An Experimental Comparison of Force-Directed and Randomized Graph Drawing Algorithms

Franz J. Brandenburg*, Michael Himsolt* and Christoph Rohrer

University of Passau, 94030 Passau, Germany
{brandenb, himsolt}@informatik.uni-passau.de

Abstract. We report on our experiments with five graph drawing algorithms for general undirected graphs. These are the algorithms FR introduced by Fruchterman and Reingold [5], KK by Kamada and Kawai [11], DH by Davidson and Harel [1], Tu by Tunkelang [13] and GEM by Frick, Ludwig and Mehldau [6]. Implementations of these algorithms have been integrated into our GraphEd system [9]. We have tested these algorithms on a wide collection of examples and with different settings of parameters. Our examples are from original papers and by our own. The obtained drawings are evaluated both empirically and by GraphEd's evaluation toolkit. As a conclusion we can confirm the reported good behaviour of the algorithms. Combining time and quality we recommend to use GEM or KK first, then FR and Tu and finally DH.

1 Introduction

Graph drawing has become an important area of research in Computer Science. There is a wide range of applications including data structures, data bases, software engineering, VLSI technology, electrical engineering, production planning, chemistry and biology. Simply speaking, graph drawing is concerned with the problem of obtaining aesthetically pleasing drawings of graphs. However, what means aesthetically pleasing? A list of criteria has been laid down, including uniformity of the edge length and the node distribution, crossings and the display of symmetries. Recent developments have brought a number of powerful and sophisticated algorithms, which attempt to cope with these aesthetic criteria. An excellent survey and classification can be found in [2].

In this study we consider five well-known algorithms for straight-line drawings of general undirected graphs. Our implementations are called FR, KK, DH, Tu and GEM, respectively. They are based on the algorithms introduced by Fruchterman and Reingold [5], Kamada and Kawai [11], Davidson and Harel [1], Tunkelang [13] and by Frick, Ludwig and Meldau [6]. The algorithms take arbitrary undirected graphs as input and produce straight-line drawings. They modify a given drawing iteratively and attempt to minimize some cost function or

* This research is partially supported by the Deutsche Forschungsgemeinschaft, Grant Br 835/6-1, Forschungsschwerpunkt "effiziente Algorithmen für diskrete Probleme und ihre Anwendungen"

the energy of the drawing. Their strategies are different, using the spring embedder or cost functions and randomization and the simulated annealing paradigms. Each algorithm has been described in detail in an original research paper, where examples of its performance and some comparisons can be found.

However, it is difficult to recall and check these results. This is partially due to the random character of the algorithms. Also, it is the concrete implementation that is in use. The implementation has a big influence particularly on the run time. The drawings depend on the settings of parameters and to a less extend on the initial drawings.

We have integrated FR, KK, DH, Tu and GEM into the GraphEd system, which is a platform for our experiments and evaluations, and we have run the programs on a wide collection of examples. This report includes only some graphs which have been named elsewhere. Further data can be found in [12]. The drawings are evaluated both empirically and by GraphEd's evaluation toolkit [10]. Our experiments confirm the good behaviour of the algorithms reported in the literature. They reach the intended goals, in particular uniform distributions of the edge lengths and the nodes. For many graphs their symmetries are well displayed.

Related experimental work on graph drawing has been presented by Davidson and Harel [1] and Fruchterman and Reingold [5], which mutually compare their drawings, by Frick et al. [6] comparing their GEM with GraphEd's FR and an earlier implementation of KK, Tunkelang [13], Himsolt [10] and di Battista et al. [3]. Himsolt considers a broad collection of graph drawing algorithms with different graph drawing standards and di Battista et al. test and compare three algorithms producing rectlinear drawings. Our spirit is similar with a focus on algorithms for straight line drawings of general graphs.

2 Algorithms

In this section we give an outline of the algorithms and explain particularities of our implementations. We followed the original descriptions unless otherwise stated.

2.1 FR

FR is the GraphEd implementation of the algorithm by Fruchterman and Reingold [5]. Based on the force-directed method, FR computes attractive and repulsive forces and simultaneously moves all nodes according to these forces, where the moved distance is bounded by a temperature t. This process is iterated for some rounds. The free parameters of the implemented algorithm are the optimal node distance, the maximal number of iterations and the temperature t. These parameters can be chosen by the user. The algorithm terminates if either the maximal force acting at a node falls below a user defined threshold or if the maximal number of iterations is exceeded.

2.2 KK

KK is the algorithm by Kamada and Kawai [11]. It computes the total energy of the drawing from the actual and the graph theoretic distances between nodes. Solving a system of partial differential equations by the Newton-Raphson method, the locally best node is moved to reduce the total energy of the drawing. This is repeated until the energy falls below a preset threshold.

C. Rohrer has tuned KK for the GraphEd system by a more efficient implementation of the Newton-Raphson method. This leads to a significant speed-up over an earlier version of KK, which was under study by Frick et al. [6] and Himsolt [10]. In the original description of the algorithm the number of iterations depends on the threshold and the structure of the drawing and thus is hard to predict. Instead, our implementation uses 10*n iterations, where n is the size of the graph.

2.3 DH

DH is the simulated annealing approach by Davidson and Harel [1]. The goal is to minimize a cost function $f(G) = \sum \lambda_i f_i(G)$, which sums over (1) the node distribution, (2) the edge length, (3) the edge crossings, (4) the node-edge distances and (5) the borderlines. Each of these components has its individual weight λ_i, which can be set by the user. Our implementation has a few individual features. They have been introduced to increase the flexibility of our DH and to save some running time, a critical factor for DH, as already stated in [1]. First, we drop the borderlines. Our concern are graphs that are (bi-) connected. Secondly, we normalize the weight parameters λ_i. For a given drawing, the related costs vary by orders of magnitude. E.g., the edge length sums over the quadratic distance between its nodes, the node distribution sums over the inverse quadratic distances and crossings are counted as integers. Therefore these costs are first normalized to 10, and then the user can set relative weights $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ with $1 \leq \lambda_1, \lambda_2 \leq 10$, $0 \leq \lambda_3 \leq 1000$ and $0 \leq \lambda_4 \leq 10$. The larger range for λ_3 is chosen to force planarity. If $\lambda_3 = 0$ or $\lambda_4 = 0$, then the computations for the detection of possible edge crossings and for the node-edge distances are suppressed.

For the motion of a vertex DH has two options. The first one follows [1] and attempts to move a randomly chosen node to some randomly chosen point on a circle around v with radius r. Starting with an initially high value the radius r decreases geometrically in each round. However, with a decreasing radius, the neighbourhood shrinks and the validity of the underlying theory is in question. Note that DH is an approximation to the simulated annealing process. In the second option, for each round, the radius r is randomly generated and for the trials of node movements the offsets are chosen randomly within a circle of radius r. Furthermore, we have added a re-enforcement heuristic, which again can be switched on/off. This concept is adopted from GEM [6]. If the last movement of a node has lead to an improvement of the cost function, its next update is restricted to almost the same direction, i.e. a point in the sector of width $\pi/2$. As in [1] the DH algorithm finishes with a fine-tuning phase, doing a linear number of moves where only improvements of the cost functions are allowed.

2.4 Tu

Tu is our implementation of Tunkelang's incremental algorithm [13]. Tunkelang uses a template of 16 locations. These are the 8 local neighbour positions and 8 positions at distance d resp. $d\sqrt{2}$, where d is the so-called quality parameter set by the user. $d = 4$ is the default value. Tu inserts the nodes one after another in some precomputed order, here breadth first from the graph theoretic center. For a new node, Tu checks the template positions of each of its neigbours and of the corners of the screen as a candidate position and chooses the locally best. After each insertion, there is a recursive finetuning. All neighbours of the current node are checked for an improved position, using the template from above for the candidate positons. The cost function is that of DH.

2.5 GEM

The graph embedder GEM has been introduced by Frick et al. [6]. It is a tuned and randomized version of a spring embedder, and combines ideas from FR and DH. GEM has been explained in detail in [6], where it is compared with FR and an earlier version of KK. A. Ludwig has made the implementation of GEM available to us.

2.6 Comparison and Measurements

We can present only an excerpt of our tests and examples. There are too many graphs and varieties of parameter settings.

For our measurements we consider the run time on a Sparc 10, the ratio of the length of the longest and the shortest edges, the normalized standard deviation of the edge length and the number of edge crossings. Furthermore, the distribution of the nodes, the ratio of the farthest and nearest pair of nodes, the number of edge crossings and the area have been computed. But this data gives a less significant picture.

In our tests we have made several runs, starting from a randomly generated input drawing, and collected the data of the best, the worst and the average run. The so obtained data is not significantly different, except for the number of crossings and when the number of iterations is too low. There it happened that the best drawings of some planar graphs such as trees, grids and triangular nets came out planar and others had many crossings.

Figures 1, 2 and 3 show test data on complete graphs up to K_{24} and on a set of mixed graphs, which consists of 59 graphs from the papers by Davidson and Harel [1] and Fruchterman and Reingold [5], ordered by the sum of the nodes and the edges. Further examples have been tested in [12]. The drawings are computed with our default settings as mentioned before. For DH they are $(1, 2, 0, 0)$ for the relative weights and $900n$ iterations split into 30 rounds of $30n$ trials and followed by $80n$ fine-tuning iterations, where n is the size of the graph. Since edge crossings and node edge distances are ignored, the inner loops of all five algorithms are of the same asymptotic complexity.

Figure 4 shows a comparison of DH without crossing costs $(1, 2, 0, 0)$ and with high crossing costs $(1, 2, 100, 0)$.

2.7 Evaluations

1. Overall, we can confirm the behaviour of the algorithms reported in the original papers. The algorithms reach the intended goals and produce drawings with uniform edge length and uniform node distribution. In this sense, the obtained drawings are aesthetically pleasing. This can also be said empirically for the visual impressions of the pictures.
2. The algorithms are stable against random input graphs. They converge towards one of the usually few stable drawings, which are of comparable quality, both empirically and by the collected data on the edge length etc.
3. FR, KK, GEM and DH without crossing optimization often produce drawings with a similar appearance. They display symmetry and perfom particularly well on (almost) complete graphs and on graphs from regular polytops and with a 3-D appearance, such as hypercubes, dodecahedron, icosahedron etc. or tori. Distorted drawings occurred for loosely connected graphs.
4. Tu often yields drawings that are different from those of the other programs. Thus, Tu is worth a trial, if the others fail. Tu does not display symmetry, which may be useful, if symmetries are not important. It performs well on grids and net structures, where the other algorithms sometimes failed to untangle distored inputs. However, its behaviour is hard to predict. Its quality parameter has an estimated exponential impact on the run time.
5. DH is the most flexible, but also the most time consuming algorithm. From our experiments we recommend to use more iterations than proposed by Davidson and Harel. This gives better and more stable results. However, DH is difficult to steer. It is difficult to find the proper mix for the relative weights. High penalties for crossings and close node edge distances often destroy the balance and the symmetry of the drawings and the uniformity of the edge length and the node distribution. And due to our implementation they cost time. This is underpinned by the data shown in Fig. 4. Some drawings produced by DH are illustrated in Fig. 5.
6. GEM and KK are very competitive in speed; the difference in speed in Fig. 1 is neglectible. They outperfom the others.
7. FR is fast on small graphs, but slows down on larger graphs with more than 60 nodes and edges.
8. KK produces smooth drawings with a low ratio of the longest and shortest edges and a small deviation of the edge length.

3 Conclusion

Each of the tested algorithms is a good tool for straight-line drawings of general undirected graphs. But there is no universal winner. For each algorithm we have found examples, where it produces the most pleasing drawings.

If you use these algorithms with the GraphEd system, we recommmend to try GEM or KK first, or FR if the graph is small with up to 60 nodes and edges. Next, try FR or Tu and finally DH. If time doesn't count, then play with the parameters of DH or Tu until you get a pleasing drawing. This recommendation takes quality and time into account.

If you have some knowledge of your input graph, e.g. planarity, and you insist on having a planar drawing, then do heavy-duty preprocessing and apply modified and adapted versions of these force-directed or randomized algorithms as a beautification step, as proposed in [7]. DH with high weights on crossings does not perform well, and it is difficult to find the proper mix of the parameters for reasonable or good drawings.

Acknowledgements. We wish to thank A. Ludwig for making the GEM available to us, and F. Dichtl for tests with the algorithms.

References

[1] Davidson, R., Harel, D.: Drawing graphs nicely using simulated annealing. Department of Applied Mathematics and Computer Science (1991)

[2] Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Algorithms for drawing graphs: an annotated bibliography. Comput. Geom. Theory Appl. **4** (1991) 235–282

[3] Di Battista, G., Garg, A., Liotta, G., Tassinari, E., Tamassia, R., Vargiu, F.: An experimental comparison of three graph drawing algorithms. Proc. 11th AMC Sympos. Comput. Geom. (1995)

[4] Eades, P.: A heuristic for graph drawing. Congressus Numeratium **42** (1984) 149–160

[5] Fruchtermann, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Software, Practice and Experience **21** (1991) 1129–1164

[6] Frick, A., Ludwig, A., Mehldau, H.: A fast adaptive layout algorithm for undirected graphs. Proc. Workshop on Graph Drawing 94. LNCS **894** (1994) 389–403

[7] Harel, D., Sardas, M.: Randomized graph drawing with heavy-duty preprocessing. Department of Applied Mathematics and Computer Science Weizmann Institute of Science, Rehovot/Israel, Technical Report **CS93-16** (1993)

[8] Himsolt, M.: Konzeption und Implementierung von Grapheneditoren. Dissertation, Universität Passau, Shaker Verlag Aachen (1993)

[9] Himsolt, M.: GraphEd: A graphical platform for the implementation of graph algorithms. Proc. Workshop on Graph Drawing 94, LNCS **894** (1994) 182–193

[10] Himsolt, M.: Comparing and evaluating layout algorithms within GraphEd. J. Visual Languages and Computing **6** (1995)

[11] Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Inf. Proc. Letters **31** (1989) 7–15

[12] Rohrer, C.: Layout von Graphen unter besonderer Berücksichtigung von probabilistischen Algorithmen. Diplomarbeit, Universität Passau (1995)

[13] Tunkelang, D.: A practical approach to drawing undirected graphs. Carnegie Mellon University (1994)

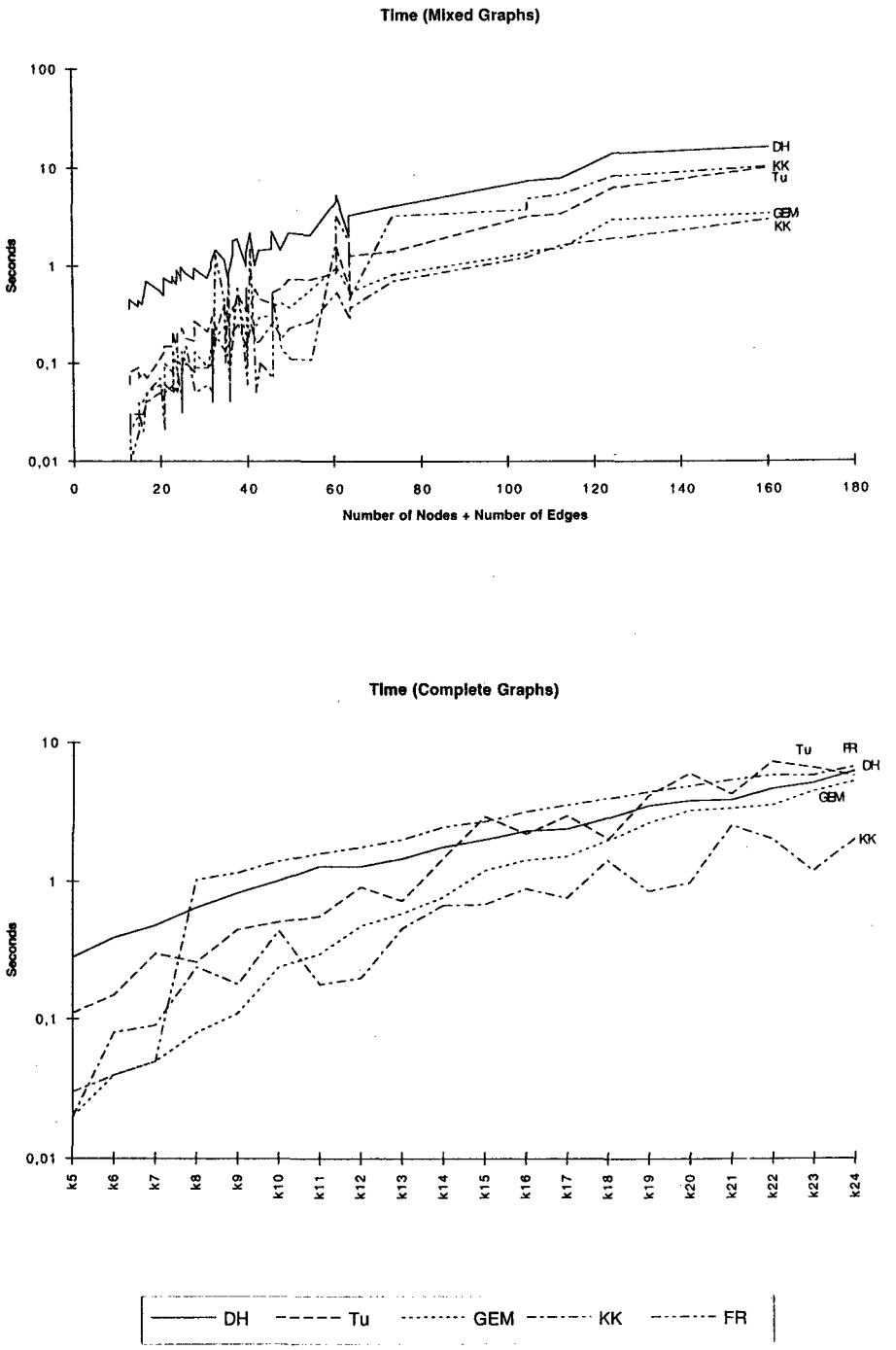

Fig. 1. Runtimes

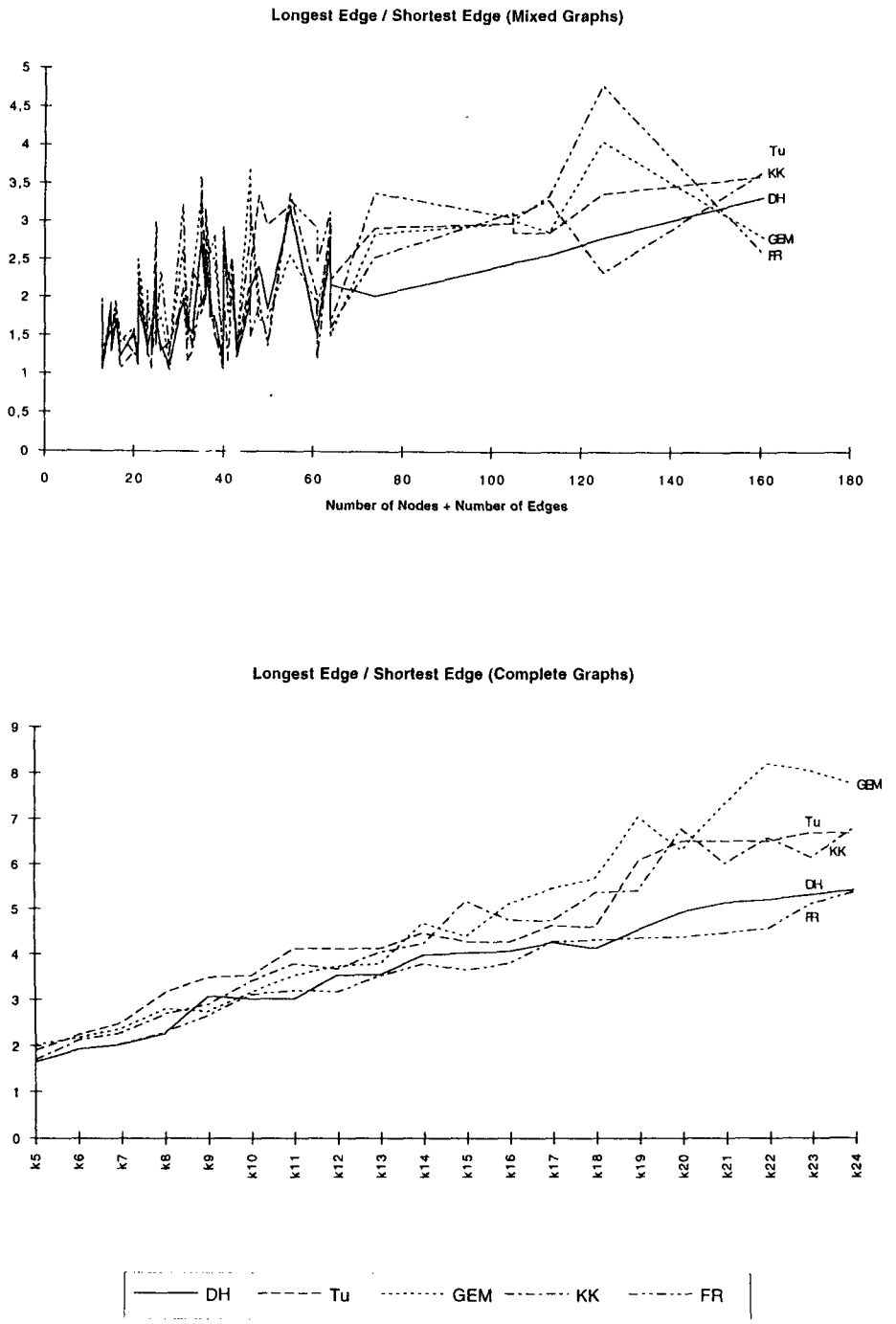

Fig. 2. Edge Length Ratios

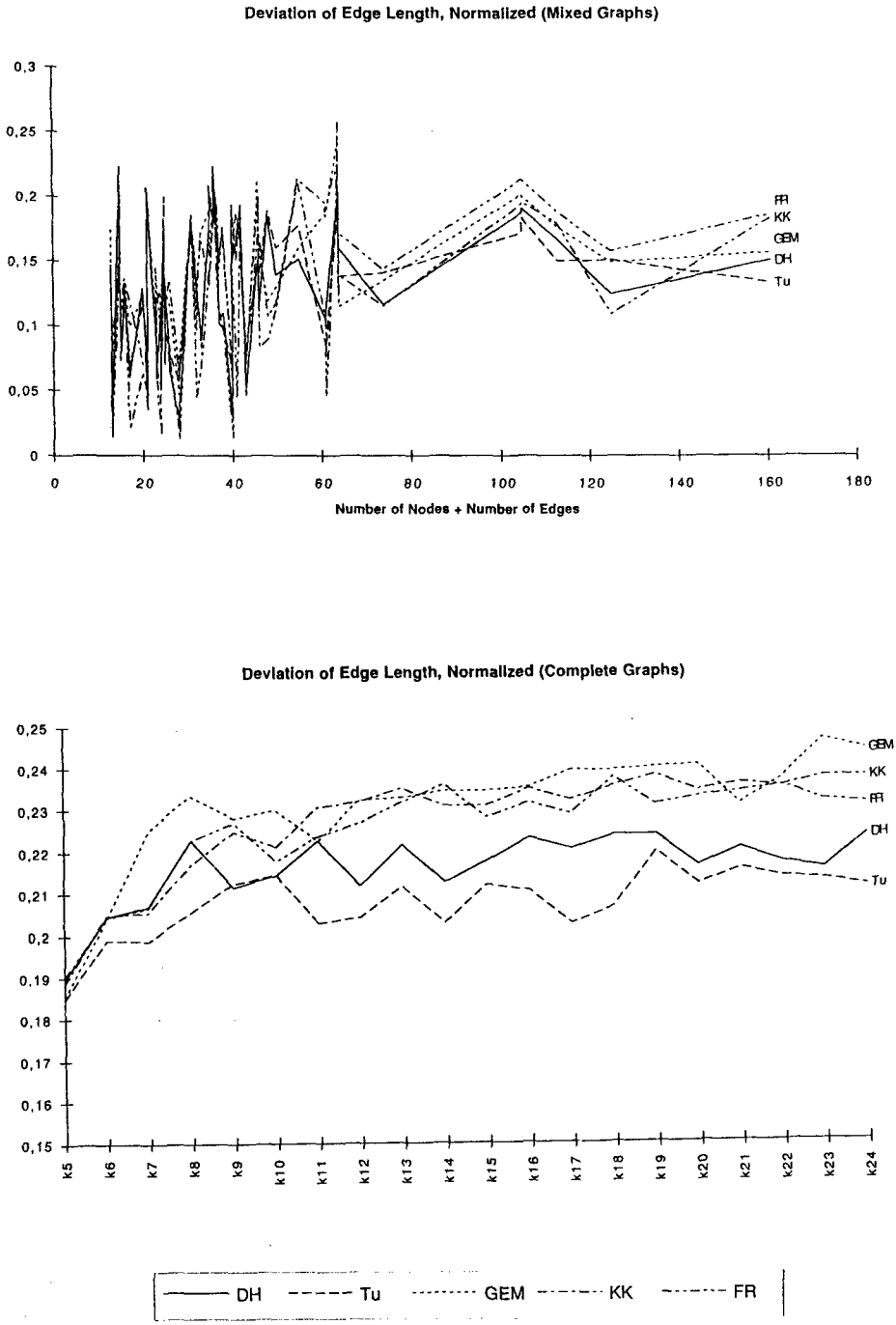

Fig. 3. Edge Length Deviation

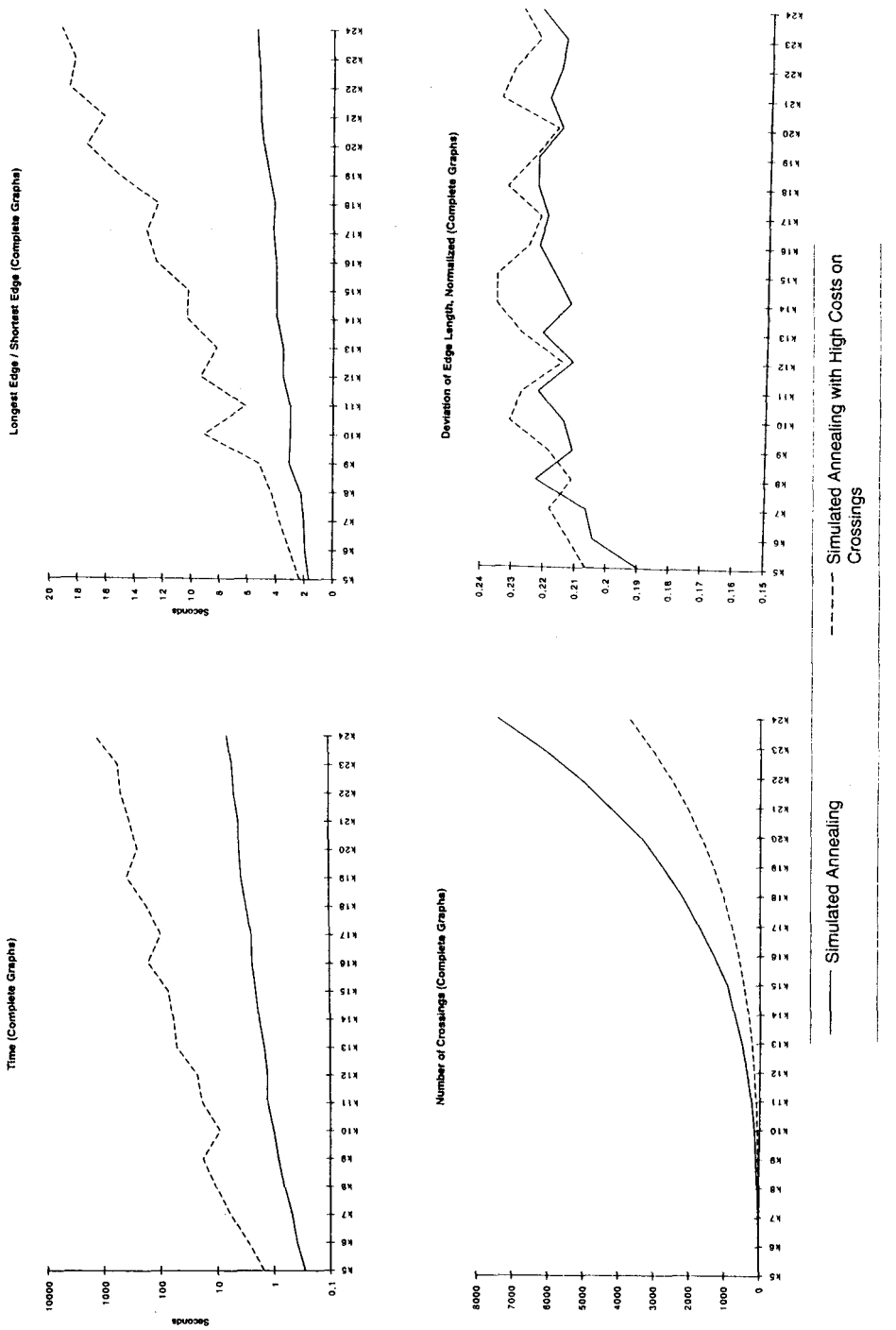

Fig. 4. Comparison of the DH algorithm with and without crossing optimization

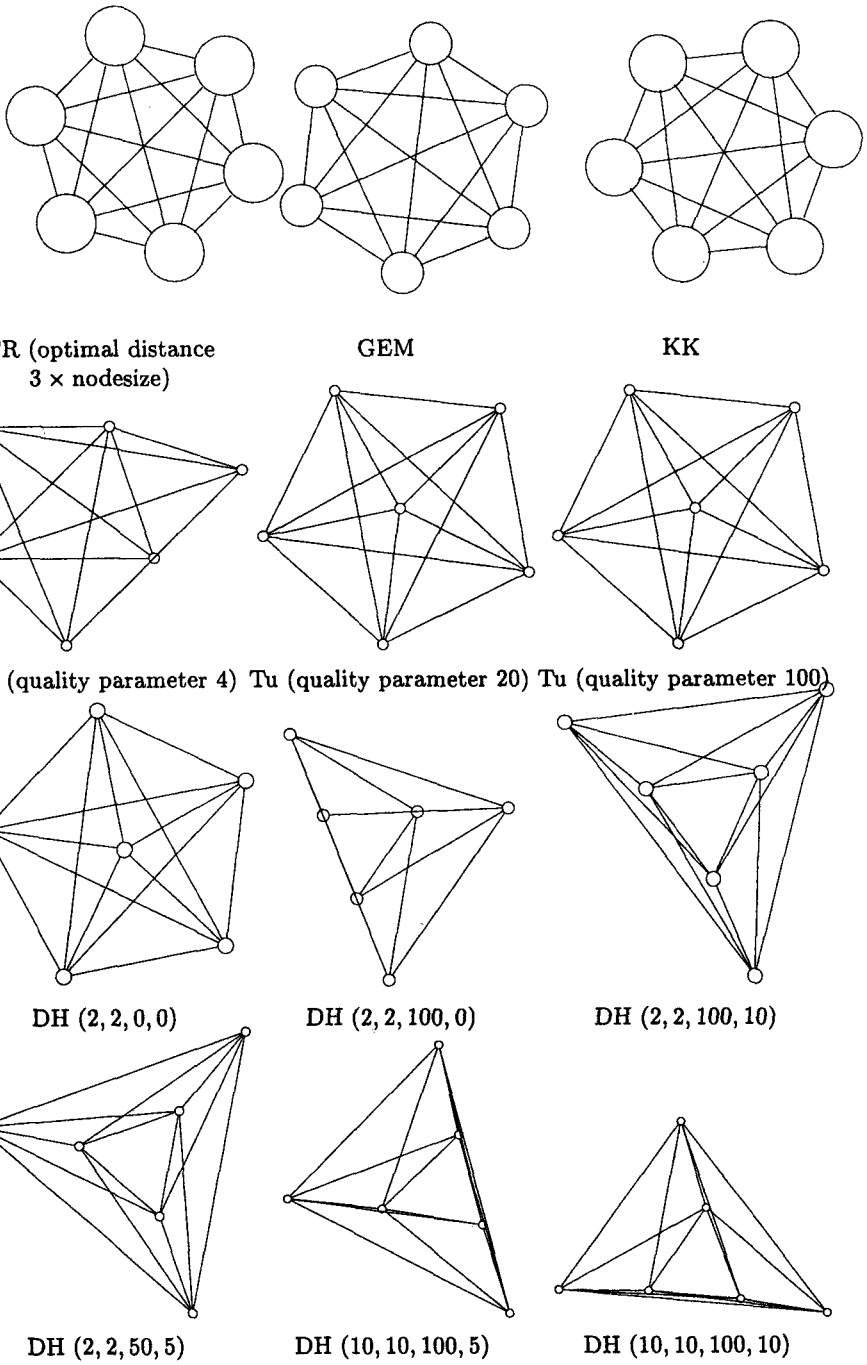

Fig. 5. Several drawings of the K_6

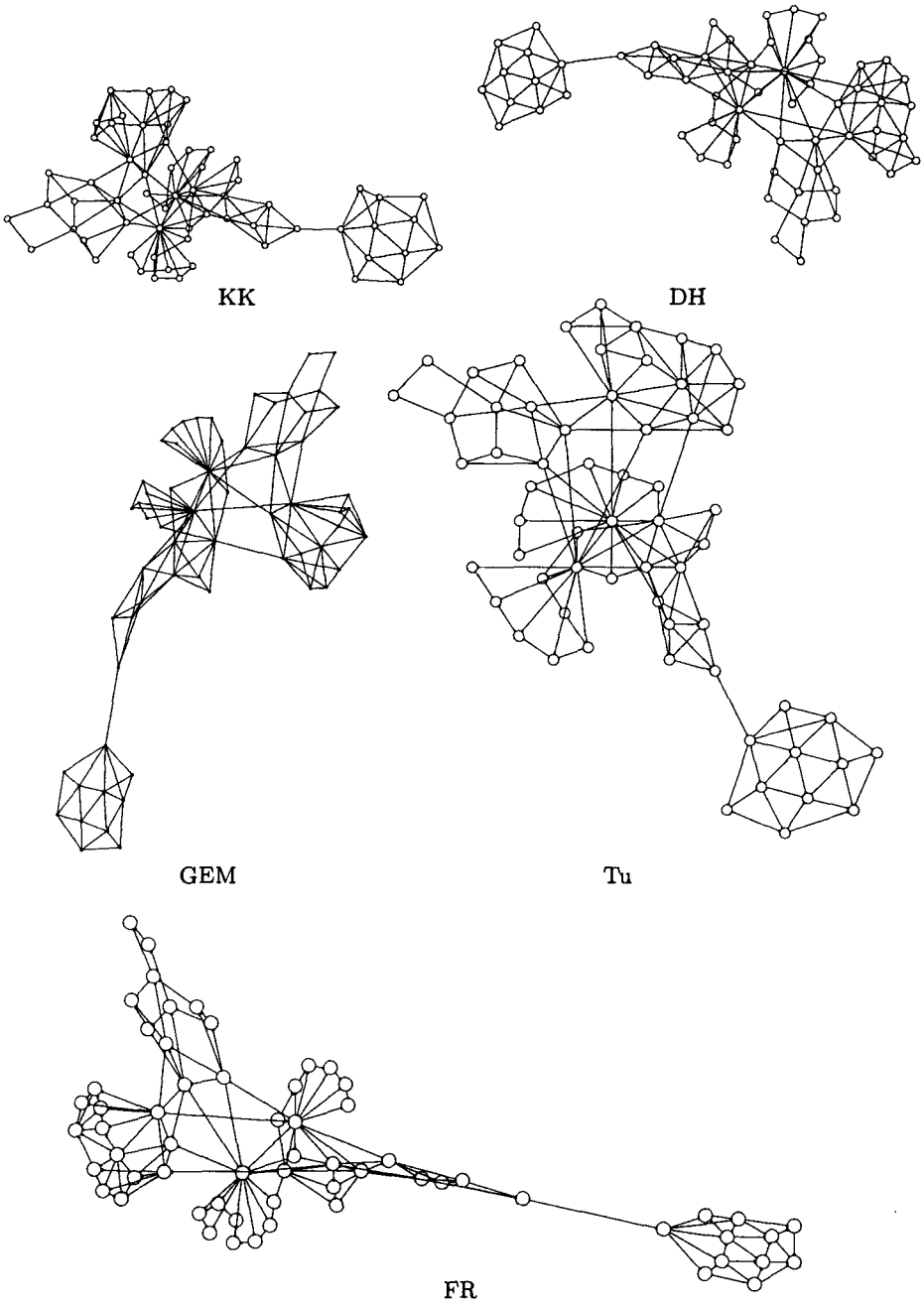

Fig. 6. Graphs from the graph drawing competition drawn with several algorithms