High-Quality Ultra-Compact Grid Layout of Grouped Networks

Vahan Yoghourdjian, Tim Dwyer, Graeme Gange, Steve Kieffer, Karsten Klein, and Kim Marriott

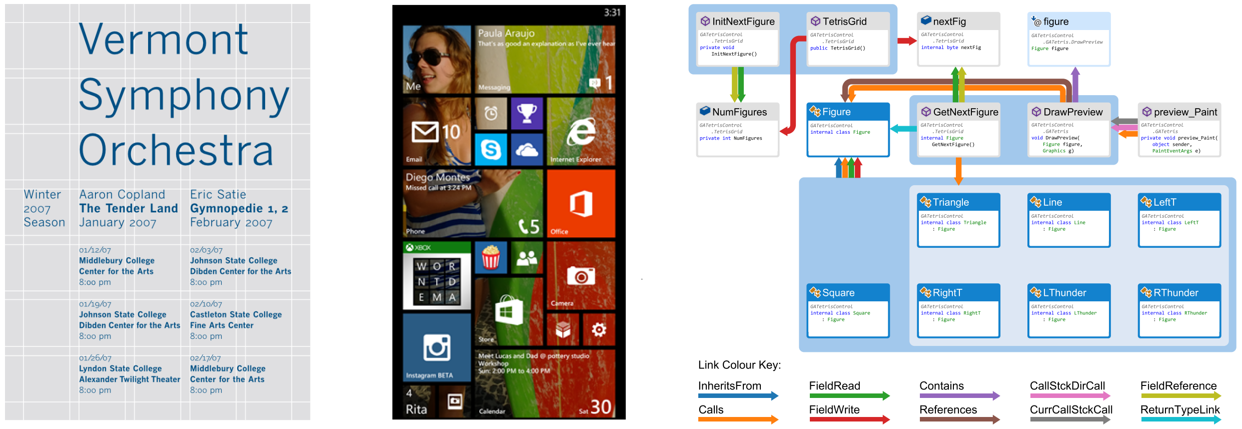


Fig. 1: Grid systems in typographic layout, UI design and an example of our proposed grid layout for a power-graph. With graphic designers playing an increasing role in the design of user interfaces for phone, tablet and desktop operating systems, this traditional grid-based design aesthetic is becoming more popular in these media. A case in point is Microsoft's "Modern" interface which seeks to unify app-design across devices. This resurgence of the grid-design aesthetic in new media leads us to re-examine some of the aesthetic assumptions that have been made in designing layout methods for network diagrams.

Abstract— Prior research into network layout has focused on fast heuristic techniques for layout of large networks, or complex multi-stage pipelines for higher quality layout of small graphs. Improvements to these pipeline techniques, especially for orthogonal-style layout, are difficult and practical results have been slight in recent years. Yet, as discussed in this paper, there remain significant issues in the quality of the layouts produced by these techniques, even for quite small networks. This is especially true when layout with additional grouping constraints is required. The first contribution of this paper is to investigate an ultra-compact, grid-like network layout aesthetic that is motivated by the grid arrangements that are used almost universally by designers in typographical layout. Since the time when these heuristic and pipeline-based graph-layout methods were conceived, generic technologies (MIP, CP and SAT) for solving combinatorial and mixed-integer optimization problems have improved massively. The second contribution of this paper is to reassess whether these techniques can be used for high-quality layout of small graphs. While they are fast enough for graphs of up to 50 nodes we found these methods do not scale up. Our third contribution is a *large-neighborhood search* meta-heuristic approach that is scalable to larger networks.

Index Terms—Network visualization, graph drawing, power graph, optimization, large-neighborhood search

1 Introduction

Computer science researchers (and others) have been exploring different ways to automatically layout and draw diagrams that represent graphs or networks for many decades. Because of the difficulty of the network layout problem and limited computational power of early computers, the primary focus was on developing fast heuristic techniques with low time complexity. As computer power rapidly increased through the '90s and 2000s, many researchers continued to focus on fast heuristic techniques in a race to see who could untangle the biggest graphs.

Other research led to the development of complex multi-stage layout frameworks for higher quality layout of smaller networks. For example, a seminal paper by Batini *et al.* [11] proposed a multi-stage layout framework called *Topology-Shape-Metrics* (TSM) which led to the development of *orthogonal* graph-drawing techniques which were designed to produce drawings with orthogonal connectors. Another family of multi-stage approaches arose following Sugiyama *et al.* [47], specifically for layered-layout of directed graphs.

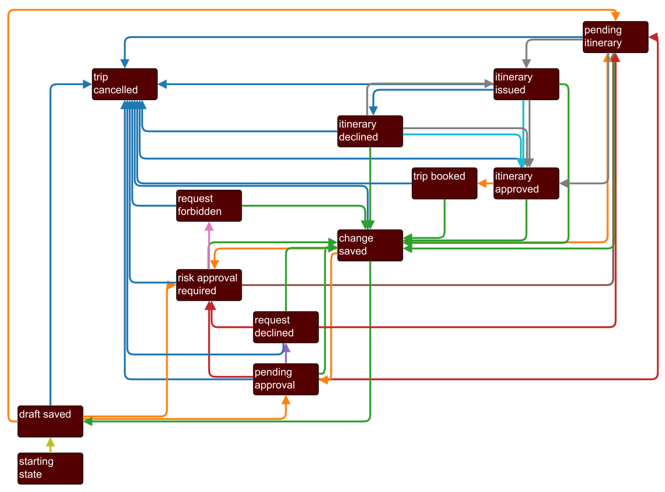
One feature common to the layouts produced by all of these different algorithms for network layout is the use of white space to clearly separate nodes and an implicit visual emphasis on edges rather than nodes. This leads to relatively sparse layouts in which most of the display space is empty. The main contribution of this paper is to investigate a new network layout aesthetic based on ultra-compact grid layout.

This new aesthetic is motivated by the grid arrangements that are used almost universally by designers in typographical layout, and are increasingly common in other media such as computer interfaces (Fig. 1). Layout in this tradition is built upon a grid that divides the viewing space into regular cells. Individual elements may span grid-cells but—as much as possible—the grid subdivisions are respected. This approach provides a regularity to the layout that leads the eye in a familiar and comfortable way [37]. Recent studies of network layout have shown that grid arrangements are memorable [35], when people arrange small diagrams themselves they prefer to place nodes at grid-points [42] and such placements are also preferred to TSM-based orthogonal layout [29].

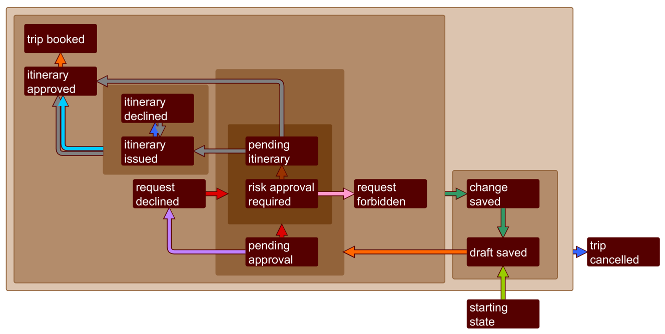
As we have mentioned, TSM-based orthogonal layout techniques were also influenced by a grid-aesthetic, though probably more due to

- *Vahan Yoghourdjian, Tim Dwyer, Steve Kieffer, Karsten Klein, and Kim Marriott are with Monash University. E-mail: {Vahan.Yoghourdjian, Tim.Dwyer, Steve.Kieffer, Karsten.Klein, Kim.Marriott}@monash.edu.*
- *Graeme Gange is with The University of Melbourne. E-mail: GkGange@gmail.com.*

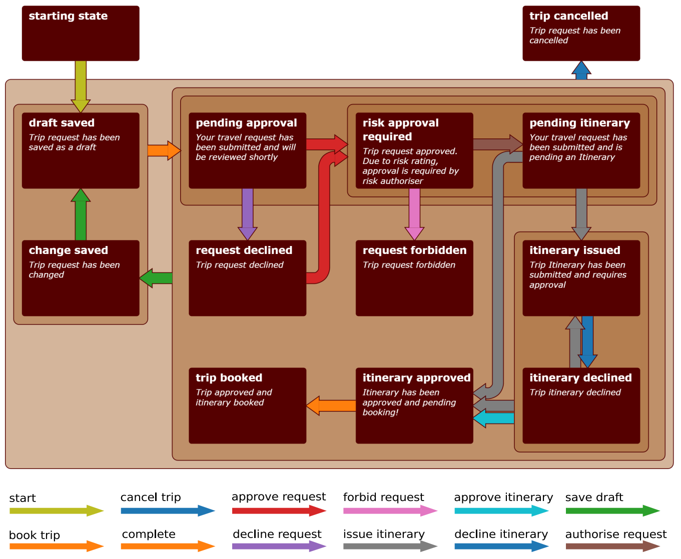
Manuscript received 31 Mar. 2015; accepted 1 Aug. 2015; date of publication xx Aug. 2015; date of current version 25 Oct. 2015. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.



(a) A traditional TSM-based orthogonal layout of the flat network.



(b) We can use power-graph like compartments to reduce the number of edges. A lot more structure is visible in this edge-compressed version of the graph from 2(a). For example, the outermost compartment makes it obvious that all states apart from "starting state" are cancellable, i.e. have a link to "trip cancelled". However, this TSM-based orthogonal layout still takes quite a large area. If coerced to a grid the dimensions would be 6×7, leaving 29 empty grid-cells.


starting state
trip cancelled
Trip request has been cancelled
draft saved
Trip request has been saved as a draft
pending approval
Your travel request has been submitted and will be reviewed shortly
risk approval required
Trip request approved. Due to risk rating, approval is required by risk authoriser
pending itinerary
Your travel request has been submitted and is pending an Itinerary
change saved
Trip request has been changed
request declined
Trip request declined
request forbidden
Trip request forbidden
itinerary issued
Trip Itinerary has been submitted and requires approval
trip booked
Trip approved and itinerary booked
itinerary approved
Itinerary has been approved and pending booking!
itinerary declined
Trip itinerary declined
start
cancel trip
approve request
forbid request
approve itinerary
save draft
book trip
complete
decline request
issue itinerary
decline itinerary
authorise request

(c) Here is the same state-machine shown using our ultra-compact grid-based layout which has grid dimensions 4×4 leaving only three empty grid-cells. This optimally compact solution was found in 0.464 seconds using the SAT solver. Although we do not explicitly minimise bends or crossings, our layout is equal to the TSM output in these respects and significantly reduces the overall area and edge-length. With the additional node area we are able to include more detailed descriptions of each state.

Fig. 2: Drawings of the state machine for a travel-booking system.

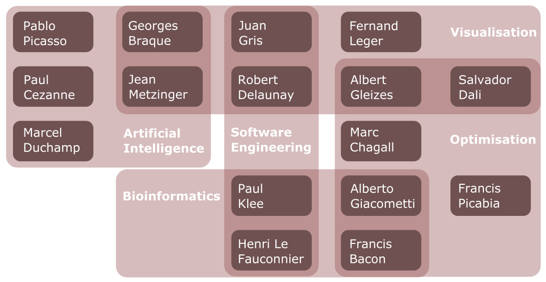


Fig. 3: An Euler diagram showing the intersecting sets of interests of researchers in a lab (anonymised). Solved in 0.047 seconds using the SAT solver.

circuit layout traditions than typography. As Fig. 2 shows, our new aesthetic leads to a very different visual style. It uses ultra-compact arrangement on a grid with group membership shown by containment in nested rectangular regions. This containment then enables the use of the power-graph convention [45] to collapse edges. In such an *edge-compressed* view [17] an edge from a group to another group implies all nodes in the first group are connected to all nodes in the second. Unlike virtually all existing approaches to power-graphs or network diagrams with clustering, we do not require that groups are hierarchical, making our approach applicable to other types of diagrams such as representations of overlapping set-membership (Fig. 3).

After exploring grid-design, this paper's second contribution is to investigate practical methods for producing ultra-compact grid layout that is of the *highest-possible quality*. Rather than developing a specialised algorithm we decided to explore more general purpose optimisation based approaches. One reason for this is that current frameworks for high-quality network layout separate the layout into a pipeline of different steps and so, the resulting layouts are often compromised because of the fixed trade-off between aesthetic criteria imposed by the pipeline. Furthermore the implementations of these methods are complex and brittle, as discussed in Sect. 6.

Since the time when pipe-line-based graph-layout methods such as TSM were conceived, generic technologies for solving combinatorial and mixed-integer optimization problems have improved by several orders of magnitude. Simultaneously, the computing power available on average desktop machines has increased exponentially. Optimisation problems that once took weeks to solve with home-sized computers can now be solved in seconds on cheap computers in the home.

We feel then, that the time is right to reassess network layout and see whether these general purpose optimization techniques can be usefully applied to solve simple mathematical models encoding such layout problems. One advantage of using such a generic approach is that it allows us to readily explore this new aesthetic by allowing us to rapidly create examples from different applications and with different aesthetic trade-offs (Sect. 2).

Thus, after developing a model for ultra-compact grid layout (Sect. 3), we compare the applicability of several generic optimisation techniques (MIP, CP and SAT) to this problem (Sect. 4). While useful for exploring the design of the layout model, we found that even the best of these solving technique was only practical (in terms of running time) for graphs of up to around 50 nodes.

A common fallback for solving difficult combinatorial optimisation problems is to use generic meta-heuristic techniques like tabu search, simulated annealing or genetic programming. Dozens of different techniques have been proposed. While not guaranteed to find an optimal solution they are routinely used to find "good" solutions to problems that are too hard to solve optimally using MIP, CP or SAT. We therefore developed a meta-heuristic to solve our layout problem.

We decided to use *large neighbourhood search* (LNS) [8, 41](Sect. 5). This class of meta-heuristic is currently *de rigueur* for solving various transportation and scheduling problems. While we are not the first to try generic meta-heuristic approaches for network layout we are the first to consider LNS. Though not guaranteed to find an optimally compact solution our evaluation shows that our LNS heuristic found reasonably good layouts (when compared to the optimal layout) and scaled to graphs with 100 nodes.

In summary, the technical contributions of this paper are to:

- Introduce a new ultra-compact grid-based aesthetic for network layout and explore the design-space (Section 2);
- Present a declarative model of layout goals and constraints that allows us to rapidly evaluate different refinements and applications of these aesthetic criteria, solvable using generic constrained optimisation techniques and without the need for specialised algorithm development (Sect. 3);
- Compare the efficiency of different generic optimisation techniques (MIP, CP, SAT) for solving our declarative model (Sect. 4);
- Explore the use of a *large-neighborhood-search* based meta-heuristic to solve this declarative model. This allowed us to obtain compact grid layouts for graphs of up to 100 nodes in less than 5 minutes (Sect. 5).

2 Ultra-compact Grid Layout

In this section we present a new layout aesthetic for network diagrams that is based on grid layout in typography and we provide a number of motivating examples. The aesthetic incorporates the following layout requirements:

R1 – Node content emphasis. Many applications have more than just simple labels associated with nodes, for example, rich graphics or text in paragraph or tabular form. In typography, grid-cells are packed quite densely in order to maximize the area devoted to this content. By contrast, orthogonal network diagram layouts are typically very sparse, devoting more space for edge paths, which—in order to minimize bends and crossings—may be very long. Networks featured in Figs. 1, 4, 5(b) and 6 all contain significant text and graphic content associated with each of the nodes. For this detail to remain readable at reasonable scales without resorting to interactive focus-and-context techniques (e.g. [27]), *compact node-placement is essential*. A strong correlation between human preference and layout compactness is also observed in a recent study by Kieffer *et al.* [29].

R2 – Proximity implies connectivity. If we are to devote less space to edge paths in our grid arrangements then we must rely more on the proximity of nodes to indicate connectivity. Recent studies have shown that layout that achieves such proximity is strongly preferred by readers of small diagrams [19]. In addition, this objective *indirectly* addresses crossings simply because shorter edges are less likely to cross. Minimizing edge-length can sometimes be a more successful strategy for minimizing crossings than heuristics which directly address crossings, e.g. [18].

R3 – Variable node dimensions. Some nodes may have significantly more content than others. Following typographical layout conventions, these nodes can be expanded to fit their content, but they must always fully fill a rectangular set of grid-cells. Furthermore, where different orientations of the node are possible (e.g. picture beside text or picture below text) the layout should choose the orientation to best suit the layout. Fig. 4 demonstrates layout with variable node orientations.

R4 – Containment. The semantics of many applications involve representing group membership over sets of nodes. In typography, such relationships are shown through nested rectangular enclosing regions.

R5 – Flow. In applications where the directionality is important we would like flow to be shown in multiple directions, for example, left-to-right *and* top-to-bottom, as in document layout.

Part of the motivation for this work was the search for an effective layout method for edge-compressed dense, directed networks [17]. Without compression, graphs that have only few nodes but many edges are already very difficult to read. For example, Fig. 2(a) shows the state-chart for a travel booking system with 13 nodes and 44 edges arranged using the commercial layout software, yFiles [7].

Figs. 2(b) and 2(c) show the edge-compressed version of the network with only 17 edges. In the compressed representation, an edge between two groups implies a *biclique*. That is, every node contained in the source group of the edge is the source of an edge to every node in the target group. Thus, precisely the same connectivity structure is conveyed but in a less cluttered way. Further, the grouping inferred by the edge-compression reveals structure, for example, it is obvious from the single edge adjacent to the largest group and the *trip cancelled* state that every state other than *start* is cancellable.

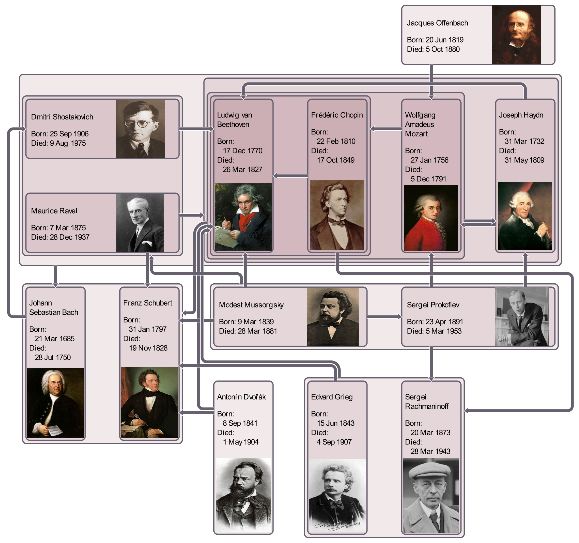


Fig. 4: Links between major composers arranged with our model with the solver choosing the best orientations for nodes. Layout took 37.422 seconds using the SAT solver - disjunctions due to variable node orientations expand the search space.

Figs. 2(b) and 2(c) compare layouts obtained by a standard TSM approach (yFiles) and by our model. Our layout model here keeps connected nodes close together (R2) while preserving group containment within rectangular regions (R4). Furthermore, the layout is as compact as possible while respecting node and group containments, thus maximising space for, and hence readability of node labels (R1). When node area is maximised in this way, we are able to include additional explanatory content for each node and the diagram becomes a more complete, stand-alone description of the state-machine.

Fig. 4 demonstrates the possibility to provide a very compact layout for a graph with nodes that require more than a single grid-cell to fit their content (R3). The network is a section of the “Composers Graph" that was one of the challenges for the 2014 Graph Drawing Conference contest [1]. Each node is a composer for whom we want to show both biographical details and a portrait. We allow the textual biographical details to fill one grid-cell, while the portrait can go in an adjacent cell, either beside or below the text. The solver automatically chooses the orientation of each node that permits layout that is optimal with respect to the other layout requirements.

Fig. 3 is an Euler Diagram representing the research interests of members of our lab (anonymised). Set labels are also treated as nodes and laid out within the same grid system. Note that the containment (R4) is no longer a strict hierarchy yet our general layout model is still applicable. Drawing Euler diagrams under certain constraints such as convexity of the regions is not always possible. Actually realising the drawing in an aesthetic way is a further challenge. Both of these problems have seen a lot of interest from computer-scientists and mathematicians and sophisticated algorithms have been developed [43, 46]. Here, we have defined the layout for rectangular boxes with a relatively simple declarative model, and left both the problems of determining feasibility and (if possible) layout to the solver.

In Fig. 5 we compare two different arrangements of a biological pathway network. In such pathways the direction of the edges is often very important, for example, indicating the direction of a reaction. It is therefore a common convention to show flow in such diagrams from top-to-bottom or from left-to-right. Fig. 5(a) uses a standard Sugiyama style [47] layout obtained, again, with yFiles. This method assigns nodes to layers such that edges exclusively span layers. By contrast, Fig. 5(b) introduces a disjunction constraint which allows edges to flow *either* left-to-right or top-to-bottom.

Fig. 6(b) shows a software-dependency graph. This network shows dependencies between types, methods and properties in C$^\sharp$ code and was obtained in a debugging scenario using the Visual Studio *Code Map* tool. This layout neatly illustrates the cause of the bug: that *Square* is the only sub-class of *Figure* not created by the *GetNextFigure* method. Code snippets and icons on each of the nodes give added context, again illustrating the need for node content emphasis (R1).

3 Layout Model

In this section we present a high-level declarative model for placing grouped nodes in a grid layout that formulates the problem as a constrained optimisation problem[1].

3.1 Node-Placement Model

The high-level model for node-placement takes the following as input:
1. The set of leaf or *base* nodes $B = \{1,...,n_B\}$ and the set of *container* nodes $C = \{n_B + 1,..,n_C\}$ which contain groups of other nodes.
2. A fixed width w_u and height h_u for every base node $u \in B$. These are positive integers.
3. Every container node has a set of nodes that are contained inside it. These can be container or base nodes. This is specified by the Boolean matrix $con[u,v]$ which is true iff $v \in B \cup C$ is inside $u \in C$. The containment relationship need not be hierarchical.
4. The containment relationship gives rise to a non-overlap relationship between nodes. For convenience this is pre-computed and passed into the model. It is given by the symmetric Boolean matrix $disj[u,v]$ which is true if $u,v \in B \cup C$ should not overlap.
5. For each pair of nodes $u,v \in B \cup C$ there is a non-negative desired distance $dd[u,v]$ between them with a non-negative weight $ddw[u,v]$. The weight $ddw[u,v]$ is 0 if u is contained in v or vice versa.
6. A maximum grid size, g_x and g_y, both of which are positive integers big enough to ensure that they contain the optimal layout.

Neither *con* nor *disj* need to contain redundant constraints: for efficiency they should be minimal.

We experimented with different desired distances and weights. Following *stress*-based methods [25], we tried setting the desired distance between two nodes to the graph-theoretic-distance taking into account containment[2]. We also tried simply setting the desired distance and weight to the edge adjacency matrix. Observing similar results for both approaches, we opted for the latter.

Variables and constraints:
1. The core decision variable in our model is the position $(xs[u], ys[u])$ of the top-left corner of each base node $u \in B$. This must be a point on the grid: $xs[u] \in \{1,...,g_x\}$ and $ys[u] \in \{1,...,g_y\}$ where g_x and g_y give the size of the grid.
2. The position of the bottom-right corner of each base node is functionally dependent upon this: $\forall u \in B$, $xf[u] = xs[u] + w[u]$ and $yf[u] = ys[u] + h[u]$.
3. We require that the whole node fits on the grid: $\forall u \in B$, $xf[u] \le g_x$ and $yf[u] \le g_y$.
4. The position, width, and height of the container nodes are also functionally dependent on the position of the base nodes; as the containers are just the bounding box of their constituents, so $\forall u \in C, v \in B \cup C$:

$$\begin{aligned} xs[u] &= \min\{xs[v] \mid v \in B \cup C \wedge con[u,v]\} \\ xf[u] &= \max\{xf[v] \mid v \in B \cup C \wedge con[u,v]\} \\ w[u] &= xf[u] - xs[u] \end{aligned}$$

and similar in the y-dimension.
5. The following disjunction ensures that nodes do not overlap: $\forall u < v \in C$ s.t. $disj[u,v]$,

$$xf[u] \le xs[v] \vee xf[v] \le xs[u] \vee yf[u] \le ys[v] \vee yf[v] \le ys[u].$$

[1]The full model in MiniZinc is available under an open-source license [38]

[2]This is the length of the shortest path between the nodes in an extended graph where there is an edge between two nodes x,y in this extended graph if there is an edge in the original graph or if $con[x,y]$ or $con[y,x]$ holds.

Objective function to be minimised: $stress + \alpha cc + \beta oc$ where α and β are fixed weights and the functions *stress*, *cc* and *oc* measure different aesthetic criteria, as follows.

The *stress* term is the difference between the desired and actual distance between the nodes. Because we are using orthogonal connectors and grid layout we use Manhattan distance. We measure the distance between the closest points on the perimeter of the nodes rather than between the center of the nodes as this leads to considerably better layout in the case that the nodes are not squares. To compute this we use the functionally dependent variables:

$$dx[u,v] = \begin{cases} xs[v] - xf[u] + 1, & \text{if } xf[u] \le xs[v] \\ xs[u] - xf[v] + 1, & \text{if } xf[v] \le xs[u] \\ 0, & \text{otherwise.} \end{cases}$$

We define $dy[u,v]$ symmetrically. Now,

$$stress = \sum_{u,v \in B \cup C} ddw[u,v] \cdot |dx[u,v] + dy[u,v] - dd[u,v]|\,.$$

The other components of the objective function are designed to ensure $\forall u \in B \cup C$ that compartments are compact $cc = \sum_{u \in C} w[u] + h[u]$, that the entire layout is compact $oc \ge xf[u]$, and that it fits inside a rectangle with a given aspect ratio ar: $yf[u] \le ar \cdot oc$.

A great advantage of using a constrained optimisation approach is that it is straight-forward to add additional constraint encodings based on additional aesthetic criteria. Thus, for the example presented in Fig. 4 we add a constraint to dictate a fixed perimeter of size 2x1 for base nodes; placed either horizontally or vertically. In the flow layout in Fig. 5 each source node should be either above or to the left of the destination node — another disjunction. Note that the optimisation problem that we tackle with our model is NP-hard as can be shown by reduction from the rectangle packing problem [34].

Initially we tried to use a single constrained optimisation model for both node-placement and edge routing. This modelled each edge using a fixed number of horizontal and vertical segments (some of which could be 0 length) and including a penalty term for each pair of segments to penalize possible edge crossings. However, this proved too slow for any but very small networks and so is currently still not practical. We therefore developed a separate heuristic algorithm for edge routing given the positions of the nodes on a grid.

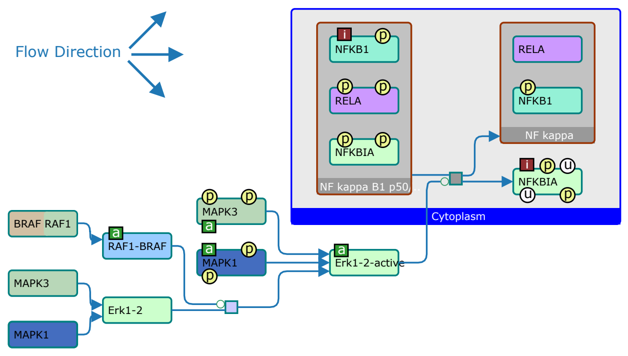
3.2 Routing

The grid-aesthetic naturally suggests routing connectors between base and container nodes in an orthogonal-style, i.e. with straight-line segments aligned to the grid. Obviously, intersections between these orthogonal edge paths and node boundaries (other than the source/target of the edge) should be avoided. Edge paths should only intersect container boundary rectangles if the container is the source or target, or is an ancestor of the source or target.

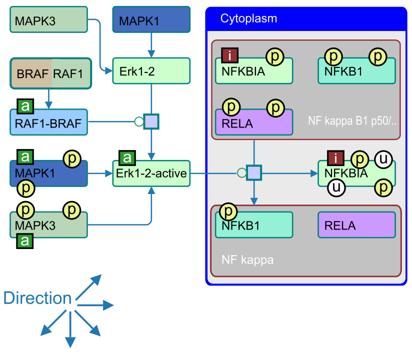
Currently the standard for orthogonal connector routing is to route over an orthogonal visibility graph, followed by a "nudging" phase to centre edge segments in channels between nodes [49]. Strict grid placement of nodes simplifies this problem considerably as we can route over the graph formed by the grid itself, intersected with the centre lines between each column and row of nodes, see Fig. 6. When we route an individual edge, we remove any edge segments which intersect nodes other than the start and end nodes, or segments intersecting containers which are not ancestors of the start and end nodes.

We also connect ports on the start and end nodes to each other. These port connections have zero cost in the subsequent shortest path finding problem between the start and end nodes. Thus, the shortest path will always run through the ports providing the shortest path between source and target. Otherwise, the cost of traversing each segment in the shortest path traversal (Dijkstra) is simply the length of that segment plus an additional penalty if traversing the edge would add a bend to the current path.

Finally, bundles of co-linear edge segments are constructed and an ordering within bundles that avoids unnecessary crossings is found



(a) Sugiyama style layered-layout with six layers and flow direction strictly left-to-right.



(b) Compact grid layout obtained with relaxed flow direction; solved in 2.035 seconds using the SAT solver.

Fig. 5: A directed biological pathway from `http://www.pathwaycommons.org`.

as suggested by Nöllenburg [39]. This ordering is used to generate constraints for a simple quadratic program subject to separation constraints (with solution as per [20]) to neatly space the edge segments in the available channels.

4 Optimal Node-Placement

The declarative model is a complete and precise mathematical formulation of the node-placement problem. In this section we evaluate three of the most widely-used generic techniques for solving such discrete constrained optimisation problems: Mixed-Integer Programming (MIP), SAT, and constraint programming (CP). These are all guaranteed to find an optimal solution to the problem. In this section we detail the exact encodings used as well as the experimental evaluation.

4.1 Constraint Programming

Subject to minor syntactic changes the model given in Sect. 3 is a MiniZinc [38] model. The actual MiniZinc is shown in Appendix A. Thus it can be directly executed and solved using any of the underlying solvers supported by MiniZinc.

For our evaluation we used a state-of-the-art constraint programming solver, G12 CPX [23] which utilises lazy clause generation. CPX like most constraint programming solvers provides global constraints for finding the minimum and maximum elements in a list or array, a predicate or function to compute the absolute value of a function and disjunctions of constraints. The calculation of distance between two nodes u, v in a given dimension was encoded as the expression

$$dx[u,v] = \max([0, xs[v] - xf[u] + 1, xs[u] - xf[v] + 1])$$

4.2 SAT

SAT solvers are designed to find values for Boolean variables that satisfy conjunctions of clauses, i.e. disjunctions of Boolean literals. When encoding an integer problem into SAT, each integer variable $x \in [1,\ldots,n]$ is usually encoded as a set of Boolean variables $[x^1,\ldots,x^n]$. There are two standard encodings for integer variables with small domains.

The *sparse* encoding requires that exactly one of the n variables is true; this gives the semantics

$$x^i \equiv [\![x = i]\!]$$

where this is read as the Boolean variable x^i in the encoded SAT model is true iff $x = i$ holds in the original integer programming model. The direct encoding of this semantic requires $O(n^2)$ clauses, since in addition to the set of disjunctions that ensure that at least one is true, it also requires an encoding that ensures that at most one value holds. Each pair of distinct values (i,j) requires a disjunction. There are $n(n-1)/2$ such pairs, resulting in $1+n(n-1)/2$ clauses.

The alternative *unary* encoding of integer variables instead ensures that the x^i are ordered; that is, $x^i \Rightarrow x^{i-1}$. These literals then have the semantics

$$x^i \equiv [\![x \geq i]\!].$$

In addition to requiring only $O(n)$ clauses, the unary encoding is convenient for encoding a range of arithmetic constraints.

Example 1 *Using the unary encoding, $x \leq y$ can be encoded as*

$$\bigwedge_i [\![x \geq i]\!] \to [\![y \geq i]\!] \equiv \bigwedge_i \neg x^i \vee y^i.$$

Example 2 *Using the unary encoding, $x = |y|$ can be encoded as*

$$\bigwedge_{i \geq 0} [\![x \geq i]\!] \leftrightarrow ([\![y \geq i]\!] \vee [\![y \leq -i]\!]) \equiv \bigwedge_{i \geq 0} x^i \leftrightarrow (y^i \vee \neg y^{1-i}).$$

Example 3 $x = \max(y_1,\ldots,y_n)$ *can be encoded as:*

$$\bigwedge_i ([\![x \geq i]\!] \leftrightarrow \bigvee_j [\![y_j \geq i]\!]) \equiv \bigwedge_i (x^i \leftrightarrow \bigvee_j y^i_j).$$

Because of these advantages we use the unary encoding. Linear arithmetic constraints, such as $x = \sum c_i y_i$ can be implemented using a range of encodings, such as BDDs, adders or cardinality networks [9, 15, 22].

Reified versions of these constraints can also be easily constructed. A *reified constraint* is of form $b \leftrightarrow C$ and constrains the Boolean b to be true iff the constraint C holds in the model. Reification is a standard technique used to encode disjunctions of constraints: the disjunction $C_1 \vee C_2$ is encoded as

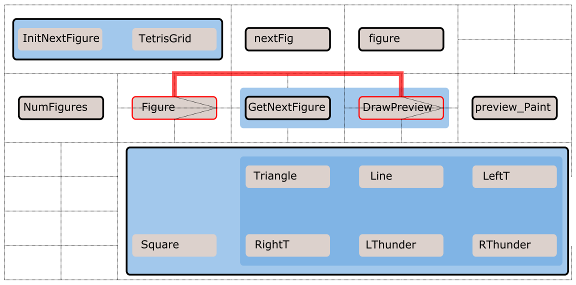
$$(b_1 \leftrightarrow C_1) \wedge (b_2 \leftrightarrow C_2) \wedge (b_1 \vee b_2).$$

Given these primitives, we can straightforwardly encode the model into SAT. For example, the non-overlap of nodes u and v becomes:

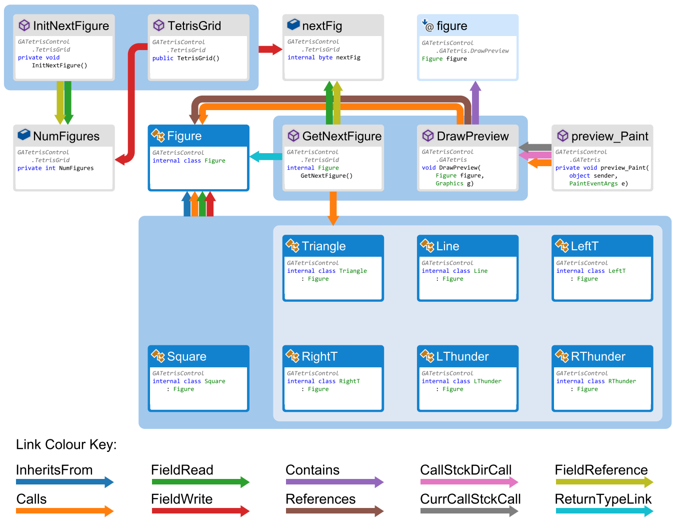
$$\left(\begin{array}{ll} & (b_{left} \vee b_{right} \vee b_{above} \vee b_{below}) \\ \wedge & b_{left} \leftrightarrow xf[u] \leq xs[v] \\ \wedge & b_{right} \leftrightarrow xf[v] \leq xs[u] \\ \wedge & b_{above} \leftrightarrow yf[u] \leq ys[v] \\ \wedge & b_{below} \leftrightarrow yf[v] \leq ys[u] \end{array}\right)$$

The encoding of the problem into SAT was performed using the *Ben-Gurion University Equi-Propagation Encoder* (BEE) [36], which compiles a declarative specification to SAT.

Primitive arithmetic constraints are encoded using direct unary adders; whereas, larger sums, such as the objective value, are encoded with *odd-even sorting networks*.



(a) Detail of routing graph. The example shows an edge being routed from "DrawPreview" to "Figure". Nodes with thick borders are obstacles to be avoided by the current path. Segments intersecting such nodes are removed from the routing graph. On the source and target nodes 0-cost port connections are visible.



(b) After routing, edge paths are bundled and separated within the available channel space.

Fig. 6: An example software-dependency graph with routing detail and the final result. Solved in 0.732 seconds with the SAT solver. This network shows dependencies between types, methods and properties in C$^\sharp$ code and was obtained in a debugging scenario using the Visual Studio *Code Map* tool. This layout neatly illustrates the cause of the bug: that *Square* is the only sub-class of *Figure* not created by the *GetNextFigure* method. Code snippets and icons on each of the nodes give added context.

The optimization is handled by solving a sequence of SAT instances. The solver initially solves the problem P and returns a solution $o = k$. Then it adds the constraint $(o < k)$ to the model and solves again. This is repeated until the resulting problem is found to be unsatisfiable. Hence the last solution found is optimal.

4.3 MIP

The MIP encoding is the most complex among the approaches we compare. We use the standard MIP encoding of minimum and maximum and absolute value [48]. We used six matrices of binary variables to keep track of the relative position of each pair of vertices u, v. The arrays $left[u,v]$, $xoverlap[u,v]$, $right[u,v]$ encode that u must be to the left, horizontally overlap, or must be to the right of v; and analogously in the y direction we have $below[u,v]$, $yoverlap[u,v]$, $above[u,v]$. The following constraint enforces the desired relationships in the x-direction, a similar constraint is used for the y-direction:

$\forall u < v \in B \cup C$

$$lt(xf[u], xs[v], left[u,v]) \wedge lt(xf[v], xs[u], right[u,v]) \wedge$$
$$lt(xs[u], xf[v], xoverlap[u,v]) \wedge lt(xs[v], xf[u], xoverlap[u,v])$$

where $lt(x_1, x_2, b)$ enforces that $b \rightarrow x_1 \leq x_2$ and has the standard MIP encoding $-M * (1-b) + x_1 \leq x_2$ where M is sufficiently large.

Using these it is simple to encode non-overlap and compute the distance between nodes:

1. The relative positions are mutually exclusive in each direction: $\forall u < v \in B \cup C$,
$$(left[u,v] + xoverlap[u,v] + right[u,v] = 1) \wedge$$
$$(above[u,v] + yoverlap[u,v] + below[u,v] = 1)$$
2. If there is a containment relationship between two nodes then they overlap in both directions: $\forall u < v \in B \cup C$ s.t. $con[u,v] \vee con[v,u]$ then $xoverlap[u,v] = 1 \wedge yoverlap[u,v] = 1$.
3. Enforce non-overlap: $\forall u < v \in B \cup C$ s.t. $disj[u,v]$,
$$left[u,v] + right[u,v] + above[u,v] + below[u,v] \geq 1$$
4. Compute x-distance: $\forall u < v \in B \cup C$ we have:
$$(dx[u,v] \geq 0.0) \wedge (dx[u,v] \geq xs[v] - xf[u] + 1) \wedge$$
$$(dx[u,v] \geq xs[u] - xf[v] + 1)$$
and:
$$lt(dx[u,v], 0.0, xoverlap[u,v]) \wedge$$
$$lt(dx[u,v], (xs[v] - xf[u] + 1), left[u,v]) \wedge$$
$$lt(dx[u,v], (xs[u] - xf[v] + 1), right[u,v])$$
and similarly for the y-direction.

We used MiniZinc to encode our MIP model and ran it using the state-of-the-art CPlex MIP solver [2] by using the flatzinc compatible CPlex interface.

4.4 Evaluation

In order to study the performance of our model encodings for CP, MIP and SAT on different graph characteristics we ran experiments on different types of input graphs. Experiments were run on a standard desktop machine with an Intel Core i7-4771 3.50GHz processor and 32GB RAM. Solvers were restricted to run on a single thread with a timeout of 300 seconds. The results were drawn using HTML5, javascript, D3.js [3] and Cola.js [6].

Our graph corpus consists of graphs from two sources, a set of randomly generated scale-free graphs, and a set of graphs derived from real-world instances. For the latter set, we use a selection of 100 graphs from the well-established Rome graph set [5], as already used in [28]. This sample contains 10 graphs from each group of graphs with sizes $|nodes| = 10, 20, \ldots, 100$ and covers the Rome set well [28]. Density ($|edges|/|nodes|$) ranges from tree-like (~ 1) to quite dense (1.61). We generated the flat scale-free graphs based on the model proposed by Bollobás *et al.* [17], with 10 graphs for each graph size from 7 to 100 nodes. In these generated graphs we controlled for edge density such that $|edges|/|nodes|$ is up to 1.22. Our decision to use scale-free graphs is motivated by the fact that scale-freeness is often observed in graphs stemming from important application areas like biology and the social sciences. To obtain a grouping for each of these graphs an edge-compression heuristic [21] was applied. Thus, our full corpus consists of 940 flat-graphs, 100 Rome graphs, and the corresponding 1040 grouped graphs, called *power-graphs*. We then attempted to obtain a layout for each graph using our MIP, CP and SAT

For our experiments, the components of the objective function were weighed in the following order. The stress component was given a weight of four, the compartment compactness was given a weight of

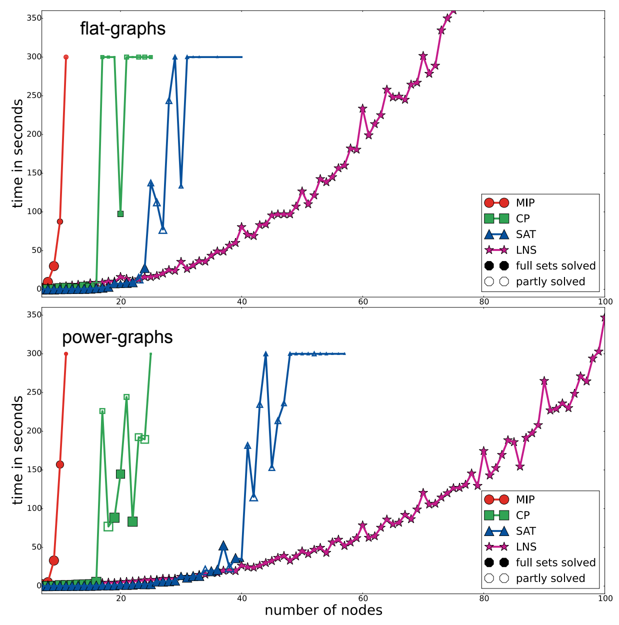


Fig. 7: Median solve times for flat-graphs (top) and power-graphs (bottom). Filled marks represent instances for which optimal results were found in under 5 minutes. Hollow marks indicate not all instances were solved in that time. Size of the marks indicate number of instances solved.

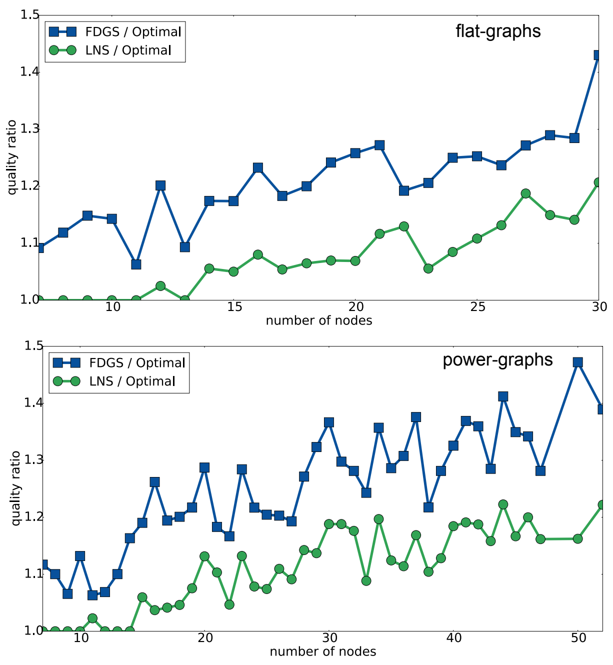


Fig. 8: Average quality of objective obtained with the Large Neighbourhood Search (LNS) heuristic and the starting layout computed using Force-Directed Grid-Snap (FDGS) compared to the optimal objective for flat-graphs (top) and power-graphs (bottom). This is shown with respect to graph size.

two ($\alpha = 1/2$), and the entire layout compactness was given a weight of three ($\beta = 3/4$). These values yielded the most desirable results, and can be modified to fit the needs and expectations of the users.

Running the solvers on dense power-graphs of different sizes showed that the SAT solver was able to solve graphs of larger sizes, while CP and MIP solvers were slower. Fig. 7 shows the median running time for all graphs up to the largest solved instance with 57 nodes. The results of Fig. 7 are presented in Table 1. While these results tend to favor SAT over MIP we cannot say definitively under what conditions such layout models are better suited to one solver over the other. Our intuition here is that SAT is typically well suited to problems with small variable domains and highly disjunctive constraints, whereas MIP can handle large domains gracefully only when a linear relaxation gives a good approximation to disjunctions.

	powergraph			**flatgraph**		
	<3s	*<1m*	*<5m*	*<3s*	*<1m*	*<5m*
MIP	7	9	11	7	9	11
CP	15	16	25	13	16	25
SAT	25	38	57	18	24	36

Table 1: The highest nodes count of graphs solved by MIP, CP, and SAT; categorized into three timeframes. It is clear that SAT performed best, followed by CP, with MIP having the worst performance for both power-graphs and flat-graphs.

5 Heuristic for Larger Networks

The results presented in Sect. 4.4 indicate that SAT may be used to find optimal ultra-compact grid layout for small grouped graphs reliably in around a second, which may be suitable—for example—for a web-service. It can be argued that interactive visualizations of small neighborhoods are useful in exploring very large graphs, through filtering or semantic zooming that restricts the view to a subgraph or aggregated overview. Still, being able to reliably visualize networks with hundreds of nodes gives a lot more flexibility.

We use *large neighbourhood search* (LNS), a class of meta-heuristics methods are currently the method of choice for solving various transportation and scheduling problems. The basic approach is to explore a large neighbourhood around the current solution and iteratively move to a high-quality neighbouring solution until no improvement is possible. One way in which the search can be done is to use a generic constrained optimisation technique to search the neighbourhood for the best solution. This guarantees that the search only finds solutions that respect the problem constraints. This is why we decided to use LNS; with other meta-heuristic techniques like simulated annealing it would have been difficult to ensure that the containment and non-overlap constraints were satisfied during the search.

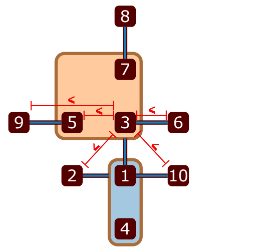
The LNS heuristic is intuitively simple: find an initial solution and then iteratively improve it by choosing a set of nodes for improvement. The space of their possible positions forms the neighbourhood for the search. Their new position is found by using MIP to solve the model given in Section 3 with some additional constraints fixing the relative position of the other nodes. The reason that we used MIP is that while MIP was slower than both CP and SAT for solving the original model we found that it was the fastest method for solving this more constrained subproblem. We believe this is for two reasons. The first is that the addition of relative position constraints removes most of the disjunctions from the model so that the linear relaxation gives a good approximation to the underlying problem. The second is that MIP solvers such as Cplex support *warm start* when solving similar problems. Algorithm 1 gives an overview of our heuristic.

The first step is to find an initial layout for the grouped network with a constraint-based force-directed approach. We implemented the "grid-snap" technique described by Kieffer et al. [30] in the `cola.js` [6] browser-based constraint-layout library. We extended this method to handle group-hierarchy containment inside rectangular regions using separation constraints as described in [18]. The layout obtained by this heuristic Force-Directed Grid Snap (FDGS) approach for a 45-node graph is shown in Fig. 11(a).

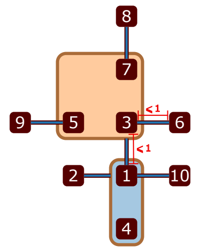
From this initial layout two types of additional constraints are added to the model (Sect. 3) to massively reduce the search space. First, we generate inequality constraints over the x- and y-positions for pairs of nodes (Fig. 9) locking their horizontal and vertical order. To allow the solver to move a neighbourhood of nodes these constraints are se-

Algorithm 1 Grid Layout Heuristic

```
1:  procedure GRID LAYOUT ( Graph $g$, Model $m$ )
2:      $l \leftarrow$ getForceDirectedLayoutWithGridSnap($g$)
3:      $d \leftarrow$ getDataWithTightConstraints($l$)
4:      SolverLoad($m, d$)
5:      $b \leftarrow 10$, $c \leftarrow nil$, $t \leftarrow |B|/5$ seconds
6:      for each $c \leftarrow$ getNextNodeOrContainer($g, c$) do
7:          relaxOrderingConstraints($c$)
8:          $\delta \leftarrow \delta \cup$ getContainedNodes($c$)
9:          $lc1 \leftarrow$ getFreeLeavesDirectlyConnected($c$)
10:         $\delta \leftarrow \delta \cup lc1$
11:         $\delta \leftarrow \delta \cup$ getFreeLeavesDirectlyConnected($lc1$)
12:         $\delta \leftarrow \delta \cup$ getContainedLeavesDirectlyConnected($c$)
13:         relaxUpTo_b_nodes($\delta, b$)
14:         $val \leftarrow$ runCPLEXsolver(timeout=t)
15:         setWarmStart(val)
16:         updateConstraints($val$)
17:     end for
18:     for each $leaves \leftarrow$ getUpTo(FreeLeaves($g$), $b$) do
19:         relax($leaves$)
20:         repeat 12-15
21:     end for
22:     return $val$
23: end procedure
```

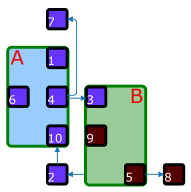


(a) Horizontal ordering constraints

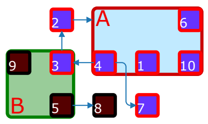


(b) Edge-length constraints

Fig. 9: Constraints types derived from the grid-snap layout to be added to the layout model. Ordering constraints for the pairwise relative positions between nodes in x and y dimension are added for all nodes pairs. (a) shows the horizontal ordering constraints for node 3, which restrict its x position to be less or equal to those of 6 and 10, and larger or equal than those of 9, 5, and 2. The constraints for the y position are obtained in the same way. (b) shows the edge-length constraints for node 3, in this case all distances to adjacent nodes have bound 1.



(a) Starting layout: container A is selected for neighborhood detection. Nodes with a blue fill are selected for relaxation. This includes the nodes contained in A, nodes 2 and 7 as directly connected free leaves, and node 3 as directly connected contained node.



(b) Result: Nodes moved to a new position are highlighted with red outline. Node 3 was repositioned within container B, decreasing the container size, and container A was assigned a more horizontal shape, allowing the drawing to fit into a much smaller grid, moving all relaxed nodes. The length of the edge between nodes 4 and 7 was decreased from 4 to 3. Grid size has been reduced from 5x5 to 5x3.

Fig. 10: A step in the Large-Neighborhood-Search heuristic.

lectively relaxed, as described below. Additional constraints on edge-length further restrict the search to only equal or shorter edges. This is done by bounding the manhattan distance between adjacent nodes by their manhattan separation from the FDGS layout.

In order to obtain improvements in reasonable time, we iteratively relax the ordering constraints for a subset δ of nodes and run the solver on this relaxed model. New ordering constraints for the nodes in δ and potentially tightened bounds on the edge-lengths are derived from the resulting layout and added to the model for the next iteration, where a solver warmstart is used to speed up the computation. The relaxation is divided into two main parts, represented by the `for` loops in Algorithm 1, lines 6 and 18. The first part processes neighborhoods of nodes, the second is a postprocessing to find the best placement for *free leaves*, i.e. leaf nodes not contained in any other node.

In each iteration of part one the selection of nodes for relaxation is as follows: first, we select a node c, whose neighborhood will be relaxed, to be included in δ (Function getNextNodeOrContainer in Algorithm 1). In case there are contained nodes, we first simply pick c from the list of containers ordered by size, i.e. the largest container in the first iteration, followed by the second largest, and so on. Otherwise we consider each node as an empty container, and in each iteration select one of them, in random order. We now add further nodes to δ up to a small bound b on the size of δ. In our implementation $b = 10$ gave a good tradeoff between running time and quality improvement. First, we randomly select up to b nodes in c. If c has fewer nodes than b, we add further nodes in the following order until $|\delta| = b$: The group $lc1$ of free leaves that are adjacent to c. The group of free leaves that are adjacent to nodes in $lc1$. Contained leaf nodes adjacent to c.

After relaxing the ordering constraints for the nodes in δ, we run the solver for a limited time t. Fig. 10 illustrates such a step of the heuristic. We choose $t = |B|/5$ such that the total run-time for the algorithm is always bounded proportionally to the size of the graph. The result is used to initialize the next iteration and the solver warmstart.

We iterate until each container (or each node in case there are no containers) has been selected once (for-loop at Line 6 of Alg. 1). Afterwards, we relax sets of free leaves independent of a container, and rerun the solver, until all free leaves have been relaxed once (for-loop at Line 18 of Algorithm 1). The final layout is shown in Fig. 11(b).

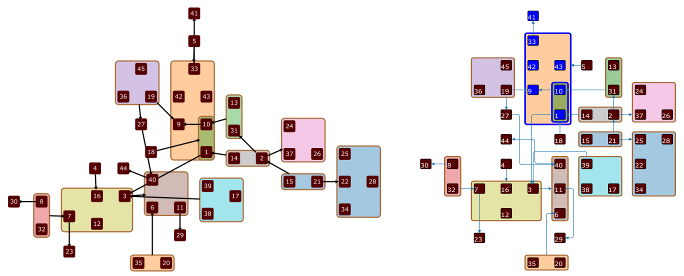
We evaluated the LNS heuristic on our graph corpus. The reduction in search space leads to significantly faster solves as can be seen in Fig. 7. Fig. 8 compares the layout quality of LNS and FDGS with the optimal as obtained by SAT. Across all graphs in our corpus the mean quality ratio for FDGS was ~ 1.24 while for LNS it was ~ 1.1, i.e. LNS was typically twice as close to the optimal as FDGS. Visually, FDGS gives a much less compact layout with a significantly larger total edge length compared to layout refined by LNS as evidenced by the side-by-side comparisons in Figs. 11 and 12 [3].

6 RELATED WORK

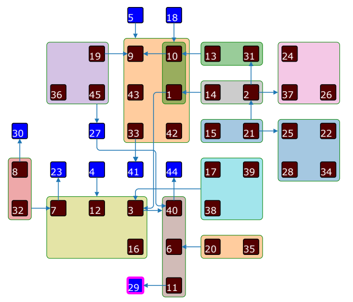
The most widely-used family of automatic layout methods for undirected graphs are based on *force-directed* layout [32]. These methods iteratively place nodes such that edge-lengths become relatively uniform while disconnected nodes are spaced further apart. This approach is attractive because the basic variants are easy to implement, it does a reasonable job of untangling the structure of small graphs and clustering nodes so that proximity implies connectivity (R2). However, the results are very organic – the antithesis of grid layout.

Rohrschneider et al. tried to overcome that problem for biological networks by first computing a stress-based node-placement on a grid, followed by an edge routing heuristic [44]. This approach does not allow group information to be taken into account, and the graph structure is hard to conceive from the layouts. Recent work from Kieffer *et al.* [30] explored augmenting the objective function of a constrained-force-directed technique to prefer nodes placed at grid-points, thereby creating a compromise between a grid-aesthetic and R2. This method (extended to grouped graphs) provides the starting point for our LNS.

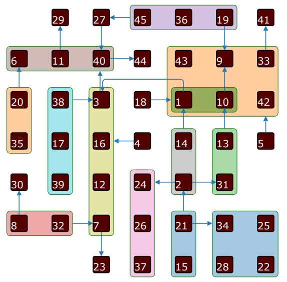
[3]More examples highlighting the difference in quality between FDGS layout and that obtained by FDGS with LNS refinement are available online [4].



(a) Left: Force-directed layout with containment. Right: Force-directed layout with grid-snap. Solve time: 2.2 seconds. Grid size: 10x11. Total Edge Length: 33. Edge Crossings: 4. Objective 35.89% higher than optimum. The first neighbourhood considered by the LNS search is highlighted.



(b) LNS outcome. Highlighted nodes are the last neighbourhood with relaxed constraints, the node with pink outline was the only one moved by the solver. Solve time: 43 seconds. Grid size: 9x8. Total Edge Length: 29. Crossings: 4. Objective 17.94% higher than optimum.



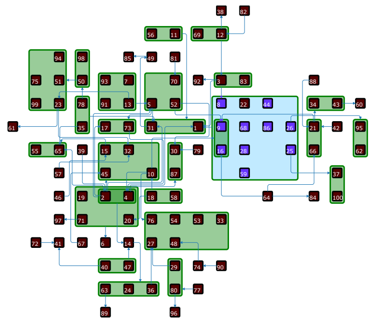
(c) Optimum. Only one edge has more than unit length. Grid size is optimal at 7x7. Total Edge Length: 25. Crossings: 0. Solved by SAT in 99.2 seconds.

Fig. 11: A network with 45 nodes arranged in four ways: the force-directed approach that forms the basis of our LNS approach; the grid-snap approach that allows us to derive the constraints for the LNS approach; the final outcome of the LNS approach; the final optimal outcome of the SAT.

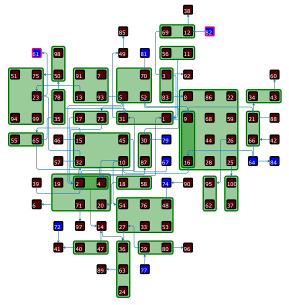
A layout exploration in the specific domain of Metro-map layout from Nöllenburg and Wolff [40] was similar to ours in spirit in its attempt to obtain high-quality layout through the use of optimal (MIP) techniques. However, the metro-map layout problem is significantly constrained in that the topology is already given by the geographical positions of the stations. As a layout adjustment problem rather than completely free arrangement of nodes, it is therefore more similar in terms of tractability to the LNS approach explored in Sect. 5.

Orthogonal layout approaches seek to represent edges with axis-parallel segments, preferably with only a small number of right-angle bends. From the approaches that were proposed, the planarisation-based Topology-Shape-Metrics (*TSM*) framework [11] has proven to be by far the most successful in practice. TSM first fixes an embedding for the planarized input graph, then solves bend minimization for this embedding to achieve an orthogonal shape, and in the final compaction step computes node positions for this shape.

However TSM has limitations that inspire the work presented in this paper: Primacy is given to minimizing edge crossings, prohibiting a good compromise between aesthetic criteria, and even optimal solutions for a single phase (e.g. calculated using ILP, MIP or SAT strategies [12,24,31]), will generally not lead to a solution close to the optimum with respect to all optimisation criteria, see Fig. 2(a). Orthogonal methods also typically do a poor job of handling nodes of widely varying dimensions, and they are difficult to extend for main constraints in applications, in particular grouping of nodes and flow direction. An exploration of applying optimal methods (ILP and SAT) to particular graph-theoretical problems related to orthogonal-drawings was considered by Biedl *et al.* [14] but their results are not readily applicable to practical layout. Betz et al. [13] integrate upward crossing minimization into a TSM approach to support non-uniform node heights for layouts of directed graphs.



(a) Force-directed layout with grid-snap. Grid size: 16x14. Total Edge Length: 138. Crossings: 52. Solve time: 14.4 seconds. The first neighbourhood considered by the LNS search is highlighted.



(b) LNS. Highlighted nodes are the last neighbourhood with relaxed constraints, the node with pink outline was the only one moved by the solver. Grid size: 13x14. Total Edge Length: 135. Crossings: 50. Solve time: 348 seconds.

Fig. 12: A network with 100 nodes.

A number of other researchers have investigated the use of meta-heuristic approaches to graph layout. Davidson and Harel [16] investigated the use of simulated annealing (SA) for undirected graphs. Harel and Sardas [26] used SA to beautify a layout drawn with a planarisation-based approach. Barsky *et al.* [10] and Kojima *et al.* [33] propose methods based on SA and local-search (respectively) for biological networks. The work of Barsky *et al.* is the most similar because they layout the nodes on a grid, however the layout is not particularly compact and does not use rectangular compartments. To our knowledge, ours is the first use of LNS in graph layout.

7 Conclusion and Further Work

We have introduced a new ultra-compact, grid-like layout aesthetic for node-link diagrams with arbitrary containment that is motivated by the grid arrangements that are used almost universally by designers in typographical layout. We have explored whether generic constrained optimisation techniques (MIP, CP and SAT) are now fast enough to be used for high-quality drawings of this kind. We found that SAT was the most effective, and quite practical for producing high-quality layouts for graphs of up to 20 nodes in under a second - useful, for example in interactive contexts where it is possible to obtain an aggregated or partial view of a larger network and graphs of up to 50 nodes in a few minutes may be useful for producing canonical offline views.

Although this paper is about solving a set of models for network layout to optimality, it is open for debate whether our particular model represents the "best possible visualisation" and we do not claim this. Rather, this is precisely the point of this paper: by rapidly modeling different types of layout through declarative techniques we are able to rapidly experiment with different approaches to the layout problem. A number of the ideas for layout presented here (e.g. reorientable nodes, ultra-compactness, multi-directional flow layout; all while respecting arbitrary group containments) are to our knowledge very novel and very difficult to experiment with by any other means.

Another use of the optimal techniques, however, is in finding a baseline against which approximate methods may be compared to assess quality. This was demonstrated in our evaluation of the LNS metaheuristic approach. Our evaluation showed that the LNS method could produce compact layouts for graphs with up to 100 nodes in a few minutes that are usually within 20% of the objective function's optimum.

In general, the ease of experimenting with different layouts through simple edits to the declarative model opens up a world of possibilities that we hope to explore in future before embarking on engineering faster heuristics. A significant open challenge is a layout model that somehow incorporates routing in a way that is efficiently solvable to optimality. To us, this remains the "holy grail".

Acknowledgments

We acknowledge the support of the Australian Research Council through Discovery Project DP140100077

References

[1] 2014 Graph Drawing Conference contest: http://graphdrawing.de/contest2014/topic2.html.

[2] CPLEX MIP solver: http://www.ibm.com/software/commerce/optimization/cplex-optimizer.

[3] d3.js: http://d3js.org.

[4] https://github.com/Vahany/HqUcGl-of-Grouped-Networks.

[5] Rome Graphs: http://www.graphdrawing.org/data/.

[6] webcola: http://marvl.infotech.monash.edu/webcola.

[7] yFiles automatic network layout software: http://www.yworks.com/en/products/yfiles/.

[8] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002.

[9] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[10] A. Barsky, T. Munzner, J. Gardy, and R. Kincaid. Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1253–1260, 2008.

[11] C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *Software Engineering, IEEE Transactions on*, (4):538–546, 1986.

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

[13] G. Betz, A. Gemsa, C. Mathies, I. Rutter, and D. Wagner. Column-based graph layouts. *Journal of Graph Algorithms and Applications*, 18(5):677–708, 2014.

[14] T. Biedl, T. Bläsius, B. Niedermann, M. Nöllenburg, R. Prutkin, and I. Rutter. Using ilp/sat to determine pathwidth, visibility representations, and other grid-based graph drawings. In *Graph Drawing*, pages 460–471. Springer, 2013.

[15] M. Codish and M. Zazon-Ivry. Pairwise cardinality networks. In *16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6355 of *LNCS*, pages 154–172. Springer, 2010.

[16] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996.

[17] T. Dwyer, N. Henry Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2596–2605, 2013.

[18] T. Dwyer, Y. Koren, and K. Marriott. Ipsep-cola: An incremental procedure for separation constraint layout of graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):821–828, 2006.

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

[20] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *Graph Drawing*, volume 3843 of *LNCS*, pages 153–164. Springer, 2006.

[21] T. Dwyer, C. Mears, K. Morgan, T. Niven, K. Marriott, and M. Wallace. Improved optimal and approximate power graph compression for clearer visualisation of dense graphs. In *Pacific Visualization Symposium (PacificVis), 2014 IEEE*, pages 105–112. IEEE, 2014.

[22] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[23] T. Feydy, A. Schutt, and P. Stuckey. Semantic learning for lazy clause generation. In *TRICS workshop, held alongside CP*, 2013.

[24] G. Gange, P. Stuckey, and K. Marriott. Optimal k-level planarization and crossing minimization. *Graph Drawing*, pages 238–249, 2011.

[25] E. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Graph Drawing*, volume 3383 of *LNCS*, pages 239–250. Springer, 2005.

[26] D. Harel and M. Sardas. Randomized graph drawing with heavy-duty preprocessing. *Journal of Visual Languages & Computing*, 6(3):233–253, 1995.

[27] T. Jankun-Kelly and K.-L. Ma. Moiregraphs: Radial focus+ context visualization and interaction for graphs with visual nodes. In *Information Visualization, 2003. INFOVIS 2003. IEEE Symposium on*, pages 59–66. IEEE, 2003.

[28] A. Kennedy, K. Klein, and A. Nguyen. The graph landscape – a concept for the visual analysis of graph set properties. In Submission.

[29] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow. Hola: Human-like orthogonal network layout. In submission.

[30] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow. Incremental grid-like layout using soft and hard constraints. In *Graph Drawing*, pages 448–459. Springer, 2013.

[31] G. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings. *Integer Programming and Combinatorial Optimization*, pages 304–319, 1999.

[32] S. G. Kobourov. Force-directed drawing algorithms. *Handbook of Graph Drawing and Visualization*, pages 383–408, 2013.

[33] K. Kojima, M. Nagasaki, E. Jeong, M. Kato, and S. Miyano. An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC bioinformatics*, 8(1):76, 2007.

[34] R. E. Korf, M. D. Moffitt, and M. E. Pollack. Optimal rectangle packing. *Annals of Operations Research*, 179(1):261–295, 2010.

[35] K. Marriott, H. Purchase, M. Wybrow, and C. Goncu. Memorability of visual features in network diagrams. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2477–2485, 2012.

[36] A. Metodi and M. Codish. Compiling finite domain constraints to sat with bee. *Theory and Practice of Logic Programming*, 12(4-5):465–483, 2012.

[37] J. Müller-Brockmann. *Grid Systems in Graphic Design*. 1981.

[38] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc: Towards a standard CP modelling language. In *Constraint Programming*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[39] M. Nöllenburg. An improved algorithm for the metro-line crossing minimization problem. In *Graph Drawing*, pages 381–392. Springer, 2010.

[40] M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In *Graph Drawing*, volume 3843 of *LNCS*, pages 321–333. Springer, 2006.

[41] D. Pisinger and S. Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

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

[43] P. Rodgers. A survey of euler diagrams. *Journal of Visual Languages & Computing*, 25(3):134–155, 2014.

[44] M. Rohrschneider, C. Heine, A. Reichenbach, A. Kerren, and G. Scheuermann. A novel grid-based visualization approach for metabolic networks with advanced focus&context view. In *Graph Drawing*, pages 268–279, 2009.

[45] L. Royer, M. Reimann, B. Andreopoulos, and M. Schroeder. Unraveling protein networks with power graph analysis. *PLoS computational biology*, 4(7):e1000108, 2008.

[46] G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. In *Graph Drawing*, volume 2912 of *LNCS*, pages 381–386. Springer, 2004.

[47] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2):109–125, 1981.

[48] H. P. Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.

[49] M. Wybrow, K. Marriott, and P. J. Stuckey. Orthogonal connector routing. In *Graph Drawing*, pages 219–231. Springer, 2010.