An Experimental Evaluation of Multilevel Layout Methods

Gereon Bartel, Carsten Gutwenger, Karsten Klein, and Petra Mutzel

Technische Universität Dortmund, Germany
{carsten.gutwenger,karsten.klein,petra.mutzel}@cs.tu-dortmund.de

Abstract. Applying the multilevel paradigm to energy-based layout algorithms can improve both the quality of the resulting drawings as well as the running time of the layout computation. In order to do this, approaches for the different multilevel phases refinement, placement, layout, and optionally scaling and postprocessing need to be implemented. A number of multilevel layout algorithms have been proposed already, which differ in the way these phases are realized. We present an experimental study that investigates the influence of varying combinations with respect to running time and quality criteria.

1 Introduction

Energy-based layout methods are used in many different application areas such as social sciences and biology to automatically compute straight-line drawings of undirected graphs. They are often preferred over alternative methods because they are reasonably fast, allow straight-forward extensions for a large number of drawing constraints, and are relatively easy to implement. However, in practice they suffer from a number of drawbacks. First of all, they tend to converge to locally optimal solutions far away from the global optimum, e.g., when the graph is not completely unfolded. This is often also a result of poor parameter settings, since it is difficult and time-consuming to optimize the parameters with regard to a large number of input instances, and suitable parameters for a class of graphs may be disadvantageous for other classes. In addition, even though improvements in the implementation and continuous advances in hardware performance allow to compute layouts for medium to large graphs, running times are still too high for many large graphs from practical applications.

The multilevel approach helps to overcome these problems both by avoiding local minima and speeding up the computation due to improved convergence to the final drawing. Due to the step-by-step construction of the final layout starting with the layout of a small graph, there is also no dependency on a good initial drawing to start with. A number of different multilevel approaches have been proposed over the last years, but the influence of the different phases as well as the effects of their combination have not been investigated so far.

There is a wealth of publications concerning energy-based layout methods; see [3, 17] for an overview. An early comparison of such methods was presented

U. Brandes and S. Cornelsen (Eds.): GD 2010, LNCS 6502, pp. 80–91, 2011.

by Brandenburg et al. [4]. Walshaw [19] introduced the multi-level paradigm for graph drawing and Hachul and Jünger [14] presented an experimental study of layout algorithms for large graphs, including energy-based and algebraic approaches. Brandes and Pich [5] presented a study on distance-based graph drawing, and Frishman and Tal [8] and Godiyal et al. [11] investigated the use of the GPU for multi-level layout computation.

We present an experimental study that investigates the performance of a large number of different combinations for the multilevel phases. Our comparison includes a selection of well-established energy-based layout methods, as well as a number of fast methods specifically developed for multilevel approaches. We check if one of the combinations can be recommended as an overall choice or if a number of combinations need to be considered due to the characteristics of the input graphs. Our benchmark set comprises graphs already known from literature and a number of additional generated and real-world graphs.

After a short description of the multilevel paradigm for graph drawing in Sect. 2, we describe the setting and results of our experiments in Sect. 3 and draw a conclusion in Sect. 4.

2 The Multilevel Paradigm

Multilevel layout computation is an iterative process that can be roughly divided in three phases: *coarsening*, *placement*, and *single level layout*. Starting with the smallest graph, the final layout for the input graph is obtained by successively computing layouts for the graph sequence computed by the coarsening phase. At each level, the additional vertices need to be placed into the layout of the preceding level, optionally after a scaling to provide the necessary space.

Coarsening Phase. Given an input graph G, the coarsening phase builds a multilevel hierarchy by computing a sequence of increasingly smaller graphs $G_0, G_1, \ldots, G_k$ with $G = G_0$. In order to coarsen the graph, sets of vertices in G_i are combined to single vertices in G_{i+1}, where a number of different criteria can be used for deciding which vertices to combine. Subsequent merging or layout steps for the graphs at each level may take into account vertex weights that represent the merging of vertex sets to a single vertex.

The influence of the coarsening method is twofold: On the one hand, the way the graphs are coarsened can have an impact on both the quality of the drawing and the running time on each level. This depends on how well the overall graph structure is represented on the levels, influencing the way the graph is unfolded. On the other hand the number of hierarchy levels may have a significant influence on the total running time.

Placement Phase. The placement phase is responsible for adding vertices to the layout when the algorithm proceeds to the next level in the multilevel hierarchy. Typically, new vertices are placed close to their representative in the previous level. How much the layout computation on each level can profit from the layout given by the previous level, and how a special placement can improve the

computation, therefore is influenced both by the way the vertices are merged and the number of vertices that are merged to a single representative. Clever placement may drastically reduce the work needed by the single level layout and also have an impact on the way the graph is unfolded. Instead of using a static position assignment, Frishman and Tal [8], Gajer et al. [10], and Godiyal et al. [11] describe an iterative method with a small number of iterations.

Single Level Layout. On each level i in the multilevel hierarchy, a layout for the corresponding graph G_i has to be computed. The main requirement for the layout method used is that it has to accept the layout resulting from the previous placement phase as initial layout such that the layout is an extended and refined version of the layout of G_{i+1}. In order to allow a running time improvement by the multilevel method, the single level run should either work with a reduced number of iterations or be dependent on a stop criterion, e.g., when the layout energy drops under a predefined threshold.

3 Experimental Study

We use a benchmark set that comprises 43 graphs of varying size and characteristics, including both real world and generated instances[1]. In addition to a selection of graphs used by Hachul and Jünger [14], we use a further real world graphs, comprising a protein interaction network, the RNA network from the InterViewer project [15], a selection of Walshaw's graph archive [18] and the AT&T graph library [2], as well as a number of generated grids. We added a rectangular grid with 400 rows and 20 columns in order to test for distortions, and two smaller squared grids, also in variants where corners of the grids are connected to make the instances more difficult to unfold.

We compare results from multilevel layout computations and also single level runs of the implemented layout methods in order to evaluate the gain from the corresponding multilevel runs. In single level runs, all layout algorithms are called using standard parameters, which are either obtained by empirical evaluation or from the original publications. We adapt these settings for use in the multilevel framework based on experiments on a small "tuning" subset to investigate reasonable ranges for parameter settings. We did this mainly by visual inspection of the resulting layouts.

For our study we ran the full set of method combinations for the three phases on the full benchmark set and mainly evaluated the results of the statistical analysis of the layout characteristics. We still did sample visual inspections, both to check that our analysis is reasonable and to judge the quality when the statistical values are ambiguous. In order to capture the quality of the drawings, we analyze a couple of aesthetic criteria, including standard criteria as number of edge crossings and drawing area, and also typical optimization goals of energy-based layout methods like edge length deviation and node distribution.

[1] More information: `http://ls11-www.cs.tu-dortmund.de/staff/klein/gdmult10`

3.1 Implementations

We implemented a multilevel framework within OGDF [1] that allows to freely combine different strategies for the multilevel phases. Therefore it is not possible to do a fine tuning for combinations to speed up the computation, and the runtime of our approach may be slightly inferior compared to particular implementations of a single multilevel approach.

We used the energy-based layout algorithms provided by OGDF and implemented various. coarsening and placement strategies (see below). In addition to these main phases, the framework provides scaling and local postprocessing at each drawing level. Although these steps may influence running time as well as layout quality, we focus on the three main phases here and use the same minimum scaling and postprocessing options for all combinations.

Coarsening methods. All coarsening methods we use take an approach, where vertices on a level G_i are merged such that a single representative of a group of vertices survives on the next level G_{i+1}.

(NM). The *Null Merger* does not merge any vertices and is used to simply realize a single-level layout within the multilevel framework.

(RM). The *Random Merger* simply selects vertices randomly and merges them with a random neighbor, until the size of the graph decreases by a predefined factor. This method allows a good control over the number of levels in the hierarchy and is used to have a minimal quality threshold that more sophisticated methods should outperform.

(MM). The *Matching Merger* corresponds to the method described in [19], where a matching is computed by visiting the vertices in a random order, matching each unmatched vertex with a random unmatched neighbor if existent. The number of vertices in G_{i+1} is then at least half the number of the vertices in G_i.

(WMM). The *Weighted Matching Merger* is the weighted variant of MM described in [19]. In order to achieve a uniform merging, the matching vertex is chosen to be the neighbor with the smallest weight, where the weight of a vertex v is the number of vertices on lower levels represented by v.

(ECM). The *Edge Cover Merger* is a variant of MM intended to eliminate the problem that for certain graphs containing star-like subgraphs a linear number of levels may be necessary. In addition to the matching, an edge cover is computed such that each contained edge is incident to at least one unmatched vertex. These cover edges are then used to merge their end vertices.

(LBM). The *Local Biconnected Merger* is a variant of ECM that tries to avoid situations where distortions are introduced, since—during the coarsening process—vertices are merged to a cut vertex at which the layout method may twist the orientation. LBM checks if biconnectivity may be lost in the local neighborhood around the potential merging position.

(SM). The *Solar Merger* corresponds to the method described in [13], where the vertices are partitioned into *solar systems*, classifying each vertex as either sun, planet or moon. The resulting solar systems are then collapsed to the sun vertex.

(ISM). The *Independent Set Merger* corresponds to the strategy used within the GRIP [10] approach. It uses a maximal independent set filtration $\mathcal{V} : V = V_0 \supset V_1 \supset \ldots \supset V_k$ with $k = O(\log n)$, such that V_i are the nodes on level i. V_1 is a maximal independent set of G and V_i is a maximal subset of V_{i-1} such that the graph theoretical distance between any pair of its elements is at least $2^{i-1}+1$. Gajer and Kobourov sped-up the computation using partial BFS-trees, which achieves subquadratic runtime behavior in practice.

Placement methods. We have implemented the following placement methods, most of which were already described in the literature.

(ZP). The *Zero Placer* uses the simplest strategy of all methods under investigation. A vertex is placed at the same position as its representative in the previous level. A small random offset avoids a zero distance between two nodes which may pose problems for some layout methods.

(RP). The *Random Placer* places vertices at random positions within the smallest circle containing all vertices around the barycenter of the current drawing. Clearly, this placement strategy may hinder convergence and lead to distortions in the drawing. It is introduced to give a minimum quality bound that more sophisticated strategies should outperform.

(BP). A well-known method is the *Barycenter Placer*, which places a vertex at the barycenter of its neighbors' positions. The influence of the neighbor position might be weighted, e.g., by the merging weight of the vertices. The barycenter placement should help the energy-based layout algorithm to reach an energy-minimal state because it at least partially considers the influence of the neighbor positions on the movement force for the placed node.

(MP). As an alternative to `BP`, the *Median Placer* uses the median position of the neighbor nodes for each coordinate axis. This smoothes the effect of neighbors at outlier positions but still takes into account the strong influence of the neighbor positions within the layout methods.

(SP). Developed for use within the FMMM approach [13], the *Solar Placer* is able to use information from the merging phase of the solar merger. A new vertex is placed on the direct line between two suns at the relative position with respect to its position on the intersystem path between these suns. If it lies on more than one intersystem path, the average value of all relevant positions is used. In combination with other mergers, `SP` resembles `ZP`.

Layout methods. We investigate the performance of energy-based layout methods that where either implemented as single level methods in OGDF or described as parts of multilevel approaches in the literature, as, e.g., the new multipole method by Hachul. In addition, we implemented two variations which try to exploit the specific multilevel setting. Due to the runtime of the Kamada-Kawai approach, we test a combination of Kamada-Kawai (`KK`) and the two faster methods Fruchterman-Reingold (`FR`) and Fast Multipole Embedder (`FME`), where Kamada-Kawai is used for all levels but the last, where `FR` or `FME` is applied instead. For

the even slower Davidson-Harel method, we compute the last layout using the very fast New Multipole Method (NMM). Clearly, these combinations will only pay off when the increase in the number of vertices at the last level is relatively large. The following layout methods are used within our study:

(EAD). This method uses the spring model proposed by Eades [7] which represents the classical force directed approach.

(FR). The model proposed by Fruchterman and Reingold [9] is based on the force-directed method, and uses a cooling scheme to limit the total displacement. We used two variations: (FRE) uses the node weighting scheme proposed by Walshaw [19] and (FR) is the original grid variant [9].

(KK). The force-directed method by *Kamada and Kawai* [16] that constitutes an MDS approach, using the graph theoretic distance between vertices as distance.

(NMM). The *New Multipole Method* introduced by Hachul and Jünger [13] that uses a fast multipole approximation of the repulsive forces with $O(N \log N)$ running time leading to an $O(|V| \log(|V| + |E|))$ overall running time.

(FME). The *Fast Multipole Embedder* method by Gronemann [12] uses the same repulsive forces as NMM, but slightly modified attractive forces. The repulsive forces are approximated with the fast multipole method using a reduced quadtree construction and well-separated pair decomposition.

(DH). The layout algorithm introduced by *Davidson and Harel* [6], based on simulated annealing; with a running time of $O(|V|^2|E|)$ by far the slowest algorithm in our comparison.

3.2 Computational Results

We ran the selected combinations and single level layout algorithms on all test graphs, except for Kamada-Kawai and Davidson-Harel, which were only run on graphs with up to 2000 (DH), 6000 (KK), 12000 (KKFR), and 15000 (KKFME) nodes, respectively, due to their space and running time requirements. We used a system with eight quad-core AMD Opteron processors and 16GB memory.

Due to the huge amount of experimental data, we provide detailed results for the most relevant methods and criteria only.

Discussion. We had a number of hypotheses regarding the performance of the methods under investigation when we conducted the experiments. First of all, we expected that combinations from established multilevel approaches should outperform the other combinations at least to a certain extent. Also the methods whose running time includes a significant layout independent computation, like for example KK that comprises an all-pairs shortest paths computation, may not benefit largely by the multilevel approach, whereas the methods FME and NMM especially developed for this scenario should dominate the others.

Our experiments showed that a significant part of our benchmark set did not pose a challenge to the layout algorithms, and that even single level calls were competitive both in speed and quality at least for graphs up to a certain size; cf.

Fig. 2. Especially the NMM method, actually developed for use within a multilevel approach, could often deliver good results in single level runs. However, FME was not able to compute layouts of similar quality with a single level.

From the visual inspection we found that the layouts computed by KK are often the most pleasing, especially for the smaller graphs, but the gain compared to NMM and EAD was not large enough to justify the additional time. E.g., for *add32* KK computes a radial-like layout emphasizing the special structure of the graph better than the best layouts w.r.t. number of crossings and area (see Fig. 3), but also needed 255s compared to 7.7s and 6.1s for FME and NMM, respectively.

(a) SP+ISM+KK (b) MP+ISM+FME (c) BP+LBM+KKFME

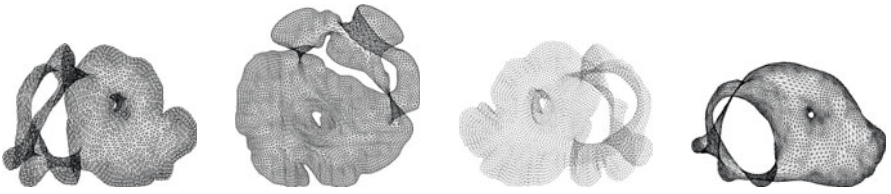
Fig. 1. Drawings of the *Grid_400_20* and *uk* graphs, which posed a challenge to many combinations regarding the distortions

Grid and mesh-like structures were mostly an easy task for all algorithms, with the exception of the narrow grid, which often suffered from folding in multilevel runs. For some of the layout methods, a specific effect can be observed when the narrow grid is laid out. The graph then is not drawn as a straight line but seems to be compressed; see Fig. 1(b). We found that this is not the result of bad unfolding, but due to the influence of the repulsion forces when vertices from the next hierarchy level are reinserted—the placement close to their parent vertex leads to large repulsion forces that distort the graph layout in the next layout step. The planar graphs *grid_400_20* and *uk* were problematic for nearly all combinations, since they were subject to a large number of distortions; see Fig. 1(a) and 1(c). LBM achieves slightly less crossings on average for these graphs, at the cost of additional running time due to more levels.

Graphs like the flower and spider graphs did a better job in separating single- and multilevel runs. Here it often came to crossings and foldings, and the single level methods typically did not obtain an acceptable layout. LBM only computed a single level for graph *flower_5* and therefore also had inacceptable results.

The interpretation of the results regarding the placement methods did not show big differences. We mainly introduced the random placer to better judge the overall quality of the other methods. Nonetheless it produced good results in most cases, but also some worst-case drawings with a huge number of crossings; therefore it cannot be recommended. Most other placers achieved comparable results without a clear winner; combining SP with SM was not better than any combination with other placers.

For the merger methods, we had a look at the resulting number of levels; see Fig. 4. Whereas RM, ECM, and ISM produced small and comparable numbers (less



(a) BP,MM,KKFR (b) MP,SM,FME (c) KKS (d) EADS

Fig. 2. Layout of the planar graph *3elt* computed with slowest (BP, MM, KKFR, 15 levels, 91s) and fastest overall combination (MP,SM,FME, 5 levels, 2.27s), compared to results of fastest (EADS, 2.77s) and slowest (KKS, 158s) single level methods

(a) KK (b) NMM (c) FME

Fig. 3. Layout of graph *add32* computed with NMM and KK, both with solar placer and merger, and the layout with fewest crossings computed with MP, LBM and FME

than 15), LBM produced slightly more levels, with the exception of *sierpinski_08* and *Grid_400_20* with about 30 levels, graph *flower_005* with only a single level. SM was able to stay below 10 levels in all cases, with a minimum of 3 levels.

The matching mergers MM and WMM produced by far the most levels, especially for the tree graphs, with a maximum of 81 and 76 for *tree_06_05*, and 33 and 31 for the smaller *snowflake_A*. For the graphs with a large star subgraph (*dg_1087* and *snowflake_C*), the number of levels (6565 and 2505) was prohibitively large, making layout computation infeasible.

As expected, there is a clear correlation between the number of levels created by the coarsening phase and the resulting running time. Depending on the graph characteristics, the coarsening may introduce a huge number of levels linear in the number of vertices in the graph. Even though the running time on each of the levels should be much smaller than a single level run (due to the smaller number of iterations and the earlier convergence), this may lead to a tremendous increase in running time without a corresponding gain in layout quality.

Running Time. We first had a look at the overall average running time with respect to the layout methods. Besides DH, which is extremely slow and only

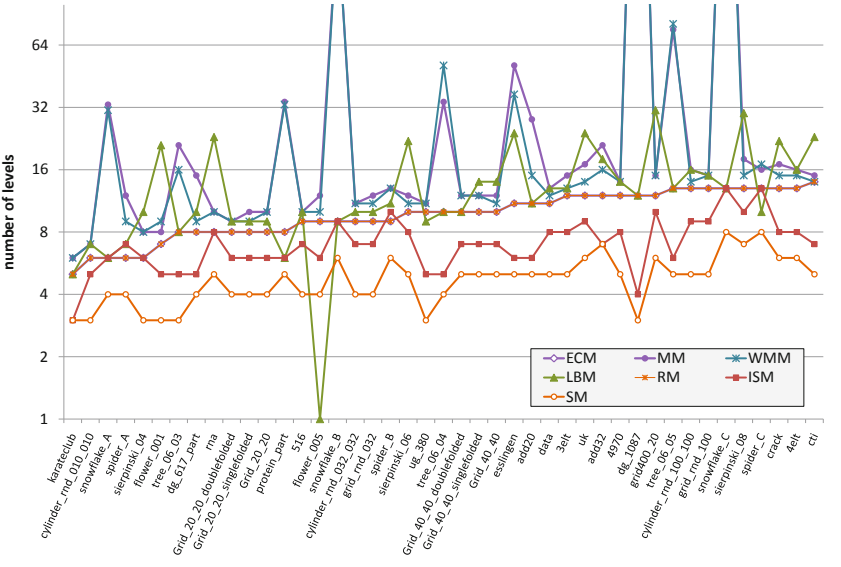


Fig. 4. Number of levels computed by the different merger methods

acceptable for the smallest graphs, we can identify three groups of layout methods: `FME`, `NMM` and `EAD` were always among the fastest layouts, dominating the `FR` and `KK` variants, which are an order of magnitude faster than `KK` alone. Above a graph size of about a few hundred vertices, the single level methods were clearly inferior to the multilevel combinations, where `FME` was slightly faster than `NMM` and `EAD`. We also compared the average running time of the mergers in combination with `EAD`, relative to `SM` (see Fig. 5); `LBM` and the matching mergers show a number of peaks while the other mergers show a balanced behavior and are always slower than `SM`.

Edge length. Our observation that `KK` often produced the most pleasing layouts corresponds well with the values of the normalized standard deviations of the edge lengths, where `KK` is clearly the best with nearly all values below 0.5, and for the fast methods `NMM` has a slight advantage over `FME` and `EAD`; see Fig. 6.

Crossings. We found that the number of crossings was a very good indicator of the layout quality, as layouts with unfolding problems often had a significant larger number of crossings. An exception are graphs which required an extreme number of crossings, e.g. because they contain complete subgraphs as the flower graphs. We observed that the random placer sometimes caused a drastic increase in crossings, especially in combination with the `FME`. We only report here the number of crossings for a selection of mergers with layout method `EAD`; see Fig. 7. For the large graphs with a large number of crossings on average, the difference

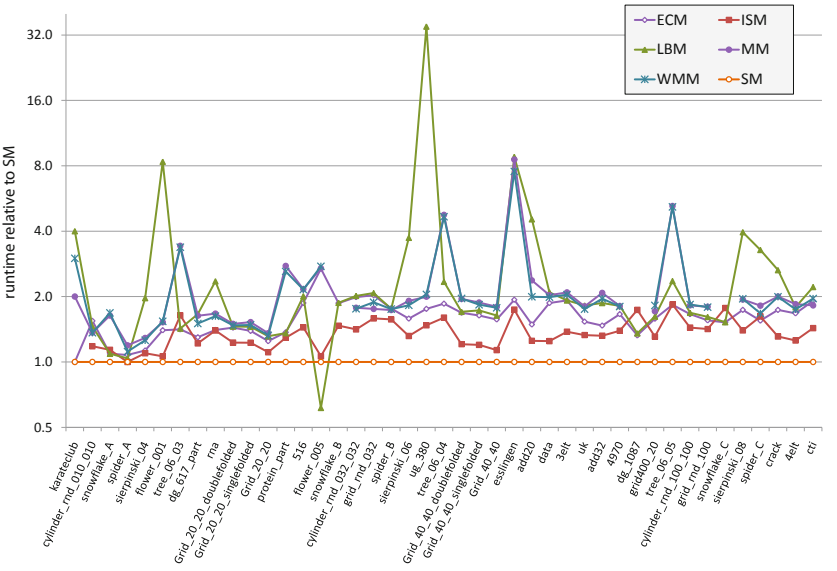


Fig. 5. Average running times of different mergers with `EAD` relative to `SM`

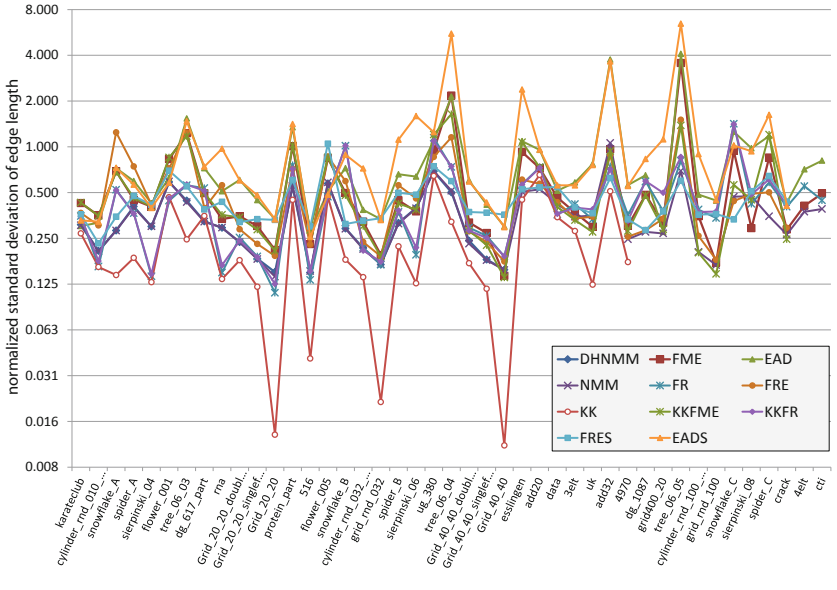


Fig. 6. Normalized standard deviations of the edge lengths for a selection of layout methods

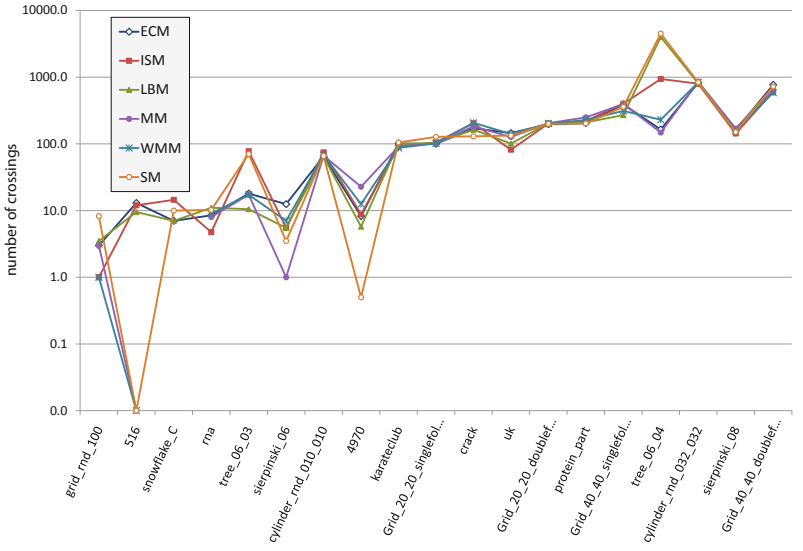


Fig. 7. Details of average number of crossings produced by different mergers with `EAD`

between the mergers is minimal, whereas for smaller crossing numbers there is no clear winner. `SM` and also `LBM` often produced quite good results in comparison.

Drawing Area. The drawing area alone is not a good quality indicator for straight-line drawings, as layouts with a huge number of crossings and overlaps may require a smaller area than drawings that reveal the graph structure well. We therefore do not report on the drawing area here.

4 Conclusions

We presented an experimental study of multilevel layout methods within a framework implemented in OGDF. Though there is no clear winning combination, a number of layout methods, mergers and placers showed a good behavior. The `EAD` layout algorithm, using the classical force model introduced by Eades worked very well within the multilevel approach and lead to similar results, both in quality and running time, as the models in `NMM` and `FME` specifically designed for multilevel computations. Mergers `ECM`, `SM` and `ISM` often produced good results, whereas the others showed bad worst-case behavior. It would be interesting to investigate how the running time is influenced by the different merging due to different graph characteristics, i.e. instead of just looking at the number of levels, to also analyze the effect on the convergence on each level. In addition, instead of looking at single criteria like edge crossings, the combination of different criteria might give a better estimation of the layout quality.

References

[1] The Open Graph Drawing Framework, http://www.ogdf.net
[2] The AT&T graph library, http://www.graphdrawing.org.
[3] Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice-Hall, Englewood Cliffs (1999)
[4] Brandenburg, F.J., Himsolt, M., Rohrer, C.: An experimental comparison of force-directed and randomized graph drawing algorithms. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 76–87. Springer, Heidelberg (1996)
[5] Brandes, U., Pich, C.: An experimental study on distance-based graph drawing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 218–229. Springer, Heidelberg (2009)
[6] Davidson, R., Harel, D.: Drawing graphs nicely using simulated annealing. ACM Trans. Graph. 15(4), 301–331 (1996)
[7] Eades, P.: A heuristic for graph drawing. Congressus Numerantium 42, 149–160 (1984)
[8] Frishman, Y., Tal, A.: Multi-level graph layout on the GPU. IEEE Transactions on Visualization and Computer Graphics 13(6), 1310–1319 (2007)
[9] Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. 21(11), 1129–1164 (1991)
[10] Gajer, P., Kobourov, S.G.: GRIP: Graph drawing with intelligent placement. J. Graph Algorithms Appl. 6(3), 203–224 (2002)
[11] Godiyal, A., Hoberock, J., Garland, M., Hart, J.C.: Rapid multipole graph drawing on the GPU. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 90–101. Springer, Heidelberg (2009)
[12] Gronemann, M.: Engineering the fast-multipole-multilevel method for multicore and SIMD architectures. Master's thesis, Technische Universität Dortmund (2009)
[13] Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 285–295. Springer, Heidelberg (2005)
[14] Hachul, S., Jünger, M.: Large-graph layout algorithms at work: An experimental study. J. Graph Algorithms Appl. 11(2), 345–369 (2007)
[15] Han, K., Ju, B.-H., Park, J.H.: InterViewer: Dynamic visualization of protein-protein interactions, http://interviewer.inha.ac.kr/
[16] Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31(1), 7–15 (1989)
[17] Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
[18] Walshaw, C.: The graph partitioning archive, http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/
[19] Walshaw, C.: A multilevel algorithm for force-directed graph-drawing. J. Graph Algorithms Appl. 7(3), 253–285 (2003)