An Experimental Comparison of Orthogonal Compaction Algorithms*
(Extended Abstract)

Gunnar W. Klau[1], Karsten Klein[2], and Petra Mutzel[3]

[1] Technische Universität Wien, Austria, gunnar@ads.tuwien.ac.at
[2] MPI für Informatik, Saarbrücken, Germany, karsten@mpi-sb.mpg.de
[3] Technische Universität Wien, Austria, mutzel@ads.tuwien.ac.at

Abstract. We present an experimental study in which we compare the state-of-the-art methods for compacting orthogonal graph layouts. Given the shape of a planar orthogonal drawing, the task is to place the vertices and the bends on grid points so that the total area or the total edge length is minimised. We compare four constructive heuristics based on rectangular dissection and on turn-regularity, also in combination with two improvement heuristics based on longest paths and network flows, and an exact method which is able to compute provable optimal drawings of minimum total edge length.
We provide a performance evaluation in terms of quality and running time. The test data consists of two test-suites already used in previous experimental research. In order to get hard instances, we randomly generated an additional set of planar graphs.

1 Introduction

Orthogonal graph drawing is getting increasing attention from industry because of its numerous applications, *e.g.*, in database design, software engineering and many more. For many of these applications, the *topology-shape-metrics approach* leads to the best results. Here, a first phase (planarisation) determines the topology of the drawing. The aim is to generate a drawing with a small number of edge crossings, *e.g.*, by computing a large planar subgraph and carefully reinserting the removed edges. Then, the crossings are replaced by artificial vertices resulting in a planar (also called planarised) graph. A second phase determines the shape of the final layout. This phase is often restricted to planar orthogonal drawings (orthogonalisation). A widely accepted optimisation criterion is to minimise the number of bends without changing the topology. Finally, the third phase (compaction) deals with computing the metrics of the layout. Here, the optimisation problem is to compute a layout of minimum area or with minimum total edge length.

* This work is partially supported by the Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (No. 03–MU7MP1–4).

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 37–51, 2001.

Although Batini *et al.* [1,2] suggested the topology-shape-metrics approach already in 1984, it took some years until the approach was getting popular. The reason for this is twofold: On the one hand, expertise in planarity algorithms, combinatorial embeddings, planar graph drawing algorithms and combinatorial optimisation is necessary in order to deal with the upcoming combinatorial graph problems—most of them are *NP*-hard, including the compaction problem we are dealing with in this paper. On the other hand, it is time consuming to implement the approach, since many different kinds of algorithms are needed.

Recently, a lot of research has been done to solve many of the upcoming problems. Major improvements have been made concerning various aspects of the planarisation phase, *e.g.*, [3,4,5], the orthogonalisation phase, *e.g.*, [6,7,8,9], and the compaction phase [10,11]. Moreover, today there exist some software libraries containing the topology-shape-metrics approach [12,13,14,15]. Often it is implemented in a modular form, so that it is easy to experiment with.

This enables experimental comparisons between various algorithms in order to understand their influence on the final drawing. Already in [16], the impact of choosing two different algorithms for the orthogonalisation phase has been analysed and compared to two different orthogonal drawing methods. The experimental results showed that the topology-shape-metrics approach is superior to other orthogonal methods. Experimental comparisons for various hierarchical drawing methods appeared in [17,18] and for force-directed methods in [19].

In [11], we have suggested an exact method for compacting orthogonal drawings. Our implementation is able to solve instances of up to 1,000 vertices to provable optimality in short computation time. Independently, Bridgeman *et al.* came up with a new heuristics for the compaction problem in [10]. We were interested how the behaviour of the various compaction heuristics are related to the optimal solutions. Moreover, we wanted to find the best strategies among the heuristic methods. These questions led to the present computational study.

We compare the following constructive methods for orthogonal compaction: the original dissection method introduced in [20] based on longest paths, the same method based on network flows, two methods based on turn-regularity described in [10], and the exact method suggested in [11]. An orthogonal drawing can be improved by using iteratively one-dimensional compaction methods based on longest paths or network flows. We have tested all possible combinations of construction and improvement heuristics.

Section 2 introduces the orthogonal compaction problem formally. The compaction methods under evaluation are described in Section 3. We present computational experiments on various benchmark sets of graphs in Section 4. The paper closes with our conclusions in Section 5.

2 The Compaction Problem

In this section, we present a precise formulation of the compaction problem we are considering in this paper. We assume familiarity with planarity and basic graph theory.

Let $G = (V, E)$ be a 4-planar graph, *i.e.*, a planar graph whose maximum vertex degree does not exceed four. We associate with each undirected edge $(v, w) \in E$ two directed *half-edges* (v, w) and (w, v). A *planar representation* P for G describes the topology of a drawing for G in the plane by specifying the set of faces F as lists of counter-clockwise ordered half-edges and one explicit external face f_0.

An *orthogonal representation* H for G is an extension of P and describes, in addition to the topology, the *shape* of a drawing for G by specifying the bends in the edges and angles inside the faces. It is an equivalence class of planar orthogonal drawings. Two orthogonal drawings belong to the same class if one can be obtained from the other by rotating the drawing and modifying the lengths of the horizontal and vertical edge segments without changing the angles formed by them.

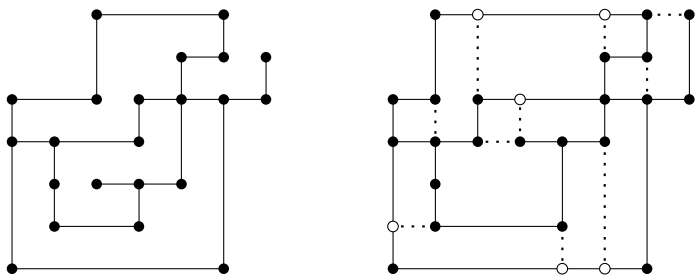
We call an orthogonal representation *simple* if its number of bends is zero. In the following, we always assume that an orthogonal representation is simple by treating bends as artificial vertices. A simple orthogonal representation H is defined by giving for each half-edge in P the angle it forms with its cyclic successor in the same face. A *planar orthogonal grid drawing* for a 4-planar graph maps vertices and bends to distinct grid points and edge segments to horizontal or vertical non-crossing non-empty line segments in the grid which connect the images of their endpoints. A drawing for a simple orthogonal representation H is a planar orthogonal drawing for the corresponding 4-planar graph which respects the shape coded in H. As with representations, we call orthogonal drawings simple if they do not contain bends.

Problem (Two-dimensional compaction problem in orthogonal graph drawing). *Given a simple orthogonal representation H for a 4-planar graph, find a simple planar orthogonal drawing for H of minimum total edge length.*

Patrignani shows in [21] that the compaction problem—and the related problems which ask for minimum area and minimum maximum edge length—are *NP*-complete.

3 Orthogonal Compaction Algorithms

In this section we briefly introduce the compaction algorithms under evaluation. We first describe the constructive techniques which produce a drawing for a given orthogonal representation H. The key idea behind these methods is to transform H into an auxiliary representation H' by introducing artificial edges and vertices and to find a drawing for H' in polynomial time. Removing the artificial vertices and edges in the auxiliary drawing leads to a drawing for H. Unfortunately, in the general case, these drawings can be far away from an optimal solution. We therefore present techniques which operate directly on drawings in order to improve the total edge length and area. Finally, we summarise an approach based on an integer linear program (*ILP*) to compute an optimal drawing for H.



(a) (b)

Fig. 1. The dissection method. (a) Original representation H, (b) Transformed representation H'. Dashed lines and empty vertices represent artificial edges and vertices

3.1 Constructive Heuristics

Tamassia mentions the first and still most common method to produce an auxiliary representation H' which is easier to deal with in his ground-breaking paper on bend-minimisation [20]. He introduces the *dissection method* which consists of decomposing each internal face of the given simple orthogonal representation H into a set of faces each of which has rectangular shape by introducing artificial vertices and edges. Figure 1 illustrates this method with an example—please note that the method works at the level of the representation; the coordinates have not yet been assigned. This process can be done in $O(n)$ time where n denotes the number of vertices in H. In the resulting orthogonal representation H', all interior faces have rectangular shape. Of course, the artificial vertices and edges impose additional constraints on the geometry which may lead to suboptimal total edge length and area in the resulting drawing.

Bridgeman *et al.* present in [10] another, more sophisticated, approach to produce a polynomial-time compactable auxiliary representation H'. Using the concept of *turn-regularity*, they manage to introduce a significantly lower number of artificial vertices and edges. Let f be a face in H. With each occurrence of a vertex v on the boundary of f, one or two *corners* are associated with v, depending on the angle internal to f between the edges preceding and following v. For each ordered pair of corners (c_i, c_j) associated with vertices of f, let $\rho(c_i, c_j)$ be the difference of the left and right turns along the boundary of f between c_i (included) and c_j (excluded). The value $\rho(c_i, c_j)$ defines the net angle between the edges preceding the vertices associated with c_i and c_j. Two corners at angles of at least 270 degrees are called *kitty corners* if $\rho(c_i, c_j) = 2$ or $\rho(c_j, c_i) = 2$. A face of an orthogonal representation is *turn-regular* if it has no kitty corners. Observe that rectangular faces are turn-regular. A representation is called *turn-regular*, if all its faces are turn-regular. We describe two methods from [10] to transform H into a turn-regular representation H'.

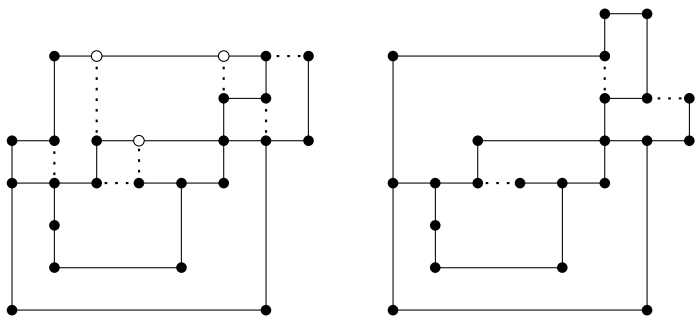


Fig. 2. The two turn-regularity-based dissection methods

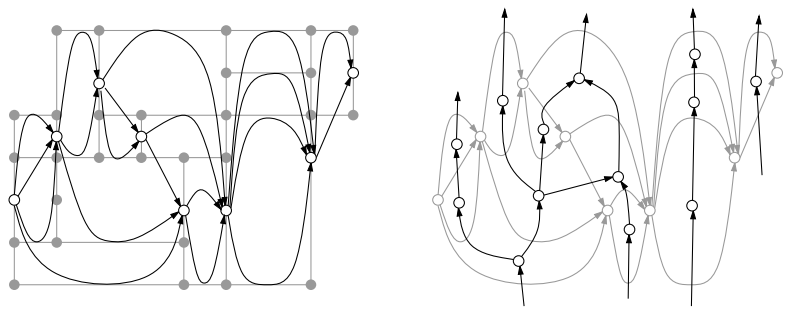
The first heuristics uses the rectangular dissection method described above, but only for faces in H which are not turn-regular (see Fig. 2(a)). The second heuristics recursively adds an artificial edge between each pair of kitty corners until the face has been decomposed into smaller turn-regular, but not necessarily rectangular, faces. The direction of the inserted edge (vertical or horizontal) is chosen randomly. See Fig. 2(b).

Two methods can be used to compute drawings for an orthogonal representation H' in which all internal faces have rectangular shape; variants of these methods generate a drawing if H' is general turn-regular. The methods, longest path-based compaction and flow-based compaction, are one-dimensional methods, *i.e.*, they assign horizontal and vertical coordinates separately—we therefore restrict our description to the computation of x-coordinates, the same techniques can be used for the y-coordinates.

We construct a directed graph D_x as follows: Each maximally connected vertical path in H' corresponds to a node in D_x—for a vertex v in H' we denote by $\text{vert}(v)$ the unique node in D_x it belongs to. For each horizontal edge $e = (v, w)$ in H' we assume that it is directed from left to right and insert an arc $(\text{vert}(v), \text{vert}(w))$ into D_x. Figure 3(a) shows this construction for the orthogonal representation from Fig. 1(b). Generally, we will refer to the pair of graphs in which nodes correspond to maximal horizontal and vertical paths and arcs to distance relations between these paths as *constraint graphs*.

Topologically sorting the nodes in D_x by computing longest paths and setting the x-coordinate of each vertex in H' to the topological number of $\text{vert}(v)$ leads to a feasible assignment of x-coordinates; y-coordinates result from a similar computation in the directed graph D_y. This approach, illustrated in Fig. 3(a), yields a drawing for H' with minimum width, height and area, but in general not minimum edge length. The running time is $O(n)$.

A more elaborate method which also minimises the total edge length for H' is the flow-based compaction. It assigns topological numbers for the nodes in D_x



(a) (b)

Fig. 3. (a) The graph D_x in the longest path-based compaction method. (b) The dual network D_x^* in the flow-based compaction method. Unconnected arcs are linked with an undisplayed node corresponding to the external face

by computing a minimum cost flow in its dual graph D_x^*. Flow through an arc a in D_x^* determines the length of the edge corresponding to the primal arc in D_x, thus a has a lower bound of one, infinite capacity and unit cost. See Fig. 3(b) for an example of the method which has a running time of $O(n^{7/4} \log n)$. If H' is turn-regular but not all faces have rectangular shape, the construction of the flow network is more complicated. In this case additional arcs which correspond to so-called *saturating* edges connecting switches in two oriented copies of H' have to be included in the flow network. For details see [10].

3.2 Improvement Heuristics

The constructive heuristics have a serious drawback: Though removing the artificial objects in the drawing for H' leads to a feasible drawing for H it is in general not the best one. Fortunately, variants of the longest path-based and flow-based compaction methods can be used to operate directly on the layout (this has also been used in VLSI-design; see, *e.g.*, [22]).

Unlike above, where the arcs in D_x correspond to edges in H', we insert arcs based on the visibility properties in the layout. If a maximally connected vertical path "sees" another one to its right, we insert an arc between the corresponding nodes in D_x. This will preserve the one-dimensional relative positions. Topologically sorting D_x or computing a flow in D_x^* results in new x-coordinates. Alternating the direction of the compaction and performing another step results in an iterative process. However, at each step the decisions are purely local, and compaction in one direction may prevent greater progress in the other direction. Furthermore, the layout may be blocked in both dimensions, but still be far away from an optimal solution.

3.3 Optimal Compaction

In [11], Klau and Mutzel present an ILP-based approach to solve instances of the two-dimensional compaction problem to optimality. It is based on a characterisation of the set of feasible solutions in terms of paths in the pair of constraint graphs. Given a pair of constraint graphs in which only the relative positions known from the shape of H are present (*shape graphs*), the compaction problem can be seen as optimising over the set of certain extensions of these graphs. The quality that such an extension must comply is based on geometric properties and establishes a direct link between the two graphs D_x and D_y. The new combinatorial task can be naturally formulated as an ILP which can be solved using a branch-and-cut algorithm. If there is only one possible extension of the given shape graphs, the authors show that their algorithm runs in polynomial time. Even if the algorithm does not find an optimal solution, it will report a feasible solution with a quality guarantee.

4 Computational Experiments

In this section we present a selection of our computational results for the comparison of compaction techniques. We concentrate on the—in our opinion—most interesting and usable experiments. We provide the full data at [23].

Experimental Settings. Each of the compaction strategies introduced in Sect. 3 is available as a module inside the AGD library (see [13]). Many heuristics rely on flow computations: In all modules we use the LEDA-function for this task. Our implementation of the ILP-based method uses CPLEX, an ABACUS-version is available. Here, we set a time limit of 15 minutes CPU time and return the upper and lower bound.

We test the implementations of the constructive heuristics from Sect. 3.1 both stand-alone and in combination with implementations of the two improvement heuristics from Sect. 3.2. Additionally, we test the implementation of the ILP-based algorithm. We run the implementations of the 13 resulting compaction techniques on a Sun Enterprise 450 with 1.1 GB main memory and two 400 MHz-CPUs and refer to them using the following scheme: We call the implementations of the constructive heuristics LP, FL, T1 and T2, corresponding to the rectangular dissection method with longest path compaction and with flow compaction and the two variants of turn-regularity-based dissection with flow compaction. If we apply an improvement heuristics, we append either LP or FL. We call the implementation of the ILP-based approach OPT. Figure 4 shows the output of some of the methods for an example.

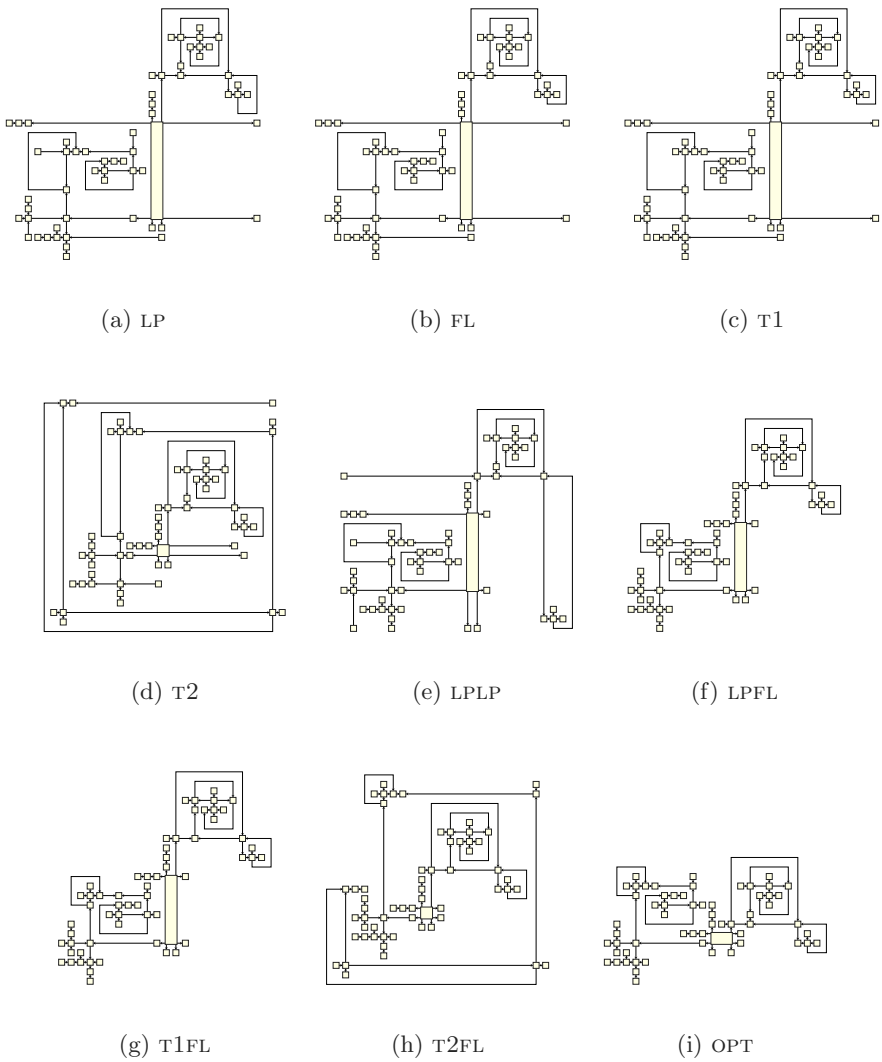
For our experiments, we use three different groups of graphs which can roughly be divided in easy instances, practical instances and hard instances. Graphs which are relatively easy to compact and which have already been used in [10] are 4-planar biconnected graphs. We use a set of 500 graphs with 10 to 100 vertices. The set of more than 11,000 practical instances has been introduced in [16] and has since then become a widely used test-suite in experimental graph drawing.

Additionally, we generated 540 hard instances for the compaction problem with a graph generator in the LEDA library: To generate a planar graph with n vertices, the generator chooses n segments whose endpoints have random coordinates of the form x/K, where K is the smallest power of two greater or equal to n, and x is a random integer in $[0, \ldots, K-1]$. It then constructs the arrangement defined by the segments and keeps the n nodes with the smallest x-coordinates. Finally, it adds edges to make the graph connected. We call this test-suite quasi-trees because large subgraphs of the resulting graphs are trees. In [11], quasi-trees have shown to be hard instances of the compaction problem; they have many fundamentally different drawings which makes the compaction task difficult. We transform each test graph G into an instance of the two-dimensional compaction problem in the following way:

1. We compute a planarised graph G' using the planarisation method in the AGD library. Note that the number of vertices in G' equals the number of vertices in G plus the number of crossings. We then compute a planar embedding of G'.
2. We run the transformation phase of the Giotto algorithm [2]. The phase creates an auxiliary 4-planar graph G'_4 by replacing vertices of degree greater than four by artificial faces.
3. We use a variant of Tamassia's bend minimising algorithm in which the underlying network differs from the original one in [20]: a minimum cost flow corresponds to a bend-minimum shape in which, due to a second optimisation goal, the number of 180-degree angles between edges is maximum. This avoids unnecessary staircase-like structures in the shape and makes the following compaction task easier. We replace bends by artificial vertices in the resulting orthogonal representation and get a simple orthogonal representation H'_4 which we use as the input for the compaction algorithms.

We choose this strategy, because it is among the orthogonal methods which yield the best layouts in practice. Note that the number of nodes in H'_4 is higher than the number of nodes in G. Especially for the non-planar practical graphs, where not only the bends but also the crossings count as vertices, this leads to an uneven distribution of graph sizes. Because the number of crossings has a strong influence on the behaviour of the compaction algorithms, we provide the number of crossings and details on the distribution of the input data at [23].

Total Edge Length. First we consider the set of 4-planar graphs. We divide the instances in subgroups according to their sizes with steps of 20 vertices. For each group we compute the average total edge length first for the four constructive heuristics (see Fig. 5(a)), for the four methods with longest path-based improvement (undisplayed) and for the methods with flow-based improvement (Fig. 5(b)). We display the results relative to the optimum edge length which is provided by OPT. Longest path-based post-compaction does not lead to significant improvements in terms of total edge length. This behaviour could also be observed with the practical graphs and the quasi-trees, we therefore will not display the data for improvement with the longest path method (see also Fig. 4(e)).



(a) LP (b) FL (c) T1

(d) T2 (e) LPLP (f) LPFL

(g) T1FL (h) T2FL (i) OPT

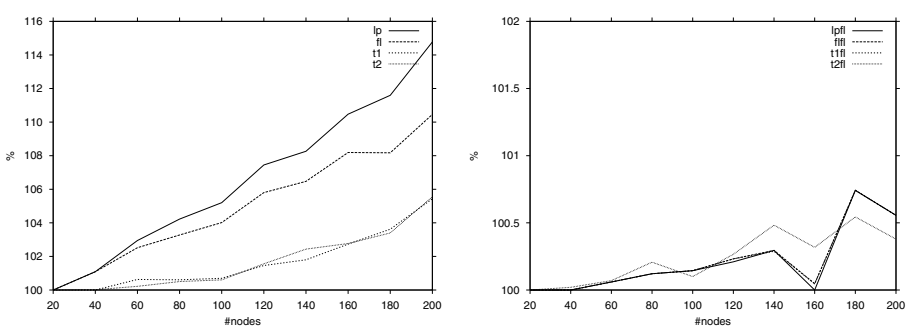
Fig. 4. `quasiTree.60.5.lgr` compacted by different methods

Obviously the biconnected graphs are easy to compact: Already the worst constructive method in terms of edge length, LP, achieves quite good results. Improving the drawings with the flow method often results in an optimal drawing. The plot also indicates that the methods LPFL, T1FL and FLFL result in very similar values—which is also true for the other test-suites.

We partition the more than 11,000 graphs corresponding to practical data in the same way as the biconnected graphs. Figure 6 shows the resulting total edge length for different methods relative to the optimum value computed by OPT. On the one hand, the practical graphs behave like the biconnected graphs: The heuristics are close to the optimum value and almost reach it when using improvement with flow. On the other hand, bigger instances are easier to compact, whereas the biconnected graphs indicate the opposite. This is due to the high number of crossings as produced by the prior planarisation step. The higher this number, the simpler the shapes of the faces. For the bigger graphs, we often observe that almost all faces have rectangular shape; this explains the good performance of all methods for big planarised graphs.

A different view of the improvement with the flow method shows its influence on the quality more drastically. In Fig. 7, we grouped the graphs according to their size using a step size of 50 vertices and showed for each constructive heuristics its value before and after the improvement step. Again, it can be observed that the big planarised graphs are almost optimally compacted by the heuristics. Additionally, the plot illustrates that the choice of the constructive heuristics in the first step does not have a big impact of the final quality: what matters is the improvement with the flow method.

The most challenging instances for the compaction problem are the quasi-trees. We first consider the smaller instances where OPT could find the optimal solution or at least a very good bound within the time limit. These are 75 graphs with 40 to 85 vertices in the original graph, resulting in 44 to 121 vertices in the simple orthogonal representation. Figure 8 illustrates the quality of the heuristics with respect to the optimum value. Again, the best heuristics perform very well: Even for these hard instances, three of them (T1FL, T2FL and FLFL) always stay below 1.1 times the optimum value, alternating at the top position among the



(a) Constructive heuristics (b) Improvement with flow

Fig. 5. 4-planar biconnected graphs: total edge length relative to optimal value

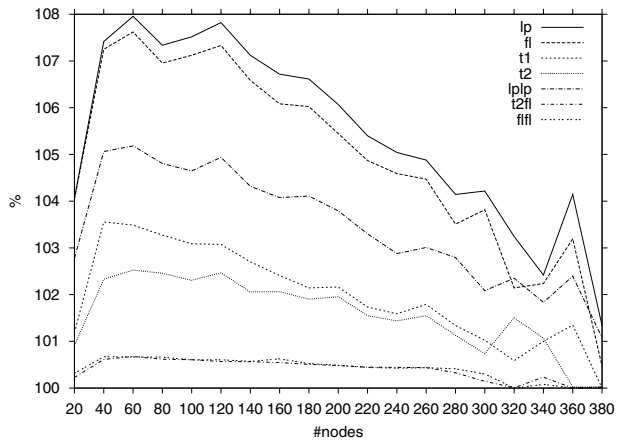


Fig. 6. Practical graphs: total edge length relative to optimal value

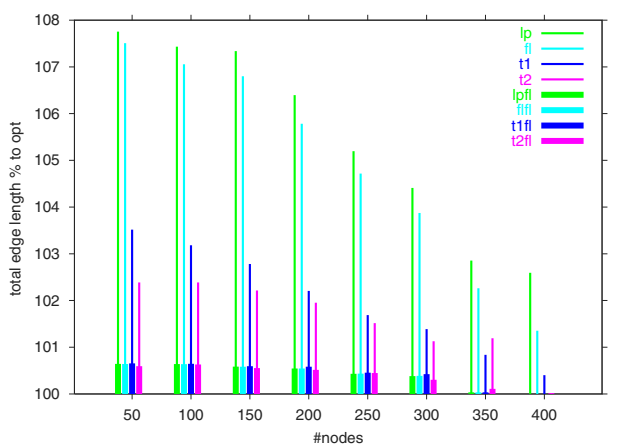


Fig. 7. Practical graphs: impact of the flow method

heuristic methods. Another observation is that the quality of the methods is relatively independent of the graph size.

We also look at the trend when the sizes of the quasi-trees grow and investigate the quality of the methods for 490 bigger instances in the range of 100 to 2,500 original vertices. Here, we choose T2FL as the comparison method, see Fig. 9. Again, T1FL, FLFL and LPFL are very close together and manage in some cases to beat the comparison method. In the plot, we display LPFL and the four constructive heuristics whose quality decreases as the instance sizes grow. It can be seen that the methods based on rectangular dissection perform similarly, as a stand-alone heuristics, T2 is the best. But using flow compaction as an improvement almost nullifies this advantage.

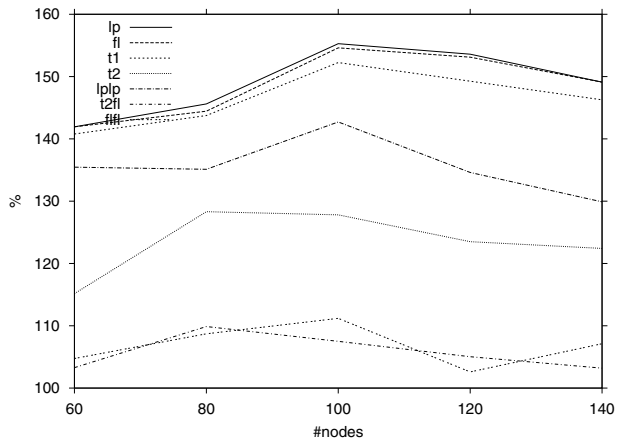


Fig. 8. Small quasi-trees: total edge length relative to optimal value

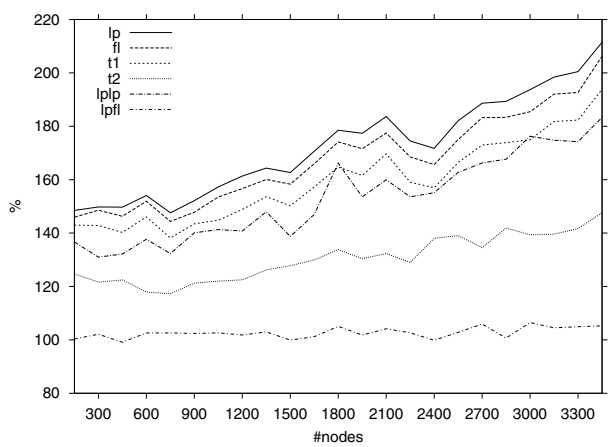


Fig. 9. Big quasi-trees: total edge length relative to T2FL

Running Time. All implementations run very fast on the biconnected and practical instances, where the constructive heuristics stay below .5 seconds and the improvement heuristics stay below .8 seconds on all instances. Even OPT which computes a provable optimal solution stays below four seconds on all instances. In the case of the hard compaction instances, most methods stay inside a five second time limit for graphs up to 1,000 vertices. On the large instances, the running time increases to up to 75 seconds for the flow improvement heuristics. Generally, there is a typical pattern for the performance of the methods which can be observed in all graph sets: Heuristics LP beats every other implementation in terms of running time. For the flow-based methods, FL has an advantage over T1 and T2, which are slowest among the constructive methods. This order remains the same when applying an improvement heuristics.

Other Criteria. In [10], Bridgeman *et al.* report a ratio of turn-regular faces of about 95% in their biconnected graphs suite. We have the same results with exception of the small quasi-trees, where only about 81% of the faces are regular.

We count the number of edges inserted during the dissection heuristics for LP, T1 and T2. For the biconnected and practical instances, the number of inserted edges for T1 is only about one tenth of the number for LP, and for T2, only about 3 percent of that number has to be inserted to achieve regular faces. In the case of harder instances, however, the number of edges to be inserted for T1 raises to roughly half the number of edges for LP and also the number for T2 raises relative to LP. Even for a small number of inserted edges, the choice which edge to insert in heuristics T2 can have a strong impact on the compaction result.

The maximal edge lengths and the area in the computed drawings behave similar to the total edge lengths for the given graph sets. For the biconnected and practical graphs, the average maximal edge lengths per subgroup lie close together. Among the constructive methods, T1 and T2 have a slight advantage over the other methods. As for the total edge length, the flow improvement has only a small impact on the results. The results for the quasi-trees, however, show a bigger variation. Heuristics T2 performs best among the constructive methods, but combination with the flow improvement can drastically reduce the lengths for all constructive methods. Here, too, the flow improvement dominates the quality of the result, the differences in the results of the initial methods vanish. The area and perimeter results show the same behaviour: In case of hard instances, flow improvement can help to enhance the quality and to close the gap created by the initial methods, otherwise the constructive heuristics lie close together and show only a slight improvement when flow is applied on their initial layout.

5 Conclusions

We have evaluated the state-of-the-art compaction techniques for orthogonal graph drawing in terms of quality and run time performance. We propose to divide the heuristic methods into constructive heuristics and improvement heuristics which yields many different compaction techniques. We have compared the results of the heuristics both against the optimal drawings and against each other. In our experiments, we have used three different test-suites to test the algorithms: easy, practical and hard instances and come to the following main conclusions:

Heuristics perform very well on most instances of the compaction problem.
Especially for the data from the easy and practical instances we could observe an excellent behaviour, both in terms of quality and running time. We want to emphasise, however, that in some cases it is desirable to get an optimal drawing, *e.g.*, when quality is more important than running time. Moreover, the implementation of the ILP-based algorithm is essential to compare the heuristics.

The choice of the constructive heuristics does not matter as long as flow compaction is used as post-processing.
Although the results for the constructive methods differ significantly, the diffe-

rences vanish when using flow-based compaction in an improvement step. We therefore propose to use a simple constructive method, *e.g.*, rectangular dissection and longest path-based coordinate assignment, followed by a flow-based post-processing step, which yields also the best running time among the good heuristics. The implementational effort for this variant is low as compared to the relatively complex turn-regularity-based techniques.

Graphs with many crossings are easy to compact.
In the topology-shape-metrics approach, crossings are treated as artificial vertices. The more crossings a drawing contains, the simpler are the shapes of its faces. For the big planarised graphs in the practical test-suite we observe that in many cases almost all faces have rectangular shape. Such graphs provide easy instances of the compaction problem.

Experimental studies help to improve the algorithms.
When we had the idea of doing a comparison between the different compaction techniques, we had in mind to simply provide the missing methods as modules inside our graph drawing library and run everything on a large number of graphs. This turned out to be more difficult than expected but at the same time very useful. During the more than 500,000 compaction runs we still found a considerable number of errors in our implementations which we considered stable in the beginning. This helped us to understand the two-dimensional compaction problem in graph drawing more deeply and we are now convinced that testing algorithms on a really large number of graphs—including pathological instances—helps a lot to provide good implementations.

References

1. C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data-flow diagrams. *IEEE Trans. Soft. Eng.*, SE-12(4):538–546, 1986.
2. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.
3. M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica, Special Issue on Graph Drawing*, 16(1):33–59, 1996.
4. P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In G. P. Cornuéjols, R. E. Burkard, and G. J. Woeginger, editors, *Integer Programming and Combinatorial Optimization (IPCO '99)*, volume 1610 of *LNCS*, pages 361–376. Springer-Verlag, 1999.
5. C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. Technical report, Technische Universität Wien, 2000. Submitted for publication.
6. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *LNCS*, pages 254–266. Springer–Verlag, 1996.
7. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. In *Proc. 5th Workshop Algorithms Data Struct. (WADS '97)*, volume 1272 of *LNCS*, pages 331–344, 1997.

8. M. Eiglsperger, U. Fößmeier, and M. Kaufmann. Orthogonal graph drawing with constraints. In *Proc. 11th Symposium on Discrete Algorithms (SODA '00)*. ACM-SIAM, 2000.
9. G. W. Klau and P. Mutzel. Quasi-orthogonal drawing of planar graphs. Technical Report MPI-I-98-1-013, Max–Planck–Institut für Informatik, Saarbrücken, 1998.
10. S. Bridgeman, G. Di Battista, W. Didimo, G. Liotta, R. Tamassia, and L. Vismara. Turn-regularity and optimal area drawings for orthogonal representations. *Computational Geometry Theory and Applications (CGTA)*. To appear.
11. G. W. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings. In G. P. Cornuéjols, R. E. Burkard, and G. J. Woeginger, editors, *Integer Programming and Combinatorial Optimization (IPCO '99)*, volume 1610 of *LNCS*, pages 304–319. Springer–Verlag, 1999.
12. C. Gutwenger, M. Jünger, G. W. Klau, and P. Mutzel. Graph drawing algorithm engineering with AGD. Technical report, Technische Universität Wien, 2000.
13. AGD. *AGD User Manual*. Max-Planck-Institut Saarbrücken, Universität Halle, Universität Köln, 1999. `http://www.mpi-sb.mpg.de/AGD`.
14. Graph drawing toolkit: An object-oriented library for handling and drawing graphs. `http://www.dia.uniroma3.it/~gdt`.
15. N. Gelfand and R. Tamassia. Algorithmic patterns for orthogonal graph drawing. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 138–152. Springer-Verlag, 1998.
16. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry: Theory and Applications*, 7:303–316, 1997.
17. G. Di Battista, A. Garg, and G. Liotta. An experimental comparison of three graph drawing algorithms. In *Proceedings of the 11th Annual Symposium on Computational Geometry (SoCG'95)*, pages 306–315, 1995.
18. M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In F. J. Brandenburg, editor, *Proceedings of the 3rd International Symposium on Graph Drawing (GD'95)*, volume 1027 of *LNCS*, pages 337–348. Springer–Verlag, 1995.
19. F. J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In F. J. Brandenburg, editor, *Proceedings of the 3rd International Symposium on Graph Drawing (GD '95)*, volume 1027 of *LNCS*, pages 76–87. Springer–Verlag, 1996.
20. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
21. M. Patrignani. On the complexity of orthogonal compaction. In F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Proc. 6th International Workshop on Algorithms and Data Structures (WADS '99)*, volume 1663 of *LNCS*, pages 56–61. Springer–Verlag, 1999.
22. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, New York, 1990.
23. G. W. Klau, K. Klein, and P. Mutzel. An experimental comparison of orthogonal compaction algorithms. Technical Report TR-186-1-00-03, Technische Universität Wien, 2000. Online version at
`http://www.ads.tuwien.ac.at/publications/TR/TR-186-1-00-03`.