An Experimental Comparison of Fast Algorithms for Drawing General Large Graphs

Stefan Hachul and Michael Jünger

Universität zu Köln, Institut für Informatik,
Pohligstraße 1, 50969 Köln, Germany
{hachul,mjuenger}@informatik.uni-koeln.de

Abstract. In the last decade several algorithms that generate straight-line drawings of general large graphs have been invented. In this paper we investigate some of these methods that are based on force-directed or algebraic approaches in terms of running time and drawing quality on a big variety of artificial and real-world graphs. Our experiments indicate that there exist significant differences in drawing qualities and running times depending on the classes of tested graphs and algorithms.

1 Introduction

Force-directed graph drawing methods generate drawings of a given general graph $G = (V, E)$ in the plane in which each edge is represented by a straight line connecting its two adjacent nodes. The computation of the drawings is based on associating G with a physical model. Then, an iterative algorithm tries to find a placement of the nodes so that the total energy of the physical system is minimal. Important esthetic criteria are uniformity of edge length, few edge crossings, non-overlapping nodes, and the display of symmetries if some exist.

Classical force-directed algorithms like [5, 15, 7, 4, 6] are used successfully in practice (see e.g. [2]) for drawing general graphs containing few hundreds of vertices. However, in order to generate drawings of graphs that contain thousands or hundreds of thousands of vertices more efficient force-directed techniques have been developed [19, 18, 9, 8, 12, 21, 11, 10]. Besides fast force-directed algorithms other very fast methods for drawing large graphs (see e.g. [13, 16]) have been invented. These methods are based on techniques of linear algebra instead of physical analogies. But they strive for the same esthetic drawing criteria.

Previous experimental tests of these methods are mainly restricted to regular graphs with *grid-like* structures (see e.g. [13, 16, 9, 21, 12]). Since general graphs share these properties quite seldom, and since the test environments of these experiments are different, a standardized comparison of the methods on a wider range of graphs is needed.

In this study we experimentally compare some of the fastest state-of-the-art algorithms for straight-line drawing of general graphs on a big variety of graph classes. In particular, we investigate the force-directed algorithm `GRIP` of Gajer and Kobourov [9] and Gajer et al. [8], the *Fast Multi-scale Method* (`FMS`) of

Harel and Koren [12], and the *Fast Multipole Multilevel Method* ($\mathtt{FM}^3$) of Hachul and Jünger [11, 10]. The examined algebraic methods are the algebraic multigrid method `ACE` of Koren et al. [16] and the *high-dimensional embedding* approach (`HDE`) by Harel and Koren [13]. Additionally, one of the faster classical force-directed algorithms, namely the *grid-variant algorithm* (`GVA`) of Fruchterman and Reingold [7], is tested as a benchmark.

After a short description of the tested algorithms in Section 2 and of the experimental framework in Section 3, our results are presented in Section 4.

2 The Algorithms

2.1 The Grid-Variant Algorithm (`GVA`)

The grid-variant algorithm of Fruchterman and Reingold [7] is based on a model of pairwise repelling charged particles (the nodes) and attracting springs (the edges), similar to the model of the `Spring Embedder` of Eades [5]. Since a naive exact calculation of the repulsive forces acting between all pairs of charges needs $\Theta(|V^2|)$ time per iteration, `GVA` does only calculate the repulsive forces acting between nodes that are placed relatively near to each other. Therefore, the rectangular drawing area is subdivided into a regular square grid. The repulsive forces that act on a node v that is contained in a grid box B are approximated by summing up only the repulsive forces that are induced by the nodes contained in B and the nodes in the grid boxes that are neighbors of B. If the number of iterations is assumed to be constant, the best-case running time of the `GVA` is $\Theta(|V| + |E|)$. The worst-case running time, however, remains $\Theta(|V|^2 + |E|)$.

2.2 The Method `GRIP`

Gajer et al. [8] and Gajer and Kobourov [9] developed the force-directed multilevel algorithm `GRIP`. In general, multilevel algorithms are based on two phases. A *coarsening phase*, in which a sequence of coarse graphs with decreasing sizes is computed and a *refinement phase* in which successively drawings of finer graphs are computed, using the drawings of the next coarser graphs and a variant of a suitable force-directed single-level algorithm.

The coarsening phase of `GRIP` is based on the construction of a *maximum independent set filtration* or *MIS filtration* of the node set V. A MIS filtration is a family of sets $\{V =: V_0, V_1, \ldots, V_k\}$ with $\emptyset \subset V_k \subset V_{k-1} \ldots \subset V_0$ so that each V_i with $i \in \{1, \ldots, k\}$ is a maximal subset of V_{i-1} for which the graph-theoretic distance between any pair of its elements is at least $2^{i-1} + 1$. Gajer and Kobourov [9] use a `Spring Embedder`-like method as single-level algorithm at each level. The used force vector is similar to that used in the Kamada-Kawai method [15], but is restricted to a suitable chosen subset of V_i.

Other notable specifics of `GRIP` are that it computes the MIS filtration only and no edge sets of the coarse graphs $G_0, \ldots, G_k$ that are induced by the filtrations. Furthermore, it is designed to place the nodes in an n-dimensional space

($n \geq 2$), to draw the graph in this space, and to project it into two or three dimensions.

The asymptotic running time of the algorithm, excluding the time that is needed to construct the MIS filtration, is $\Theta(|V|(\log \mathit{diam}(G)^2))$ for graphs with bounded maximum node degree, where $\mathit{diam}(G)$ denotes the diameter of G.

2.3 The Fast Multi-scale Method (FMS)

In order to create the sequence of coarse graphs in the force-directed multilevel method FMS, Harel and Koren [12] use an $O(k|V|)$ algorithm that finds a 2-approximative solution of the $\mathcal{NP}$-hard *k-center problem.* The node set V_i of a graph G_i in the sequence $G_0, \ldots, G_k$ is determined by the approximative solution of the k_i-center problem on G with $k_i > k_{i+1}$ for all $i \in \{0, \ldots, k-1\}$.

The authors use a variation of the algorithm of Kamada and Kawai [15] as force-directed single-level algorithm. In order to speed up the computation of this method, they modify the energy function of Kamada and Kawai [15] that is associated with a graph G_i with $i \in \{0, \ldots, k-1\}$. The difference to the original energy of Kamada and Kawai [15] is that only some of the $|V(G_i)| - 1$ springs that are connected with a node $v \in V(G_i)$ are considered.

The asymptotic running time of the FMS is $\Theta(|V||E|)$. Additionally, $\Theta(|V|^2)$ memory is needed to store the distances between all pairs of nodes.

2.4 The Fast Multipole Multilevel Method (FM^3)

The force-directed multilevel algorithm FM^3 has been introduced by Hachul and Jünger [11, 10]. It is based on a combination of an efficient multilevel technique with an $O(|V| \log |V|)$ approximation algorithm to obtain the repulsive forces between all pairs of nodes.

In the coarsening step subgraphs with a small diameter (called *solar systems*) are collapsed to obtain a multilevel representation of the graph. In the used single-level algorithm, the bottleneck of calculating the repulsive forces acting between all pairs of charged particles in the Spring Embedder-like force model is overcome by rapidly evaluating potential fields using a novel multipole-based tree-code. The worst-case running time of FM^3 is $O(|V| \log |V| + |E|)$ with linear memory requirements.

2.5 The Algebraic Multigrid Method ACE

In the description of their method ACE, Koren et al. [16] define the quadratic optimization problem

$$(P) \qquad \min\ x^T L x \quad \text{so that} \quad x^T x = 1 \quad \text{in the subspace} \quad x^T 1_n = 0\,.$$

Here $n = |V|$ and L is the *Laplacian* matrix of G.

The minimum of (P) is obtained by the eigenvector that corresponds to the smallest positive eigenvalue of L. The problem of drawing the graph G in two dimensions is reduced to the problem of finding the two eigenvectors of L that are associated with the two smallest eigenvalues.

Instead of calculating the eigenvectors directly, an *algebraic multigrid algorithm* is used. Similar to the force-directed multilevel ideas, the idea is to express the originally high-dimensional problem in lower and lower dimensions, solving the problem at the lowest dimension, and progressively solving a high-dimensional problem by using the solutions of the low-dimensional problems.

The authors do not give an upper bound on the asymptotic running time of `ACE` in the number of nodes and edges.

2.6 High-Dimensional Embedding (`HDE`)

The method `HDE` of Harel and Koren [13] is based on a two phase approach that first generates an embedding of the graph in a very high-dimensional vector space and then projects this drawing into the plane.

The high-dimensional embedding of the graph is generated by first using a linear time algorithm for approximatively solving the k-center problem. A fixed value of $k = 50$ is chosen, and k is also the dimension of the high-dimensional vector space. Then, breadth-first search starting from each of the k center nodes is performed resulting in k $|V|$-dimensional vectors that store the graph-theoretic distances of each $v \in V$ to each of the k centers. These vectors are interpreted as a k-dimensional embedding of the graph.

In order to project the high-dimensional embedding of the graph into the plane, the k vectors are used to define a *covariance* matrix S. The x- and y-coordinates of the two-dimensional drawing are obtained by calculating the two eigenvectors of S that are associated with its two largest eigenvalues. `HDE` runs in $O(|V| + |E|)$ time.

3 The Experiments

3.1 Test-Environment, Implementations, and Parameter Settings

All experiments were performed on a 2.8 GHz Intel Pentium 4 PC with one gigabyte of memory.

We tested a version of `GVA` that has been implemented in the framework of AGD [14] by S. Näher and D. Alberts, an implementation of `GRIP` by R. Yusufov that is available from [22], and implementations of `FMS`, `ACE`, `HDE` by Y. Koren that are available from [17]. Finally, we tested our own implementation of $\mathtt{FM}^3$.

In order to obtain a fair comparison, we ran each algorithm with the same set of standard-parameter settings (given by the authors) on each tested graph. However, we are aware that in some cases it might be possible to obtain better results by spending a considerable amount of time with trial-and-error searching for an optimal set of parameters for each algorithm and graph.

3.2 The Set of Test Graphs

Since only few implementations can handle disconnected and weighted graphs, we restrict to connected unweighted graphs, here.

We generated several classes of artificial graphs to examine the scaling of the algorithms on graphs with predefined structures but different sizes.

These are *random grid* graphs that were obtained by first creating regular square grid graphs and then randomly deleting 3% of the nodes. The *sierpinski* graphs were created by associating the *Sierpinski Triangles* with graphs. Furthermore, we generated complete 6-*nary trees*.

The next two classes of artificial graphs were designed to test how well the algorithms can handle highly non-uniform distributions of the nodes and high node degrees. Therefore, we created these graphs in a way so that one can expect that an energy-minimal configuration of the nodes in a drawing that relies on a `Spring Embedder`-like force model induces a tiny subregion of the drawing area which contains $\Theta(|V|)$ nodes. In particular, we constructed trees that contain a root node r with $|V|/4$ neighbors. The other nodes were subdivided into six subtrees of equal size rooted at r. We called these graphs *snowflake* graphs.

Additionally we created *spider* graphs by constructing a circle C containing 25% of the nodes. Each node of C is also adjacent to 12 other nodes of the circle. The remaining nodes were distributed on 8 paths of equal length that were rooted at one node of C. In contrast to the snowflake graphs is that the spider graphs have bounded maximum degree.

The last kind of artificial graphs are graphs with a relatively high edge density $|E|/|V| \geq 14$. We called them *flower* graphs. They are constructed by joining 6 circles of equal length at a single node before replacing each of the nodes by a complete subgraph with 30 nodes (K_{30}).

The rest of the test graphs are taken from real-world applications. In particular, we selected graphs from the *AT&T graph library* [1], from C. Walshaw's graph collection [20], and a graph that describes a social network of 2113 people that we obtained from C. Lipp.

We partitioned the artificial and real-world graphs into two sets. The first set are graphs that consist of few biconnected components, have a constant maximum node degree, and have a low edge density. Furthermore, one can expect that an energy-minimal configuration of the nodes in a `Spring Embedder` drawing of such a graph does not contain $\Theta(|V|)$ nodes in an extremely tiny subregion of the drawing area. Since one can anticipate from previous experiments [13, 16, 9, 12] that the graphs contained in this set do not cause problems for many of the tested algorithms, we call the set of these graphs *kind*. The second set is the complement of the first one, and we call the set of these graphs *challenging*.

3.3 The Criteria of Evaluation

The natural criteria to evaluate a graph-drawing algorithm in practice are the needed running times and the quality of the drawings.

Unlike evaluating the first criterion, evaluating the quality of a drawing is a difficult task. Possible ways are the calculation of the total energy in the underlying force models or the measurement of relevant esthetic criteria (e.g. crossing number, uniformity of the edge lengths). However, one of the most important goals is that an individual user is satisfied with a drawing. Hence, we decided to print the drawings and to comment how well they display the structure of each graph by keeping the modeled esthetic criteria in mind.

4 The Results

4.1 Comparison of the Running Times

Table 1 presents the running times of the methods `GVA`, `FM`^{3}, `GRIP`, `FMS`, `ACE`, and `HDE` for the tested graphs.

As expected, in most cases `GVA` is the slowest method among the force-directed algorithms. The largest graph fe_ocean is drawn by `GVA` in 5 hours and 20 minutes.

Graph Information							Algorithm Information					
							CPU Time in Seconds					
Type	Name	$\|V\|$	$\|E\|$	$\|B\|$	$\frac{\|E\|}{\|V\|}$	max. degree	GVA	FM^3	GRIP	FMS	ACE	HDE
Kind Artificial	rnd_grid_032	985	1834	2	1.8	4	12.5	1.9	0.3	1.0	< 0.1	< 0.1
	rnd_grid_100	9497	17849	6	1.8	4	203.4	19.1	4.4	32.0	0.5	0.1
	rnd_grid_320	97359	184532	2	1.9	4	6316.1	215.4	(E)	(M)	4.1	1.3
	sierpinski_06	1095	2187	1	2.0	4	13.1	1.8	0.3	1.0	< 0.1	< 0.1
	sierpinski_08	9843	19683	1	2.0	4	171.7	16.8	4.8	33.0	1.0	0.1
	sierpinski_10	88575	177147	1	2.0	4	3606.4	162.0	(E)	(M)	23.4	1.0
Kind Real World	crack	10240	30380	1	2.9	9	317.5	23.0	6.8	(M)	0.4	0.2
	fe_pwt	36463	144794	55	3.9	15	1869.1	69.0	(E)	(M)	(T)	0.5
	finan_512	74752	261120	1	3.4	54	6319.8	158.2	(E)	(M)	7.5	1.0
	fe_ocean	143437	409593	39	2.8	6	19247.0	355.9	(E)	(M)	4.0	3.4
Challenging Artificial	tree_d_4	1555	1554	1554	1.0	7	14.3	2.6	0.3	2.0	< 0.1	< 0.1
	tree_d_5	9331	9330	9330	1.0	7	130.3	17.7	2.4	43.0	0.5	< 0.1
	tree_d_6	55987	55986	55986	1.0	7	1769.2	121.3	(E)	(M)	4.5	0.5
	snowflake_A	971	970	970	1.0	256	8.0	1.6	0.4	73.0	0.4	< 0.1
	snowflake_B	9701	9700	9700	1.0	2506	143.2	17.4	6.1	3320.0	(T)	< 0.1
	snowflake_C	97001	97000	97000	1.0	25006	14685.7	166.5	(E)	(M)	(T)	0.8
	spider_A	1000	2200	801	2.2	18	17.6	1.9	0.4	1.0	1.1	< 0.1
	spider_B	10000	22000	8001	2.2	18	189.0	17.7	7.2	47.0	8.9	0.1
	spider_C	100000	220000	80001	2.2	18	4568.3	177.2	(E)	(M)	280.7	1.3
	flower_A	930	13521	1	14.5	30	61.7	1.2	0.7	1.0	< 0.1	< 0.1
	flower_B	9030	131241	1	14.5	30	595.1	11.9	19.3	46.0	1.4	0.2
	flower_C	90030	1308441	1	14.5	30	11841.5	121.4	(E)	(M)	(T)	1.4
Challenging Real World	ug_380	1104	3231	27	2.9	856	23.1	2.1	0.4	1.0	< 0.1	< 0.1
	esslingen	2075	5530	867	2.6	97	43.8	4.0	0.5	404.0	1.0	< 0.1
	add_32	4960	9462	951	1.9	31	80.6	12.1	1.6	17.0	0.5	< 0.1
	dg_1087	7602	7601	7601	1.0	6566	624.8	18.1	3.6	5402.0	108.4	< 0.1
	bcsstk_33	8738	291583	1	33.3	140	1494.6	23.8	29.1	6636.0	0.4	0.3
	bcsstk_31	35586	572913	48	16.1	188	4338.4	83.6	(E)	(M)	1.9	0.7
	bcsstk_32	44609	985046	3	22.0	215	6387.1	110.9	(E)	(M)	3.6	0.9

Table 1. The test graphs and the running times that are needed by the tested algorithms to draw them. Explanations: (E) No drawing was generated due to an error in the executable. (M) No drawing was generated because the memory is restricted to graphs with $\leq 10,000$ nodes. (T) No drawing was generated within 10 hours of CPU time. B denotes the set of biconnected components of the graphs.

The method FM^3 is significantly faster than GVA for all tested graphs. The running times range from less than 2 seconds for the smallest graphs to less than 6 minutes for the largest graph fe_ocean. The subquadratic scaling of FM^3 can be experimentally confirmed for all classes of tested graphs.

Except for the dense graphs flower_B and bcsstk_33 GRIP is faster than FM^3 (up to a factor 9). Unfortunately, we could not examine the scaling of GRIP for the largest graphs due to an error in the executable.

Since the memory requirement of FMS is quadratic in the size of the graph, the implementation of FMS is restricted to graphs that contain at most 10, 000 nodes. The running time of FMS is comparable with that of FM^3 for the smallest and the medium sized kind graphs. In contrast to this, the CPU time of FMS increases drastically for several challenging graphs, in particular for graphs that either contain nodes with a very high degree or have a high edge density.

The algorithm ACE is much faster than the force-directed algorithms for nearly all kind graphs. However, like for FMS, the running times grow extremely when ACE is used to draw several of the challenging graphs.

The linear time method HDE is by far the fastest algorithm. It needs less than 3.4 seconds for drawing even the largest tested graph.

4.2 Comparison of the Drawings

For all kind graphs the classical method GVA does not untangle the drawings that were induced by the random initial placements.

In contrast to this nearly all algorithms generated comparable pleasing drawings of the kind graphs (see Figure 1, Figure 2, and Figure 3(a)-(d)).

None of the drawings of the complete 6-nary trees (see Figure 3(e)-(j)) is really convincing, since the force-directed algorithms produce many unnecessary edge crossings. However, the drawings generated by FM^3 and FMS display parts of the regularity of these graphs. The algebraic methods ACE and HDE place many nodes at the same coordinates. In general, this behavior of the algebraic methods can be observed for graphs that consist of many biconnected components. Explanations of the theoretical reasons can be found in [3, 16] and [10].

Except FM^3 none of the tested algorithms displays the global structure of the snowflake graphs. Even the drawings of the smallest snowflake graph (see Figure 4(a)-(f)) leave room for improvement. However, GVA and GRIP visualize parts of its structure in an appropriate way.

The drawings of the spider_A graph (see Figure 4(g)-(l)) that are generated by GRIP, FMS, and HDE are not as symmetric as that generated by FM^3. But they display the global structure of the graph. The drawing generated by GVA shows the dense subregion, but GVA does not untangle the 8 paths. The paths in the drawing of ACE are not displayed in the same length. The drawings of the larger spider graphs are of comparable quality.

The drawings of the flower_B graph (see Figure 5(a)-(f)) that are generated by FMS and HDE display the global structure of the graph but the symmetries are not as clear as in the drawing generated by FM^3. The drawings of the other flower graphs are of comparable quality.

We concentrate on the challenging real-world graphs now. The graphs ug_380 and dg_1087 both contain one node with a very high degree. Furthermore, dg_1087 has many biconnected components, since it is a tree. Only the drawings that are generated by GVA, FM^3, and GRIP (see Figure 5(g)-(l) and Figure 6(a)-(f)) clearly display the central regions of these graphs. It can be observed that the edge lengths of the drawing of dg_1087 that is generated by FM^3 are more uniform than in the drawings of dg_1087 that are generated by GVA and GRIP.

The social network esslingen (see Figure 6(g)-(l)) consists of two big well-connected subgraphs. This can be visualized by FM^3, GRIP, and HDE. But all drawings contain many edge crossings.

Since add_32 that describes a 32 bit adder contains many biconnected components, we expect that the drawings have a tree-like shape. This structure is visualized by GVA, FM^3, GRIP, and ACE (see Figure 7(a)-(f)). The drawings of GVA and GRIP contain many edge crossings, while the drawing of ACE displays the global structure, but hides local details.

Finally, we discuss the drawings of the graphs bcsstk_31_con, bcsstk_32, and bcsstk_33 that have a very high edge density. The drawings of bcsstk_33 (see Figure 7(g)-(l)) that are generated by FM^3, GRIP, and ACE are comparable and visualize the regular structure of the graph. The car body that is modeled by the graph bcsstk_31_con (see Figure 8(a)-(d)) is visualized by FM^3 and ACE only. All drawings of bcsstk_32 are completely different (see Figure 8(e)-(h)) and an evaluation of the drawings is left to the reader.

5 Conclusion

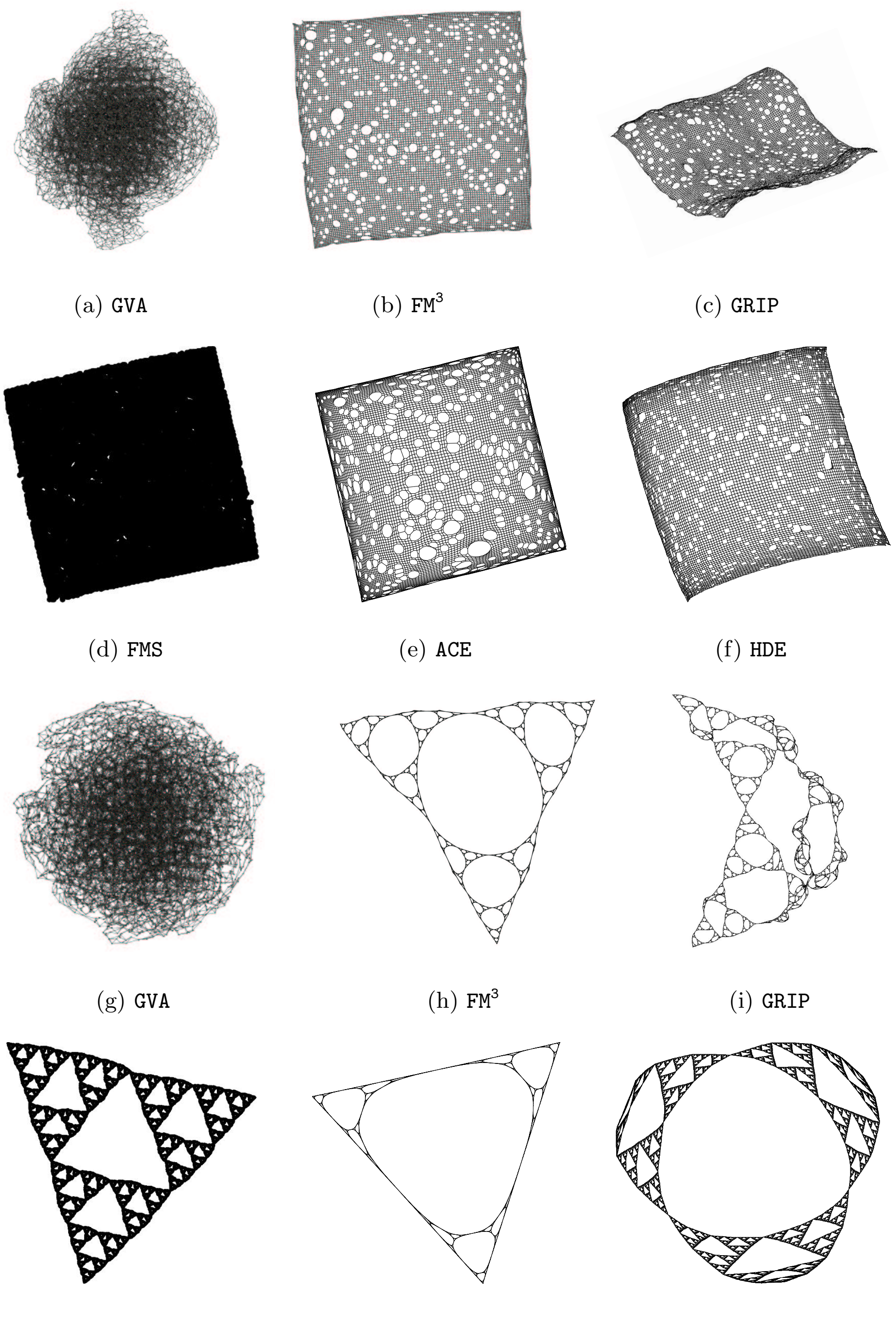
We can summarize that only GVA, FM^3, and HDE generate drawings of all tested graphs. The force-directed multilevel methods and the algebraic methods are — except the methods FMS and ACE for some graphs — much faster than the comparatively slow classical algorithm GVA. HDE, FM^3 and GRIP scale well on all tested graphs. FM^3 needs few minutes to draw the largest graphs. GRIP is up to factor 9 faster than FM^3 but it could not be tested on the largest graphs. All tested methods are much slower than HDE that needs only few seconds to draw even the largest graphs.

As expected, all algorithms, except GVA, generate pleasing drawings of the kind graphs. In contrast to this, the quality of the generated drawings varies a lot depending on the structures of the tested challenging graphs. Only FM^3 generates pleasing drawings for the majority of the challenging graphs. But there still remain classes of tested graphs (the complete trees and the social network graph esslingen) for which the drawing quality of all tested algorithms leaves room for improvement.

Acknowledgments We would like to thank David Alberts, Steven Kobourov, Yehuda Koren, Stefan Näher, and Roman Yusufov for making the implementations of their algorithms available to us. We thank Ulrik Brandes, Carola Lipp and Chris Walshaw for the access to the real-world test graphs.

References

1. The AT&T graph collection: `www.graphdrawing.org`.
2. F. J. Brandenburg, M. Himsolt, and C. Rohrer. An Experimental Comparison of Force-Directed and Randomized Graph Drawing Methods. In *Graph Drawing 1995*, volume 1027 of *LNCS*, pages 76–87. Springer-Verlag, 1996.
3. U. Brandes and D. Wagner. In *Graph Drawing Software*, volume XII of *Mathematics and Visualization*, chapter visone - Analysis and Visualization of Social Networks, pages 321–340. Springer-Verlag, 2004.
4. R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.
5. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
6. A. Frick, A. Ludwig, and H. Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs. In *Graph Drawing 1994*, volume 894 of *LNCS*, pages 388–403. Springer-Verlag, 1995.
7. T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-directed Placement. *Software–Practice and Experience*, 21(11):1129–1164, 1991.
8. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs. In *Graph Drawing 2000*, volume 1984 of *LNCS*, pages 211–221. Springer-Verlag, 2001.
9. P. Gajer and S. G. Kobourov. GRIP: Graph Drawing with Intelligent Placement. In *Graph Drawing 2000*, volume 1984 of *LNCS*, pages 222–228. Springer-Verlag, 2001.
10. S. Hachul. *A Potential-Field-Based Multilevel Algorithm for Drawing Large Graphs*. PhD thesis, Institut für Informatik, Universität zu Köln, Germany, 2005.
11. S. Hachul and M. Jünger. Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm (Extended Abstract). In *Graph Drawing 2004*, volume 3383 of *Lecture Notes in Computer Science*, pages 285–295. Springer-Verlag, 2005.
12. D. Harel and Y. Koren. A Fast Multi-scale Method for Drawing Large Graphs. In *Graph Drawing 2000*, volume 1984 of *LNCS*, pages 183–196. Springer-Verlag, 2001.
13. D. Harel and Y. Koren. Graph Drawing by High-Dimensional Embedding. In *Graph Drawing 2002*, volume 2528 of *LNCS*, pages 207–219. Springer-Verlag, 2002.
14. M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. In *Graph Drawing Software*, volume XII of *Mathematics and Visualization*, chapter AGD - A Library of Algorithms for Graph Drawing, pages 149–172. Springer-Verlag, 2004.
15. T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31:7–15, 1989.
16. Y. Koren, L. Carmel, and D. Harel. Drawing Huge Graphs by Algebraic Multigrid Optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003.
17. Y. Koren's algorithms: `research.att.com/~yehuda/index_programs.html`.
18. A. Quigley and P. Eades. FADE: Graph Drawing, Clustering, and Visual Abstraction. In *Graph Drawing 2000*, volume 1984 of *LNCS*, pages 197–210. Springer-Verlag, 2001.
19. D. Tunkelang. JIGGLE: Java Interactive Graph Layout Environment. In *Graph Drawing 1998*, volume 1547 of *LNCS*, pages 413–422. Springer-Verlag, 1998.
20. C. Walshaw's graph collection: `staffweb.cms.gre.ac.uk/~c.walshaw/partition`.
21. C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. In *Graph Drawing 2000*, volume 1984 of *LNCS*, pages 171–182. Springer-Verlag, 2001.
22. R. Yusufov's implementation of GRIP: `www.cs.arizona.edu/~kobourov/GRIP`.



(a) GVA (b) FM^3 (c) GRIP

(d) FMS (e) ACE (f) HDE

(g) GVA (h) FM^3 (i) GRIP

(j) FMS (k) ACE (l) HDE

Fig. 1. (a)-(f) Drawings of rnd_grid_100 and (g)-(l) sierpinski_08 generated by different algorithms.

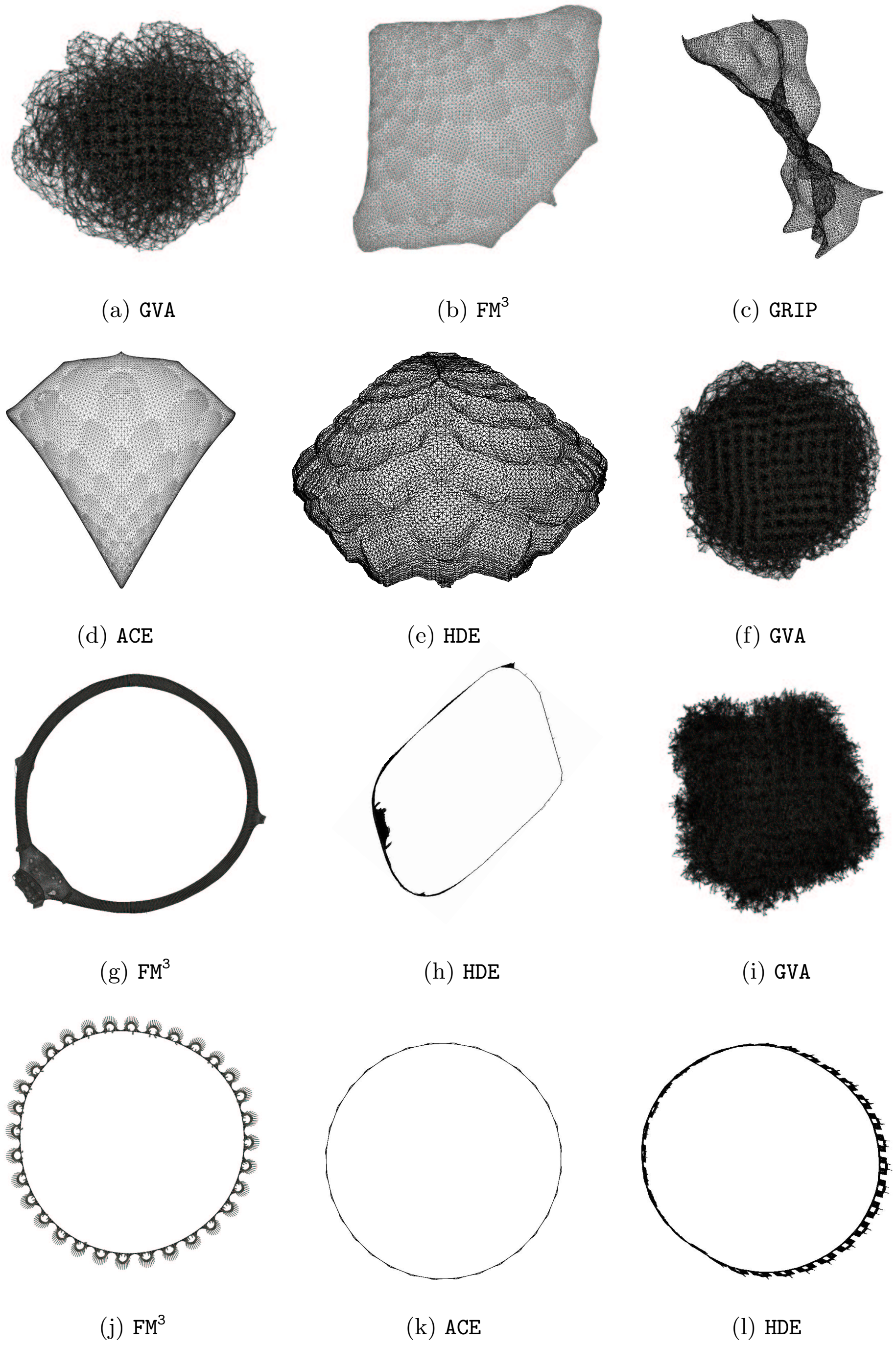


Fig. 2. (a)-(e) Drawings of crack, (f)-(h) fe_pwt, and (i)-(l) finan_512 generated by different algorithms.

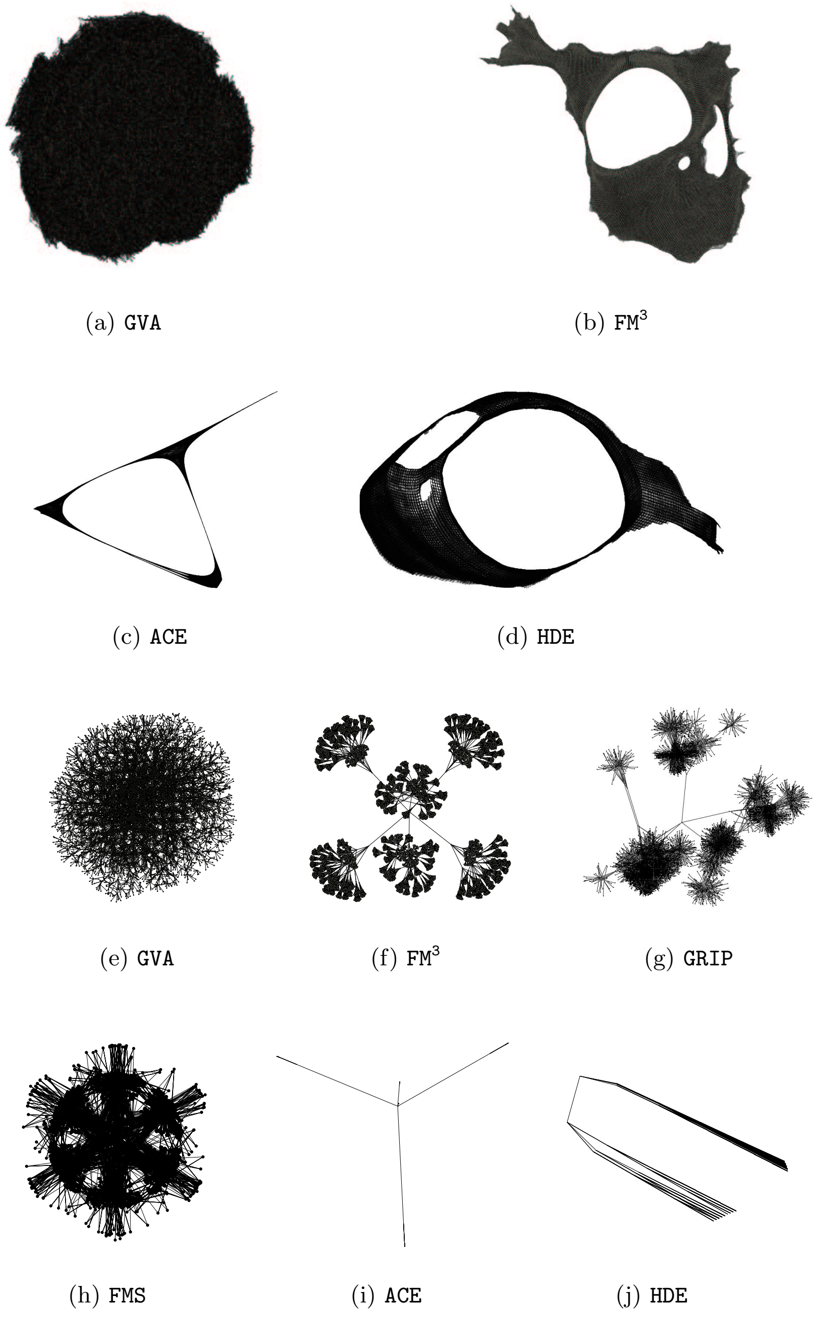
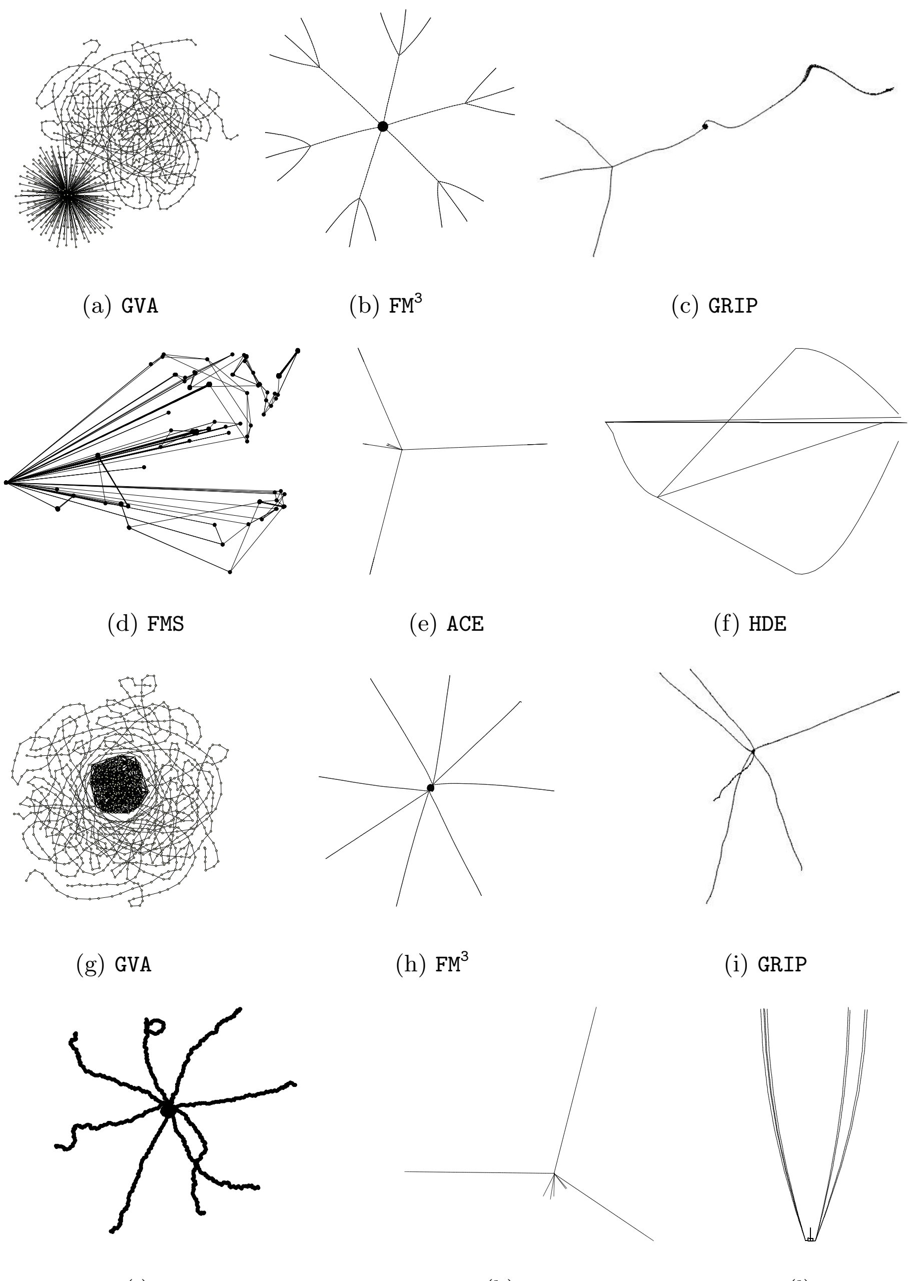


Fig. 3. (a)-(d) Drawings of fe_ocean and (e)-(j) tree_06_05 generated by different algorithms.

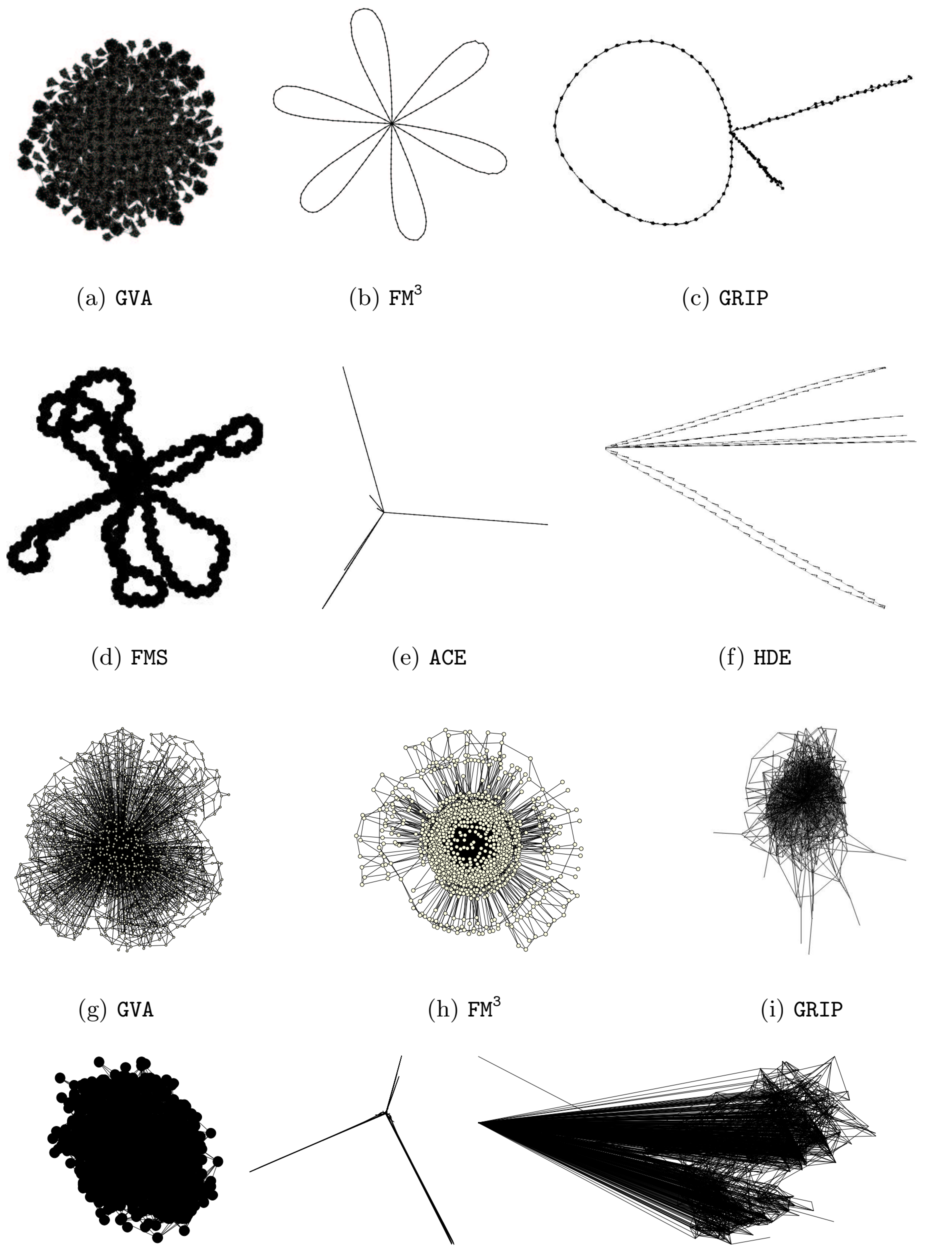

(a) GVA (b) FM^3 (c) GRIP

(d) FMS (e) ACE (f) HDE

(g) GVA (h) FM^3 (i) GRIP

(j) FMS (k) ACE (l) HDE

Fig. 4. (a)-(f) Drawings of snowflake_A and (g)-(l) spider_A generated by different algorithms.



(a) GVA (b) FM^3 (c) GRIP

(d) FMS (e) ACE (f) HDE

(g) GVA (h) FM^3 (i) GRIP

(j) FMS (k) ACE (l) HDE

Fig. 5. (a)-(f) Drawings of flower_B and (g)-(l) ug_380 generated by different algorithms.

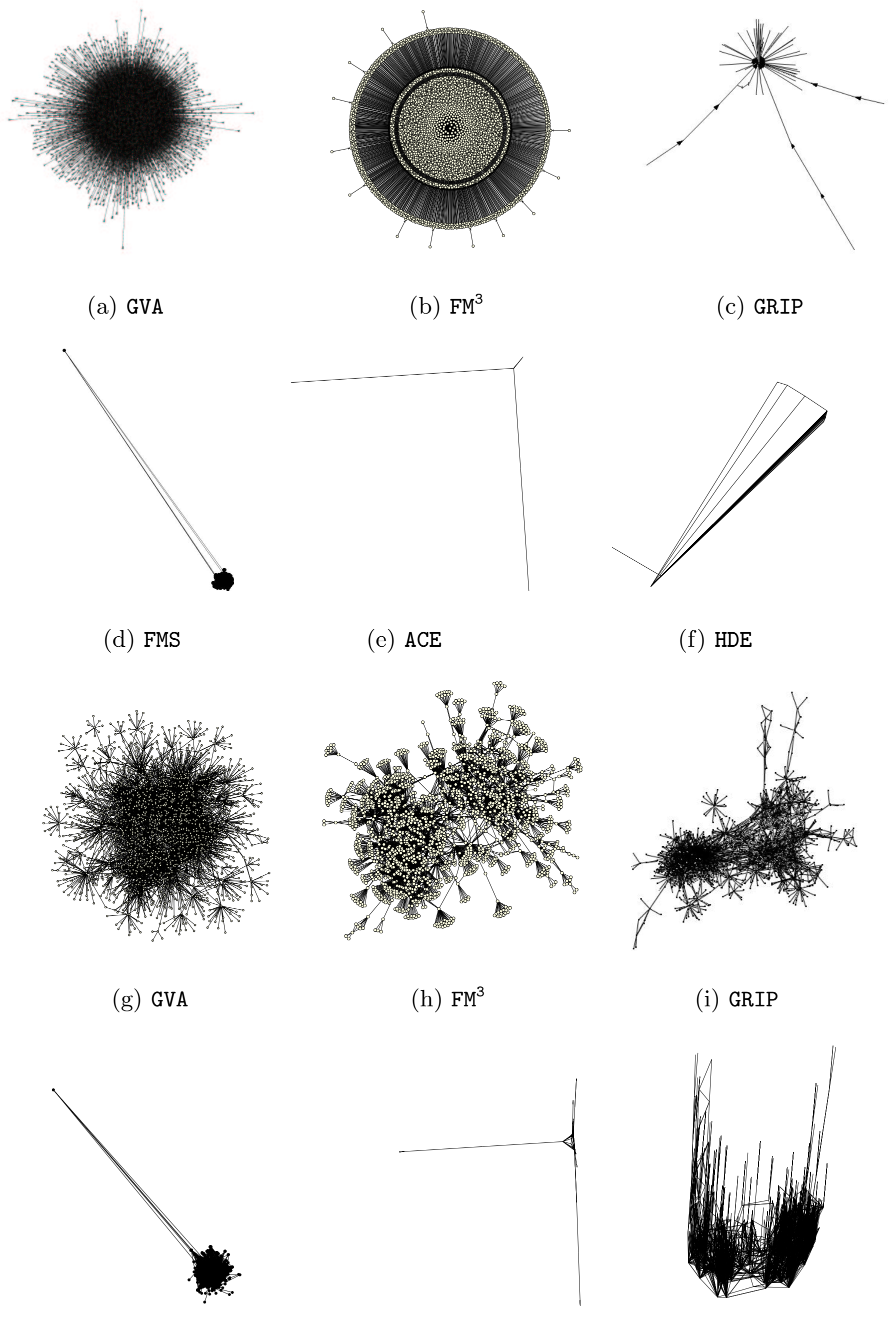


Fig. 6. (a)-(f) Drawings of dg_1087 and (g)-(l) esslingen generated by different algorithms

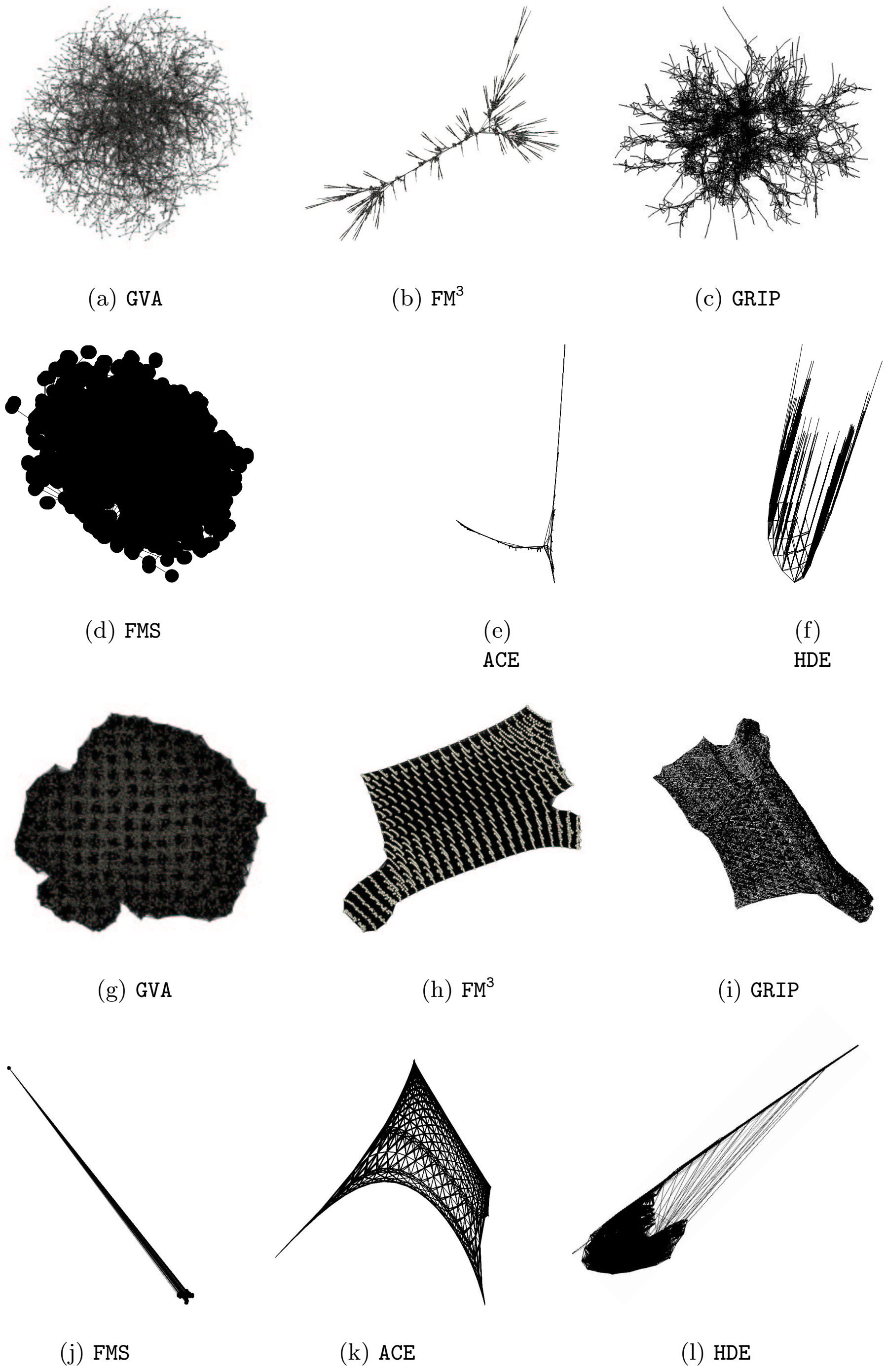


Fig. 7. (a)-(f) Drawings of add_32 and (g)-(l) bcsstk_33 generated by different algorithms.

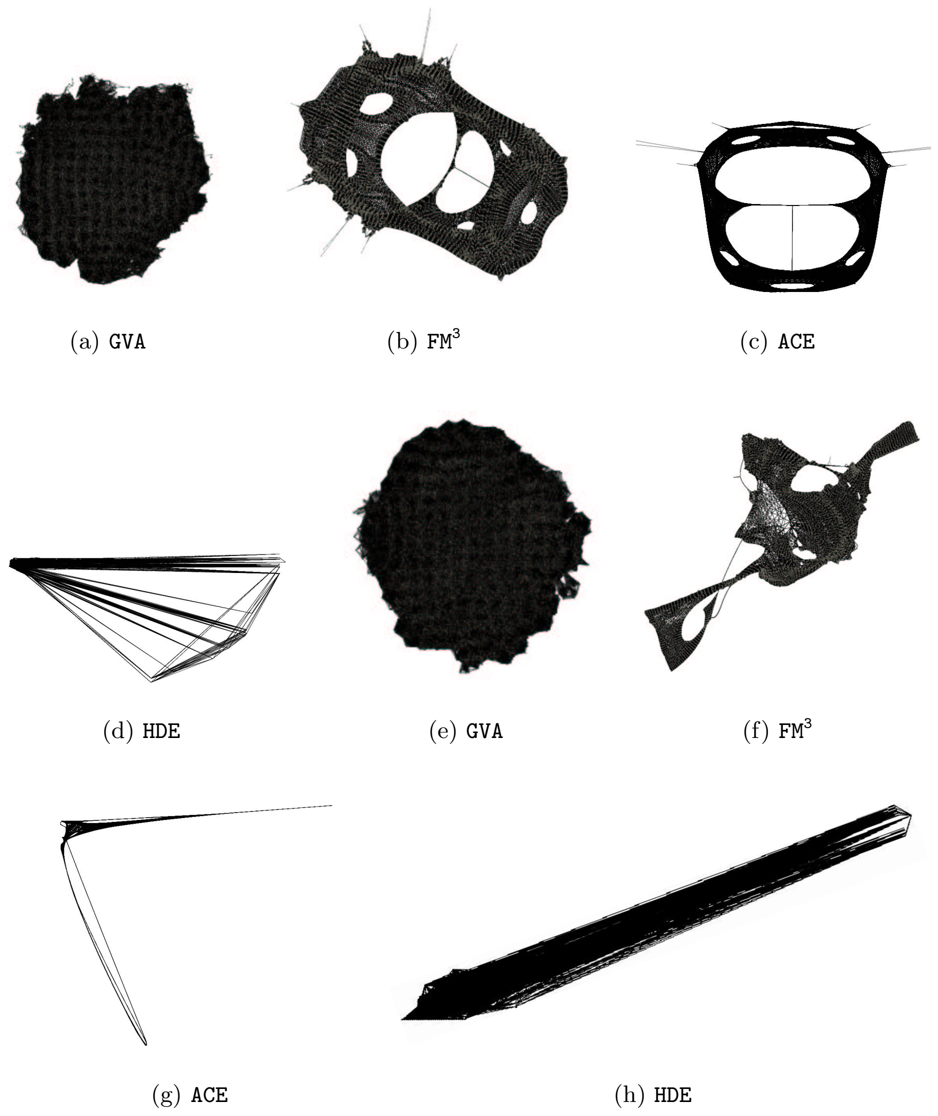


Fig. 8. (a)-(d) Drawings of bcsstk_31_con and (e)-(h) bcsstk_32 generated by different algorithms.