Optimal Sankey Diagrams via Integer Programming

David Cheng Zarate*
Faculty of Information Technology
Monash University

Pierre Le Bodic
Faculty of Information Technology
Monash University

Tim Dwyer†
Faculty of Information Technology
Monash University

Graeme Gange
School of Computing and Information Systems
University of Melbourne

Peter Stuckey
School of Computing and Information Systems
University of Melbourne

Abstract

We present the first practical Integer Linear Programming model for Sankey Diagram layout. We show that this approach is viable in terms of running time for reasonably complex diagrams (e.g. more than 50 nodes and 100 edges) and also that the quality of the layout is measurably and visibly better than heuristic approaches in terms of crossing reduction. Finally, we demonstrate that the model is easily extensible (compared to complex heuristics) through the addition of constraints, such as arbitrary grouping of nodes.

Keywords: Visualization. graph drawings. integer programming.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Graph drawings

1 Introduction

Sankey diagrams are useful in presenting quantity and connectivity of flows between entities across time (see Figures 1 and 2 for examples). Usually, Sankey diagrams are drawn with a fixed set of layers (vertical lines) corresponding to different time steps. Although a very old representation (originally used for visualizing energy transfer in steam engines [12]), they are growing in popularity and importance as a method for visualizing event sequence data, such as customer take up of different product categories, or web page views in popular tools like Google Analytics.

With similarities to the layered-graph drawing problem, the placement of entities within layers of a Sankey Diagram to avoid crossings between flows (edges), is a challenging combinatorial problem. It differs from layered graph drawing in that the edges of a Sankey Diagram have thickness corresponding to the value they encode, and so not all crossings have equal importance. The current approaches to arranging nodes in Sankey Diagrams adapt layered graph-drawing heuristics, as described in Section 2.

In this paper we propose an Integer Linear Programming (ILP) model that allows us to use state-of-the-art solver technology to obtain layouts that are provably optimal (Section 3). We show that the optimal layouts are measurably and visibly better, in terms of crossing cost, than that achieved by heuristics (Section 4). While there is a significant cost in terms of running time, we find that the solver is able to find optimal solutions for realistically complex Sankey diagrams in only a few seconds or so. Thus, the technique is already applicable in situations where a highly interactive layout is not necessary (e.g. static diagrams for communication purposes). Layout that is proven to be optimal is also very useful, for example, as a baseline or ground-truth in usability studies or evaluations of heuristics [4].

*e-mail: dche0005@student.monash.edu
†e-mail: tim.dwyer@monash.edu

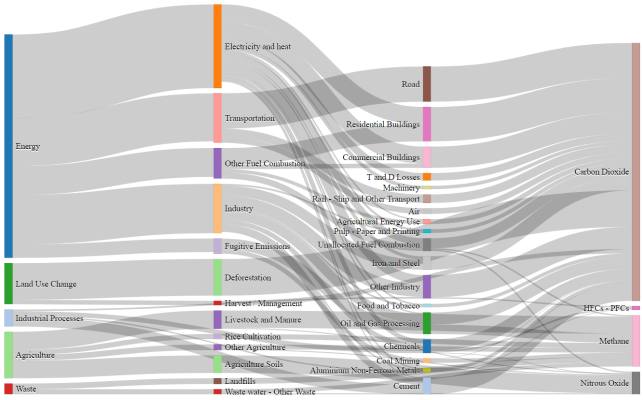

Figure 1: Heuristic Layout (Sugiyama)

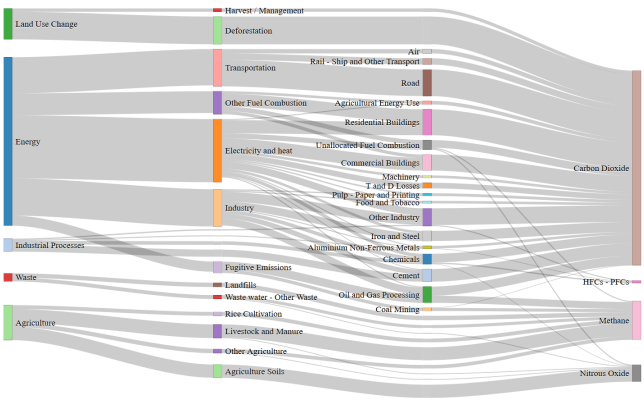

Figure 2: Optimal Layout

Another advantage of our ILP model is that, as a declarative representation of the problem rather than an imperative implementation of a specific algorithm, it is easily customizable or extensible to different applications through the specification of additional constraints. We explore one such possibility—the addition of grouping constraints over the node positions—in Section 5.

2 Related Work

Layered or hierarchical network diagrams are used to represent dependency relations between components that belong to different layers. The readability of such a diagram can typically be measured in terms of edge crossings between layers, which depends on the position of the nodes in each layer. Since the number of node orderings on each layer is factorial in the number of nodes in that layer, the problem of minimizing the number of crossings is very combinatorial. Unsurprisingly, this problem has been proven to be

NP-hard even when there are only two layers [7]. For that reason, past research has focused on developing heuristics methods that can quickly compute a node ordering that gives few crossings, albeit a non-optimal ordering. Sugiyama et al. [13] developed a four-stage heuristic method to draw a hierarchical network. The first two stages of this heuristic handle cycle removal and the assignment of nodes to layers. The third stage in the Sugiyama method defines the node ordering with a barycentric method. In this heuristic approach, one layer has a fixed node ordering and the subsequent layer is able to move its nodes. The free nodes are placed in the barycentric coordinate of its neighbor nodes in the fixed layer. After setting the node ordering of the free layer, this layer is now fixed and the next adjacent layer is free to move its nodes and find its corresponding node ordering. A similar approach, called the median heuristic, is studied by Eades and Wormald [5] where the position of the free nodes is given by the median of its neighbour nodes and not the barycentric. The fourth stage is, given the ordering within levels chosen at the previous step, finding an aesthetic, non-overlapping arrangement of node boxes and routing of edge curves. Various heuristic or mathematical programming techniques are applicable for this phase [3]. This paper focuses on optimal techniques for the third, node ordering phase as it applies to Sankey Diagrams.

Another approach to optimize the readability of a layered graph was introduced by Mutzel [10] and consists in solving the k-level planarization problem. In this research, the aim is to find the minimum set of edges that need to be removed for the diagram to become planar. Mutzel discusses that this approach can provide a clearer representation of a layered network. Gange et al. [6] have further built on Mutzel's work and created a hybrid method that uses both crossing edges minimization and k-level planarization to determine if this can improve the readability of a layered diagram. In this work, an optimization model was developed with an objective function that aims to minimize the weighted sum of the number of crossing edges and the number of removable edges for planarization. They state that a combination of both criteria leads to better unweighted layered drawings than only considering one of the approaches.

Previously mentioned works focus chiefly on the reduction of the number of crossing edges, but they do not consider the case where edges have a weight, i.e. various degrees of importance, which can for instance be represented by different edge widths. In this situation, a crossing between two "thick" edges should be penalized more than a crossing between two "thin" edges. Alemasoom et al. [1] developed a two-step criterion to convey a more readable Sankey diagram that considers a weighted layered graph. They first applied the heuristic approach from Sugiyama's method to find a node ordering that reduces the number of edge crossings. Node placement within a layer is solved with a linear program that aims to minimize the weighted sum of the distance between two nodes. This layout is not optimal, as it uses a heuristic method to find the node ordering. Further, this heuristic does not take into consideration the widths of the edges.

3 An Integer Linear Programming Model

An ILP model encodes a problem and its solutions using integer variables, linear constraints on these variables, and a linear objective function that is either maximized or minimized [2]. ILP can model many optimization problems and is NP-hard to solve in general. However, there exists very efficient software (called solvers) that allow many practical NP-hard problems to be solved to optimality within reasonable time once modeled as ILPs. In many cases, ILP models are easier to write and faster to solve (using ILP solvers) than dedicated algorithms. The main reason is that ILP solvers are very mature products: they have been developed for decades and benefit from 50+ years of research [9]. They incorporate techniques that work on many different problems and are likely to work on newly defined ones as well. Further, the expressiveness and generality of ILP allows models to be extended without having to change algorithms. We demonstrate this property in Section 5. Note that ILP solvers are usually referred to as "MIP solvers" as they can solve Mixed-Integer Programs, which are more general than ILPs.

Input Data

We first describe the input of the Sankey diagram problem. A graph $G = (V, E)$ is given with a partition of the nodes across k layers, such that no edge has both endpoints in the same layer (i.e. the graph is k-partite), and weights w_{uv} for each edge $uv \in E$. We denote by $\mathscr{L} = \{0, \dots, k-1\}$ the set of all layers and by $\mathscr{L}_1 = \{0, \dots, k-2\}$ the set of all layers except the last one. We further denote the set of nodes in layer $k \in \mathscr{L}$ by V_k, and we thus have $V = V_0 \uplus \cdots \uplus V_{k-1}$, where $\uplus$ is the symbol for the disjoint union. Finally, we define E_k for all $k \in \mathscr{L}_1$ as the set of edges that have one endpoint in layer k and the other in layer $k+1$. For all $uv \in E_k, k \in \mathscr{L}$, we will use the convention that the first node $u \in V_k$ and the second node $v \in V_{k+1}$.

In our models, we will suppose without loss of generality that any edge $uv \in E$ satisfies $u \in V_k$ and $v \in V_{k+1}$, i.e. any edge has endpoints in two consecutive layers. If this is not the case in the original graph, additional nodes can be introduced to transform a "long" edge into multiple edges between consecutive layers, with weights on the "small" edges being equal to the weight of the "long" edge. We therefore have $E = E_0 \uplus \cdots \uplus E_{k-2}$.

Decision Variables

We define two types of binary variables:

- $\forall k \in \mathscr{L}, \forall u_1, u_2 \in V_k, u_1 \neq u_2$, x_{u_1,u_2} is a binary variable that indicates the relative position of the nodes u_1 and u_2 in a layer k, i.e. $x_{u_1,u_2} = 1$ if and only if u_1 is "above" u_2 on layer k (assuming vertical layers).
- $\forall k \in \mathscr{L}_1, \forall u_1v_1, u_2v_2 \in E_k, u_1v_1 \neq u_2v_2, c_{u_1v_1,u_2v_2}$ is the binary variable that indicates whether edges u_1v_1 and u_2v_2 cross.

Objective Function

As discussed in Section 2, the objective function to optimize must quantify the readability of a diagram. We thus define the objective function to minimize as the sum, over every pair of edges, of their crossing area. For a pair of edges that cross, the crossing area is simply the product of the weights of these edges. Otherwise, if two edges do not cross, the area naturally equals zero. Note that although we are calling this a "crossing area", this only corresponds to an area in the geometric sense if the width of the drawn edges is linear in the weight, and if the crossing has the shape of a rectangle. Formally, the objective function is given by

$$\text{Minimize:} \quad \sum_{\substack{k \in \mathscr{L}_1 \\ u_1v_1, u_2v_2 \in E_k \\ u_1v_1 \neq u_2v_2}} (w_{u_1v_1} \times w_{u_2v_2}) c_{u_1v_1,u_2v_2}. \tag{1}$$

Note that if every edge has weight 1, then (1) simply minimizes the number of crossings.

Constraints

The next step to develop the full model is to define the constraints of the model. By definition of x_{u_1,u_2}, either u_1 is above u_2 or, exclusively, u_2 is above u_1. This can be encoded by the constraint

$$x_{u_1,u_2} + x_{u_2,u_1} = 1 \qquad \forall k \in \mathscr{L}, u_1, u_2 \in V_k, u_1 \neq u_2. \tag{2}$$

Constraint (2) enforces consistency for any two nodes in the same layer, i.e. both of them cannot be "above" the other one. However, it does not prevent this phenomenon for three of more nodes, e.g. u_1 is above u_2 ($x_{u_1,u_2} = 1$), u_2 is above u_3 ($x_{u_2,u_3} = 1$), and u_3 is above

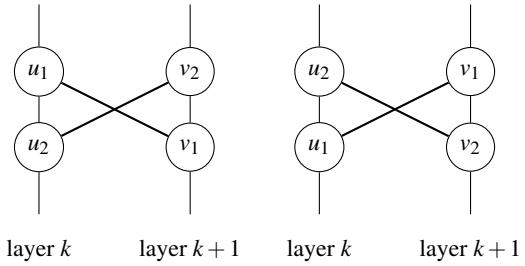

Figure 3: The two configurations in which edges u_1v_1 and u_2v_2 cross.

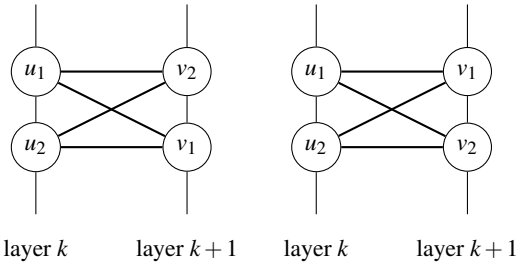

Figure 4: Two (of four) configurations for nodes u_1, v_1, u_2 and v_2.

u_1 ($x_{u_3,u_1} = 1$). We therefore need to enforce the transitivity of the relation "is above", and we do so by defining the constraint

$$x_{u_3,u_1} \geq x_{u_3,u_2} + x_{u_2,u_1} - 1 \qquad \begin{cases} \forall k \in \mathscr{L}, u_1, u_2, u_3 \in V_k, \\ u_1 \neq u_2 \neq u_3, u_1 \neq u_3. \end{cases} \tag{3}$$

This constraint is only active (i.e. it forces $x_{u_3,u_1} = 1$) when both $x_{u_3,u_2} = x_{u_2,u_1} = 1$, which means it forces u_3 to be above u_1 if u_3 is above u_2, and u_2 is itself above u_1. Doing so for every triplet of nodes in each layer incurs a total ordering of the nodes on that layer.

The variables c must now be defined and linked to the positioning variables x. For a layer $k \in \mathscr{L}_1$ and two edges u_1v_1 and u_2v_2 in E_k, the two edges cross if and only if u_1 is above u_2 and v_2 is above v_1, or if u_2 is above u_1 and v_1 is above v_2, as shown in Figure 3. The following constraints enforce c to take value 1 in these two cases:

$$c_{u_1v_1,u_2v_2} + x_{u_2,u_1} + x_{v_1,v_2} \geq 1, \tag{4}$$

$$c_{u_1v_1,u_2v_2} + x_{u_1,u_2} + x_{v_2,v_1} \geq 1, \tag{5}$$

for all $\forall k \in \mathscr{L}_1, \forall u_1v_1, u_2v_2 \in E_k, u_1v_1 \neq u_2v_2$. Indeed, in Figure 1, on the left, we have u_1 above u_2, i.e. $x_{u_2,u_1} = 0$, and also $x_{v_1,v_2} = 0$, thus in this case constraint (4) enforces $c_{u_1v_1,u_2v_2} = 1$. On the right of Figure 1, we have $x_{u_1,u_2} = 0$ and $x_{v_2,v_1} = 0$, therefore constraint (5) ensures that $c_{u_1v_1,u_2v_2} = 1$. Note that we do not need to enforce c variables to take value 0 when the corresponding edges are not crossing since we are minimizing (1), and all weights are positive.

Finally, we formally specify that variables x and c are binary:

$$x_{u_1,u_2} \in \{0,1\} \qquad \forall k \in \mathscr{L}, \forall u_1, u_2 \in V_k, u_1 \neq u_2, \tag{6}$$

$$c_{u_1v_1,u_2v_2} \in \{0,1\} \qquad \forall k \in \mathscr{L}_1, \forall u_1v_1, u_2v_2 \in E_k, u_1v_1 \neq u_2v_2. \tag{7}$$

Improving performance with additional constraints

Constraints (2), (3), (4), (5), (6) and (7) define the set of feasible solutions of our ILP model, each of which corresponds to a valid Sankey diagram. However, additional constraints can be added that may improve the performance of ILP solvers. Such constraints can be devised by discovering additional structure in the problem.

A first simple constraint is due to the symmetry of the "crossing" relationship encoded by variables c, in that if the edge $u_1v_1 \in E_k$ crosses an edge $u_2v_2 \in E_k$, then obviously u_2v_2 crosses u_1v_1. The symmetry constraint is set by:

$$c_{u_1v_1,u_2v_2} = c_{u_2v_2,u_1v_1}. \tag{8}$$

Although it is a direct observation, ILP solvers currently do not automatically detect and exploit this equality. Adding (8) allows for better presolving, i.e. the ILP solver can simplify the input model by deleting redundant variables and constraints, which results in a slight improvement in performance.

A second constraint comes from the observation that if all four possible edges $u_1v_1, u_1v_2, u_2v_1, u_2v_2$ exist for two nodes u_1 and u_2 in layer k and two nodes v_1 and v_2 in layer $k+1$, then exactly two of them must cross:

$$c_{u_1v_1,u_2v_2} + c_{u_1v_2,u_2v_1} = 1. \tag{9}$$

Figure 4 depicts the fact that a crossing must occur regardless of the positions of the nodes. Constraint (9) is implied by constraints (2-7) all together, but not by (2-5) only. This is where constraint (9) improves the performance of ILP solvers: ILP solvers solve a continuous relaxation of the ILP (i.e. by ignoring constraints (6) and (7)) very efficiently (typically with the simplex algorithm) to guide and shorten the search for an optimal solution, but this does not provide a feasible binary solution in general. Indeed, with constraints (2-5) only, fractional variables are allowed, and an optimal solution to this continuous problem is $x = 0.5$ and $c = 0$ and its objective value is 0. By adding constraint (9), the ILP solver will detect that at least one crossing will arise every time four edges are in the configuration we have described. This produces significant performance improvements.

Further improvements with branching priorities

In the ILP model we define, the integrality of variables x implies the integrality of variables c: if we omitted constraint (7) from the ILP model, and thus c were allowed to take fractional values, then c would still be assigned integer values by the solver. This can be easily established using constraints (4) and (5).

One way to exploit this observation and increase performance is therefore to completely remove constraint (7) from the model. However, a better improvement can be obtained by keeping constraint (7), and instead increasing the branching priorities of all variables x. Branching priorities are used in the enumeration algorithm that is central to all ILP solvers, the Branch-and-Bound algorithm, which chooses to "branch" on some integer variables to recursively explore the solution space. By changing the branching priorities, we indicate to the solver that only variables x need to be branched on, as the integrality of variables c will follow. In the experimental setup of Section 4, setting a higher priority to the x variables than to the c variables improved the solving time by a factor of 2 for the smaller instances, up to a factor of 10 for the larger ones.

4 Results

We tested the described model on an AMD Opteron 6272 processor at 2.7 GHz with 32GB of RAM running Ubuntu 16.04 LTS. We used Zimpl v3.3.4 to model the ILPs and Gurobi v7.5.1 (single-threaded) to solve them. The D3's Sankey plugin was used to plot the heuristic and optimal diagrams.

A real world example

We have tested our method on the "World Greenhouse Gas Emissions" data from the World Resources Institute [8]. After transforming the "long" edges of the graph into "short" edges and adding the

$\lvert V_k\rvert$	k	Time (in seconds) to	
		Optimal Layout	Optimality Proof
9	5	0.1	0.8
	6	0.5	1.0
	7	0.7	1.4
	8	1.6	2.3
	9	3.1	3.9
	10	2.8	4.2
	11	4.0	5.8
	12	5.4	10.3
10	5	1.2	2.0
	6	3.3	4.7
	7	6.4	9.1
	8	7.0	9.4
	9	12.8	16.0
	10	19.5	22.9
11	5	9.6	11.8
	6	12.5	17.5
	7	36.7	41.7
	8	52.9	73.7
12	5	16.8	23.4
	6	76.0	104.0

Table 1: ILP Experiments for random examples

corresponding dummy nodes, as described in Section 3, this example has 4 layers, 55 nodes and 100 edges. The node ordering in the layout of the Sankey diagram shown in Figure 1 has been computed using Sugiyama's heuristic method [13]. In contrast, Figure 2 displays the layout resulting from computing the node ordering with the ILP method we propose. The optimal layout avoids many of the long and hard-to-follow crossing edges of the heuristic layout, making the diagram much easier to interpret. Furthermore, the ILP solver returned the solution (and proved its optimality) in 3 seconds. We believe that the trade-off between the quality of the layout and the running time makes this ILP model appealing for many applications.

Test on synthetic data

We have generated synthetic graphs $G = (V, E)$ to further test the performance of our model. The generation of G is controlled by three parameters: the number of layers k of G, the number of nodes per layer $|V_k|$, and the total number $|E|$ of edges of the graph. In practical Sankey diagrams, the number of nodes needs not be constant across all layers, but this allows us to better analyze the behavior of the running time in that parameter, as the number of possible node configurations is then simply $(|V_k|!)^k$.

Typical Sankey diagrams are fairly sparse even with the addition of dummy nodes to split the long edges into consecutive short edges. For these experiments, the edge density was set to be 20% of possible combinations between two consecutive layers (i.e. each node from a layer would be connected to the 20% of nodes of the subsequent layer). Table 1 shows the time that it takes to find the optimal layout, as well as the time required to prove its optimality (this includes the time to find the layout). Each value in the table is the average time over ten random graphs with the same parameters. Unsurprisingly, the running time generally increases as either $|V_k|$ or k increases, but in a much slower fashion than $(|V_k|!)^k$, thanks to the efficiency of ILP solvers. It is remarkable that a significant part of the computation time is devoted to finding the best solution, even though, in our tests, the ILP solver always finds a dummy solution as well as a few improving solutions within the first second. This observation supports our decision to use ILP, as it seems that very good or even optimal solutions for Sankey diagrams are very hard to find and that some degree of enumeration is most likely required.

Note that in an interactive setting, ILP solvers can output the best solution found so far, while still looking for improving solutions.

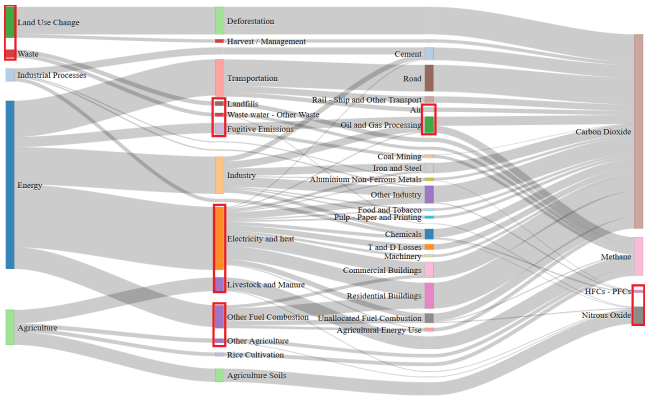

Figure 5: Sankey Example with Grouping Constraints

Indeed, for the "real world" data and all synthetic data, a first solution was always found during the first second of the search. For some applications, this may be a preferable setup than a heuristic that quickly terminates but offers no possibility of improvement over the first solution.

5 Extending the Sankey Diagram with Group Constraints

We provide an example of the extensibility of ILP models by defining group constraints between nodes on a given layer. For some applications it may be desirable to represent some nodes as a group or in a common box on a layer. For example, the nodes might represent divisions of a larger company, hence grouping them together makes the impact of the company easily visible.

We extend the model in Section 3 with additional input data that defines groups. Each group $U \subset V_k$ is a set of nodes belonging to the group on a layer k. For each node u in layer k that does *not* belong to the group U, we define a binary variable $y_{u,U}$ that is equal to 1 if and only if u is above every node in the group U. For a group U in layer k, we can thus enforce all nodes within U to have the same relative position with respect to each node outside of U with the constraint:

$$x_{u_1,u_2} = y_{U,u_2} \qquad \forall u_1 \in U, u_2 \in V_k \setminus U. \tag{10}$$

Figure 5 has been obtained by specifying six arbitrary groups of nodes in different layers with the same data as used in Section 4, and adding to the models the corresponding y variables and constraints (10). One interesting feature of these group constraints is that even though we are adding variables and constraints, the problem is solved faster. Indeed, not only is the set of feasible solutions significantly reduced by the introduction of groups, but fixing the value of one y variable also fixes $|U|$ variables of type x, which allows the space of solutions to be searched by ILP solvers more efficiently.

6 Conclusion and Future Work

We introduced a news criterion to find an optimal layout of a Sankey diagram by reducing the total area of crossing edges. This method displayed a more readable drawing than using a heuristic method that also finds the order of nodes in all layers. Furthermore, we provided an ILP model that finds the optimal node ordering in each layer and this model can be easily extended by adding other constraints to maintain predefined nodes to be kept together in a layer.

While the running time is already viable for moderate-size diagrams, further improvements may be achieved by refinements to this model (or others). Further scalability to very large problems is possible using large-neighborhood search techniques [11] which find optimal solutions for part of the diagram, in an iterative manner across the whole diagram.

REFERENCES

[1] H. Alemasoom, F. Samavati, J. Brosz, and D. Layzell. Energyviz: an interactive system for visualization of energy systems. *The Visual Computer*, 32(3):403–413, 2016.

[2] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*, vol. 271 of *Graduate Texts in Mathematics*. Springer, 2014. doi: 10.1007/978-3-319-11008-0

[3] T. Dwyer, K. Marriott, and M. Wybrow. Dunnart: A constraint-based network diagram authoring tool. In *Graph Drawing*, vol. 5417, pp. 420–431. Springer.

[4] T. Dwyer, N. H. Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE transactions on visualization and computer graphics*, 19(12):2596–2605, 2013.

[5] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.

[6] G. Gange, P. J. Stuckey, and K. Marriott. Optimal k-level planarization and crossing minimization. In *Graph Drawing*, vol. 6502, pp. 238–249. Springer, 2010.

[7] M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[8] W. R. Institute. World greenhouse gas emissions, 2005. Retrieved from `http://www.wri.org/resources/charts-graphs/world-greenhouse-gas-emissions-2005`.

[9] M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, eds. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010. doi: 10.1007/978-3-540-68279-0

[10] P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. *SIAM Journal on Optimization*, 11(4):1065–1080, 2001.

[11] D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.-Y. Potvin, eds., *Handbook of Metaheuristics*, pp. 399–419. Springer US, Boston, MA, 2010. doi: 10.1007/978-1-4419-1665-5_13

[12] M. Sankey. Introductory note on the thermal efficiency of steam-engines. In *Minutes of Proceedings of the Institution of Civil Engineers*, vol. 134, pp. 278–283, 1898.

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