (1993). An incremental approach to aesthetic graph layout, *Proceedings of the 6th International Workshop on Computer-Aided Software Engineering*, IEEE Computer Society, pp. 297–308.

Misue, K., Eades, P., Lai, W. and Sugiyama, K. (1995). Layout adjustment and the mental map, *Journal of Visual Languages and Computing* **6**(2): 183–210.

Mitchell, J. S. B. (2000). Geometric shortest paths and network optimization, *in* J.-R. Sack and J. Urrutia (eds), *Handbook of Computational Geometry*, Elsevier Science Publishers B.V., Amsterdam, pp. 633–701.

Myers, B. A., Giuse, D. A., Dannenberg, R. B., Vander Zanden, B., Kosbie, D. S., Pervin, E., Mickish, A. and Marchal, P. (1990). Garnet: Comprehensive support for graphical, highly-interactive user interfaces, *IEEE Computer* **23**(11): 71–85.

Myers, B. A. and Kosbie, D. S. (1996). Reusable hierarchical command objects, *CHI '96: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, pp. 260–267.

Myers, B. A., McDaniel, R. G., Miller, R. C., Ferrency, A. S., Faulring, A., Kyle, B. D., Mickish, A., Klimovitski, A. and Doane, P. (1997). The Amulet environment: New models for effective user interface software development, *IEEE Transactions on Software Engineering* **23**(6): 347–365.

Nelson, G. (1985). Juno, a constraint-based graphics system, *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, pp. 235–243.

Nelson, R. C. and Samet, H. (1986). A consistent hierarchical representation for vector data, *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, pp. 197–206.

Oberlander, J. (1996). Grice for graphics: Pragmatic implicature in network diagrams, *Information Design Journal* **8**(2): 163–179.

Overmars, M. H. and Welzl, E. (1988). New methods for computing visibility graphs, *Proceedings of the 4th Annual Symposium on Computational Geometry*, ACM Press, pp. 164–171.

Perlin, K. and Fox, D. (1993). Pad: an alternative approach to the computer interface, *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, pp. 57–64.

Petre, M. (1995). Why looking isn't always seeing: readership skills and graphical programming, *Communications of the ACM* **38**(6): 33–44.

Pettifer, S. R., Sinnott, J. R. and Attwood, T. K. (2004). UTOPIA: User friendly tools for operating informatics applications, *Comparative and Functional Genomics* **5**: 56–60.

Purchase, H. C., Allder, J.-A. and Carrington, D. (2002). Graph layout aesthetics in UML diagrams: User preferences, *Journal of Graph Algorithms and Applications* **6**(3): 255–279.

Raisamo, R. and Räihä, K.-J. (1996). A new direct manipulation technique for aligning objects in drawing programs, *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 157–164.

Robbins, J., Kantor, M. and Redmiles, D. (1999). Sweeping away disorder with the broom alignment tool, *CHI '99 extended abstracts on Human Factors in Computing Systems*, ACM Press, pp. 250–251.

Rohnert, H. (1986). Shortest paths in the plane with convex polygonal obstacles, *Information Processing Letters* **23**(2): 71–76.

Ryall, K., Marks, J. and Shieber, S. (1997). An interactive constraint-based system for drawing graphs, *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 97–104.

Sannella, M., Maloney, J., Freeman-Benson, B. N. and Borning, A. (1993). Multi-way versus one-way constraints in user interfaces: Experience with the DeltaBlue algorithm, *Software—Practice and Experience* **23**(5): 529–566.

Schreiber, F. (2001). *Visualisierung biochemischer Reaktionsnetze*, PhD thesis, Universität Passau.

Sengupta, S., Kimura, T. D. and Apte, A. (1994). An artist's studio: a metaphor for modularity and abstraction in a graphical diagramming environment, *Proceedings of the 1994 IEEE Symposium on Visual Languages*, IEEE Press, pp. 128–136.

Sharir, M. (1997). Algorithmic motion planning, *in* J. E. Goodman and J. O'Rourke (eds), *Handbook of Discrete and Computational Geometry*, CRC Press LLC, Boca Raton, FL, chapter 40, pp. 733–754.

Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages, *IEEE Computer* **16**(8): 57–69.

Simpson, K. (2006). Semi-supervised layout in interactive diagram editors, *Honours thesis*, Monash University, Clayton School of Information Technology, Australia. `http://www.csse.monash.edu.au/hons/se-projects/2006/Kieran.Simpson/`

Snodgrass, J. G., Levy-Berger, G. and Haydon, M. (1985). *Human Experimental Psychology*, Oxford University Press, New York.

Stahovich, T. F. (2001). Interpreting the engineer's sketch: A picture is worth a thousand constraints, *in* M. Anderson, B. Meyer and P. Olivier (eds), *Diagrammatic Representation and Reasoning*, Springer, Secaucus, NJ, USA, chapter 26, pp. 467–483.

Sugiyama, K. and Misue, K. (1995). Graph drawing by the magnetic spring model, *Journal of Visual Languages and Computing* **6**(3): 217–231.

Sugiyama, K., Tagawa, S. and Toda, M. (1981). Methods for visual understanding of hierarchical system structures, *IEEE Transactions on Systems, Man, and Cybernetics* **11**(2): 109–125.

Sutherland, I. E. (1963). *Sketchpad: A Man-Machine Graphical Communication System*, PhD thesis, Massachusetts Institute of Technology.

Tamassia, R., Battista, G. D. and Batini, C. (1988). Automatic graph drawing and readability of diagrams, *IEEE Transactions on Systems, Man, and Cybernetics* **18**(1): 61–79.

Tufte, E. R. (1983). *The visual display of quantitative information*, Graphics Press, Cheshire, CT, USA.

Van Wyk, C. J. (1982). A high-level language for specifying pictures, *ACM Transactions on Graphics (TOG)* **1**(2): 163–182.

Vander Zanden, B. (1996). An incremental algorithm for satisfying hierarchies of multi-way dataflow constraints, *ACM Transactions on Programming Languages and Systems* **18**(1): 30–72.

Vander Zanden, B. T., Halterman, R., Myers, B. A., McDaniel, R., Miller, R., Szekely, P., Giuse, D. A. and Kosbie, D. (2001). Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **23**(6): 776–796.

Ware, C. and Bobrow, R. (2004). Motion to support rapid interactive queries on node-link diagrams, *ACM Transactions on Applied Perception* **1**(1): 3–18.

Ware, C., Purchase, H., Colpoys, L. and McGill, M. (2002). Cognitive measurements of graph aesthetics, *Information Visualization* **1**(2): 103–110.

Weitzman, L. and Wittenburg, K. (1994). Automatic presentation of multimedia documents using relational grammars, *MULTIMEDIA '94: Proceedings of the 2nd ACM International Conference on Multimedia*, ACM Press, pp. 443–451.

Welzl, E. (1985). Constructing the visibility graph for n line segments in $O(n^2)$ time, *Information Processing Letters* **20**(4): 167–171.

Winn, W. (1990). Encoding and retrieval of information in maps and diagrams, *Transactions on Professional Communication* **33**(3): 103–107.

Woodberry, O. J. (2003). Knowledge engineering a Bayesian network for an ecological risk assessment, *Honours thesis*, Monash University, School of Computer Science and Software Engineering, Australia.
`http://www.csse.monash.edu.au/hons/projects/2003/Owen.Woodberry/`

Woodward, M. (2007). Adaptive layout of UML diagrams, *Honours thesis*, University of Melbourne, Department of Computer Science and Software Engineering, Australia.