A Multilevel Algorithm for Force-Directed Graph Drawing

C. Walshaw

School of Computing & Mathematical Sciences, University of Greenwich, Park Row, Greenwich, London, SE10 9LS, UK. C.Walshaw@gre.ac.uk; http://www.gre.ac.uk/~c.walshaw

Abstract. We describe a heuristic method for drawing graphs which uses a multilevel technique combined with a force-directed placement algorithm. The multilevel process groups vertices to form *clusters*, uses the clusters to define a new graph and is repeated until the graph size falls below some threshold. The coarsest graph is then given an initial layout and the layout is successively refined on all the graphs starting with the coarsest and ending with the original. In this way the multilevel algorithm both accelerates and gives a more global quality to the force-directed placement. The algorithm can compute both 2 & 3 dimensional layouts and we demonstrate it on a number of examples ranging from 500 to 225,000 vertices. It is also very fast and can compute a 2D layout of a sparse graph in around 30 seconds for a 10,000 vertex graph to around 10 minutes for the largest graph. This is an order of magnitude faster than recent implementations of force-directed placement algorithms.

1 Introduction

Graph drawing is a basic enabling technology which can aid the understanding of sets of inter-related data by producing 'nice' layouts (a comprehensive survey can be found in [2]). Several layout algorithms are based on physical models and the vertices are placed so as to minimise the 'energy' in the physical system. Typically such algorithms are well able to display structures and symmetries in small graphs but can have very high runtimes.

1.1 Motivation

The motivation behind our approach to the problem arises from our work in the field of graph partitioning. In recent years it has been recognised that an effective way of both accelerating graph partitioning algorithms and, perhaps more importantly, giving them a global perspective is to use multilevel techniques. The idea is to match pairs of vertices to form *clusters*, use the clusters to define a new graph and recursively iterate this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (possibly with a crude algorithm) and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. This sequence of contraction followed by

repeated expansion/refinement loops is known as multilevel partitioning and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin and other optimisation algorithms, e.g. [10]. The multilevel process has also recently been successfully applied to the travelling salesman problem and appears to work (for combinatorial optimisation problems at least) by sampling and smoothing the objective function, [15], thus imparting a more global perspective to the optimisation.

In this paper we apply multilevel ideas to force-directed placement (FDP) algorithms. In fact such ideas have been previously suggested in the graph drawing literature and for example, Fruchterman & Reingold, [7], suggest the possible use of 'a multigrid technique that allows whole portions of the graph to be moved' whilst Davidson & Harel, [1], suggest a multilevel approach to 'expedite the SA [simulated annealing] process'. More recently Hadany & Harel, [8], and in particular Harel & Koren, [9], have actually used multilevel ideas (or as they refer to them, *multiscale*) in combination with an FDP algorithm and are able to robustly handle graphs of 3,000 vertices (although their algorithm still contains an $O(N^2)$ component). Their approach, although derived independently (and using a different FDP algorithm), shares many features with the algorithm outlined here and in many ways confirms that the multilevel paradigm can be a powerful tool for force directed placement.

A related but somewhat different idea is that of multilevel drawings, e.g. [3, 6]. Rather than using the multilevel process to create a good layout of the original graph, a multilevel graph is created, either by natural clustering which exists in the graph or by artificial means similar to those applied here. Each level is drawn on a plane at a different height and the entire structure can then be used to aid understanding of the graph at multiple abstraction levels, [5].

2 A multilevel algorithm for graph drawing

The multilevel FDP algorithm outlined here (and fully described in [14]) works by recursively coarsening the graph until its size falls below some threshold. The coarsest graph is then given an initial layout and the layout is successively refined on all the graphs starting with the coarsest and ending with the original. The algorithm does not actually operate simultaneously on multiple levels of the graph (as, for example, a multigrid algorithm might) but instead refines the layout at each level and then interpolates the result onto the next level down.

2.1 Contraction

Graph coarsening. Given a graph $G_l(V_l, E_l)$, there are many ways to create a coarser representation $G_{l+1}(V_{l+1}, E_{l+1})$ and clustering algorithms are an active area of research within the field of graph drawing, e.g. [3, 12]. Often such clustering algorithms seek to retain the more important structural features of the graph in order that the visualisation of each level is meaningful in itself. However, here we are only interested in the drawing of the original graph. As such we seek a

fast and efficient (i.e. not necessarily optimal) algorithm that coarsens gradually (aggressive clustering may depreciate the benefits of the multilevel paradigm) and uniformly (the coarsening should not change the inherent properties of the graph differently between different regions).

To suit these requirements we use a coarsening approach known as *matching* in which vertices are matched with at most 1 neighbour so that clusters are thus formed of at most 2 vertices. Computing a matching is equivalent to finding a maximal independent subset of graph edges which are then collapsed to create the coarser graph. The set is independent if no 2 edges in the set are incident on the same vertex (so no 2 edges in the set are adjacent), and maximal if no more edges can be added to the set without breaking the independence criterion. Having found such a set, each selected edge is collapsed and the vertices, $u_1, u_2 \in V_l$ say, at either end of it are merged to form a new vertex $v \in V_{l+1}$ with weight $|v| = |u_1| + |u_2|$.

The problem of computing a matching of the vertices is known as the maximum cardinality matching problem. Although there are optimal algorithms to solve this problem, they are of at least $O(N^{2.5})$, e.g. [11]. Unfortunately this is too slow for our purposes and, since it is not too important for the multilevel process to solve the problem optimally, we use a variant of the edge contraction heuristic proposed by Hendrickson & Leland, [10]. Their method of constructing a matching is to create a randomly ordered list of the vertices and visit them in turn, matching each unmatched vertex with an unmatched neighbouring vertex (or with itself if no unmatched neighbours exist). Matched vertices are removed from the list. If there are several unmatched neighbours the choice of which to match with can be random, but in order to keep the coarser graphs as uniform as possible, and after some experimentation, we choose to match with the neighbouring vertex with the smallest weight (note that even if the original graph G_0 is unweighted, G_l for $l > 0$ will be weighted).

The initial layout. Having constructed the series of graphs until the number of vertices in the coarsest graph is smaller than some threshold, the normal practice of the multilevel partitioning strategy is to carry out an initial partition. In terms of graph drawing the analogue is to compute the initial layout. However, if the graph is coarsened down to 2 vertices (which, because of the mechanisms of the coarsening, will be connected by a single weighted edge) we can simply place these vertices at random with no loss of generality. Note that contraction down to 2 vertices should always be possible provided the graph is connected, [14].

Layout interpolation. Having refined the layout on a graph G_l, it is interpolated onto its parent G_{l-1}. The interpolation itself is a trivial matter and matched pair of vertices, $v_1, v_2 \in V_{l-1}$, are placed at the same position as the cluster, $v \in V_l$, which represents them.

2.2 The force-directed placement algorithm

At each level we use a force-directed placement (FDP) or spring-embedder algorithm to draw the graph, G_l, and more importantly to provide initial positions

for the parent graph G_{l-1}. The original FDP concept came from a paper by Eades, [4], and is based on the idea of replacing edges by springs. The vertices are given initial positions, usually random, and the system is released so that the springs move the vertices to a minimal energy state (i.e. so that the springs are compressed or extended as little as possible).

Unfortunately these local spring forces are insufficient to globally untangle a graph and so such algorithms also employ global repulsive forces, calculated between every pair of vertices in the graph, and thus the system resembles an n-body problem. Such repulsive forces between non-adjacent vertices do not have an analogue in the spring system but are a crucial part of spring-embedder algorithms to avoid minimal energy states in which the system is collapsed in on itself in some manner.

The particular variant of force-directed placement that we use is based on an algorithm by Fruchterman & Reingold (FR), [7], itself a variation of Eades' original algorithm. From the point of view of the multilevel approach it is attractive as it is an incremental scheme which iterates to convergence and which can reuse a previously calculated initial layout. We have made a number of modifications based on our experience with it and, in particular, because of the additional problems associated with drawing very large graphs. Space precludes a full description of the algorithm here but we describe the implementation in full in [14]. In principle however, it should be possible to use any iterative incremental algorithm for this part of the multilevel graph drawing, although in practice different algorithms can be somewhat sensitive and require tuning.

Natural spring length, k. A crucial part of the algorithm the choice of the natural spring length k_l (the length at which a spring is neither extended nor compressed). For the initial coarsest graph, G_L, the 2 vertices are placed at random and k_L set to be the distance between them. However at the start of the execution of the placement algorithm for graph G_l $(l < L)$ the vertices will all be in positions determined by the layout calculated for graph G_{l+1}. We must therefore somehow set k_l relative to this existing layout in order not to destroy it. If k_l is too large, then the entire graph will have to expand from its current layout and potentially ruin any advantage gained via the multilevel process.

In fact we derive the new value for k by considering the coarsening a graph, G_l, with well placed vertices (i.e. all vertices are approximately at a distance k from each other) and in [14] justify our choice of $k_l = \sqrt{4/7} \times k_{l+1}$. Remarkably this simple formula works very robustly over all the examples that we have tested although we feel that this parameter could do with further investigation.

2.3 Reducing the complexity

Unfortunately the complexity of the FR algorithm for each iteration on graph $G_l(V_l, E_l)$ is $O(|V_l|^2 + |E_l|)$. For the types of sparse graphs that we are interested in, the $|V_l|^2$ component heavily dominates this expression and we therefore use the FR grid variant for reducing the run-times. Their motivation was that long distance repulsive forces are sufficiently small enough to be neglected. If we set R to be the maximum distance over which repulsive forces will act we can

then modify the algorithm by ignoring global forces between any pair of vertices further apart than R (and this can efficiently implemented using a superimposed grid to avoid calculating the distance between every pair of vertices).

In the original FR algorithm the value $R = 2k$ was used, but for the larger graphs that we are interested in this did not prove sufficient to 'untangle' them globally. Unfortunately the larger the value given to R the longer the algorithm takes to run and so although assigning $R = 20k$ gave better results, it did so with a huge time penalty. Fortunately, however, the power of the multilevel paradigm comes to our aid once again and we can make R a function of the level l. Thus for the initial coarse graphs we can set R_l to be relatively large and achieve some impressive untangling without too much cost (since $|V|$ is very small for these graphs). Meanwhile, for the final large graphs, when most of the global untangling has already been achieved we can make R_l relatively small without penalising the placement. In fact the first such schedule that we tried, with $R_l = 2(l+1)k_l$ for each graph G_l, worked so well that we have not experimented further. This also replicates the choice of $R = 2k$ for G_0 in the original FR algorithm.

2.4 Complexity analysis

It is not easy to derive complexity results for the algorithm but we can state some bounds. Firstly the number of graph levels, L, is dependent on the rate of coarsening. At best the number of vertices will be reduced by a factor of 2 at every level (if the code succeeds in matching every vertex with another one) and in the worst case, the code may only succeed in matching 1 vertex at every level (e.g. if the graph is a star graph, a 'hub' vertex connected to every other vertex each of which is only connected to the hub). Thus we have $\log_2 |V| \leq L < |V|$. This indicates that the algorithm is not well suited to drawing star type graphs and in fact for the examples given in Section 3 the coarsening rate is close to 2.

The matching & coarsening parts of the algorithm are $O(|V_l| + |E_l|)$ for each level l but in fact the total runtime is heavily dominated by the FDP algorithm. Using the above simplification (§2.3) of neglecting long range repulsive forces we can see that each iteration of the FDP algorithm is bounded below by $O(|V_l| + |E_l|)$ although with a large coefficient. In fact if the graph is very dense, or in the worst case a complete graph, it may be that this is still $O(|V_l|^2 + |E_l|)$, dependent on the relative balance of attractive & repulsive forces. However, we suspect that no FDP algorithm is appropriate for very dense graphs (because the minimal energy state corresponds to a tightly packed 'blob').

The number of FDP iterations at every level is determined by the cooling schedule which sets t^i, the temperature at each iteration, to $t^i = (1 - \epsilon)t^{i-1}$. In the experiments below (and in [14]) we use an initial temperature $t^0 = k_l$ and $\epsilon = 0.1$ and the algorithm is deemed to have converged when all movement is less than $0.001k_l$ which means that all movement ceases at iteration i where $(1 - 0.1)^i < 0.001$ or in other words after 66 iterations.

In summary the total complexity at each level is close to $O(|V_l| + |E_l|)$ for sparse graphs and the runtime is heavily dominated by the FDP iterations.

Finally consider the FDP algorithm used, without coarsening, on a given sparse graph of size N (i.e. standard single-level placement (SLP)) and compare it with multilevel placement (MLP) used on the same graph. Let T_p be the time for the SLP algorithm to run on the graph and for MLP let T_c be the time to coarsen and contract it. If we suppose that the coarsening rate is close to 2 (which is true for the examples below) then for MLP this gives us a series of problems of size $N, N/2, \ldots, N/N$ whilst the (almost) linear complexity for FDP gives the total runtime for MLP as $T_c + T_p/N + \ldots + T_p/2 + T_p$. In all the example we have tested $T_c \ll T_p$ and so we can neglect it giving a total runtime of approximately $T_p/N + \ldots + T_p/2 + T_p = 2T_p$. In other words MLP should take *only* twice as long as SLP to run (and yet in the extended example below, §3.1, achieves far better results). In fact the final level of the MLP algorithm is likely to already have a very good initial layout which means that it should run even faster than SLP although this is neutralised somewhat by the fact that the coarsening rate is normally somewhat less than 2. Nonetheless this factor of 2 is a good 'rule of thumb' and note that if the chosen FDP algorithm were $O(N^2)$ or even $O(N^3)$ then a similar analysis suggests that the MLP runtime is substantially *less* than twice that of SLP.

3 Examples

We have implemented the algorithms described here within the framework of JOSTLE, a mesh partitioning software tool developed at Greenwich. The experiments were carried out on a Sun SPARC Ultra 10 with a 333 MHz CPU and 256 Mbytes of memory.

We have tested our multilevel algorithm on a number of example graphs, including some for which we already know a good layout (although interestingly, even the results on these graphs proved very illuminating). Most such graphs were drawn from genuine examples of meshes from various computational mechanics problems. Typically in such graphs the vertices either represent mesh nodes (the nodal graph) or mesh elements (the dual graph). However we have also considered graphs from other non mesh-based applications.

Table 1 gives a summary of some examples showing their sizes ($|V|$ & $|E|$), the maximum, minimum & average degree of the vertices, the MLP runtime and a short description. In all cases the MLP algorithm produced a good layout although sometimes with a certain amount of buckling (see [14]). However for none of these examples was the single-level FDP algorithm able to find a layout which untangled the graph in a global sense (despite adjusting the parameters and allowing considerably longer runtimes). For the shuttle, add32 & mesh100 graphs the layout was calculated in 3D and hence the runtime is a little higher than a 2D layout (e.g. compare the add32 & 4970 runtimes).

The second half of the table shows the results for a series of increasingly larger versions of the same mesh (created by adaptive mesh refinement) and gives an indication of the scalability of the algorithm (albeit on sparse & fairly

Table 1. A summary of the example graphs

graph	size $\|V\|$	$\|E\|$	degree max	min	avg	runtime (secs.)	graph type
c-fat500-10	500	46627	188	185	186.51	12.53	max clique test
516	516	729	3	1	2.83	1.14	2D dual
shuttle	2851	15093	17	3	10.59	20.05	3D nodal
add32	4960	9462	31	1	3.82	48.09	electronic circuit
4970	4970	7400	3	2	2.98	13.93	2D dual
whitaker3	9800	28989	8	3	5.92	24.10	2D nodal
finan512	74752	261120	54	2	6.99	688.46	linear programming
sierpinski10	88575	177147	4	2	4.00	217.58	2D self-similar 'fractal'
mesh100	103081	200976	4	2	3.90	1245.43	3D dual
Laplace.0	23787	35281	3	2	2.97	64.16	2D dual
Laplace.2	40851	60753	3	2	2.97	114.84	2D dual
Laplace.5	88743	132329	3	2	2.98	247.52	2D dual
Laplace.8	185761	277510	3	2	2.99	477.36	2D dual
Laplace.9	224843	336024	3	2	2.99	598.87	2D dual

homogeneous graphs). This supports the complexity analysis in §2.4 that the runtime is approximately linear in $|V| + |E|$.

3.1 An extended example

In this section we demonstrate in a little more detail how the multilevel placement (MLP) algorithm works for the 516 graph. The algorithm first coarsens the problem (reducing the number of vertices by a factor of around 1.8 at each level) and constructs a series of 9 graphs of diminishing size. The initial layout is computed by placing the 2 vertices of G_9 at random and setting the natural spring length, k, to be the distance between them. Starting from $G_l = G_8$ the layout is interpolated from G_{l+1}, by simply placing vertices at the same position as the cluster representing them in the coarser graph, and then refined.

Figure 1(a) shows the final layout on G_4 and although over 10 times smaller than the original, the layout is already beginning to take shape. Figures 1(b)-1(d) meanwhile illustrate the placement algorithm on G_2. Figure 1(b) shows the initial layout as calculated on G_3 and with many of the vertices coincident whilst Figure 1(c) then shows the layout after the first iteration and where the coincident vertices have started to separate. Figure 1(d) finally shows the layout after the placement algorithm has converged for G_2. Notice an important feature of the multilevel process, common with partitioning, that on each level the final layout (partition), does not differ greatly from the initial one. Figure 1(e) shows the final layout on the original graph, G_0. The entire runtime of the MLP algorithm to compute this layout was just over 1 second.

For comparison, Figure 1(f) shows the single-level FDP algorithm used on a random initial layout. Possibly the algorithm is not well tuned for this prob-

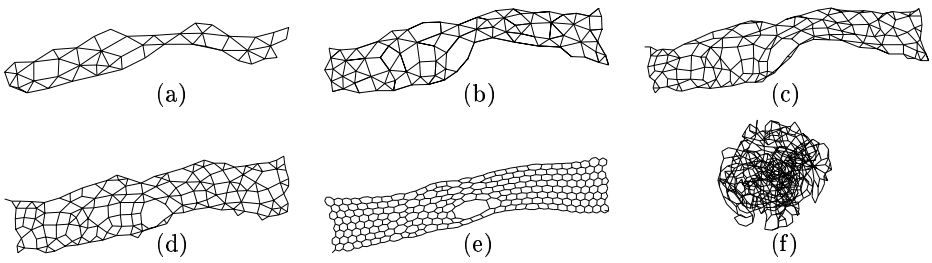

Fig. 1. The multilevel force directed placement illustrated for the mesh 516

lem (although the initial temperature was raised to $10k$), but it is clear that, although the micro structure has been reconstructed reasonably well, the single-level placement has not been able to 'untangle' the graph in a global sense.

3.2 Example layouts

Space precludes any extended presentation of the examples in Table 1, but Figures 2-7 show some of the highlights (see also [14]). Despite the suggestion that the MLP algorithm is best suited to sparse graphs (§2.4), Figure 2 shows a dense regular graph, c-fat500-10 (generated to test algorithms for the maximum clique problem), and demonstrates that the layout nicely captures the symmetries. Meanwhile Figure 3 shows the layout calculated for the 4970 graph by the MLP algorithm where, in trying to equalise the edge lengths, the drawing has actually revealed far more of the graph than the original layout. Figure 4 shows the 3D layout of the shuttle graph which which reveals 3 weakly connected 'panels', the existence of which is not evident in the original layout. Meanwhile Figure 5 shows the layout calculated for a large Sierpinski graph, a self-similar 'fractal' type structure which has large holes. Finally, in the centre-piece of [14], Figure 6 shows the layout found by the MLP algorithm of finan512, a linear programming matrix with around 75,000 vertices. Once again this layout is highly illuminating; the graph is revealed to have a fairly regular structure and consists of a ring with 32 'handles' each of which has a number of fronds protruding. Figure 7 shows a detailed view of one of these handles.

4 Summary and further research

We have described a multilevel algorithm for force-directed graph drawing which coarsens the graph, refines the layout at each level and then interpolates the result onto the next level down. The algorithm is fast, e.g. about 1 second for 2D layout of a 500 vertex sparse graph and about 10 minutes for 225,000 vertices. This is an order of magnitude faster than recent implementations of force-directed placement (e.g. around 70 seconds for a 1,000 vertex graph in [13])

and indeed it is not even clear whether current single-level algorithms can produce reasonable layout for such large graphs. As such the multilevel paradigm broadens the scope of force-directed algorithms.

We have not particularly tried to address graphs for which the technique might not work. It is likely that very dense graphs or even those which have a dense substructure are never going to be good candidates for any FDP algorithm and ours is no exception. It is also likely that graphs containing vertices of very high degree may not particularly suit the coarsening process (see §2.4). However we believe that the multilevel process can accelerate & enhance FDP algorithms for a range of useful graphs and further testing on different types of graph is an important subject for further research. We have not addressed disconnected graphs but feel that this requires only minor modifications.

References

1. R. Davidson and D. Harel. Drawing Graphs Nicely using Simulated Annealing. *ACM Trans. Graphics*, 15(4):301–331, 1996.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, New Jersey, U.S.A., 1998.
3. C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Planarity-Preserving Clustering and Embedding for Large Planar Graphs. In J. Kratochvíl, editor, *Proc. 7th Int. Symp. Graph Drawing*, volume 1731 of *LNCS*. Springer, 1999.
4. P. Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149–160, 1984.
5. P. Eades and Q. Feng. Multilevel Visualization of Clustered Graphs. In *Proc. 6th Int. Symp. Graph Drawing*, volume 1190 of *LNCS*, pages 101–112. Springer, 1996.
6. P. Eades, Q. Feng, X. Lin, and H. Nagamochi. Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. Tech. rep. 98-03, Dept. Comp. Sci., Univ. Newcastle, Callaghan 2308, Australia, 1998.
7. T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-Directed Placement. *Software — Practice & Experience*, 21(11):1129–1164, 1991.
8. R. Hadany and D. Harel. A Multi-Scale Algorithm for Drawing Graphs Nicely. Tech. Rep. CS99-01, Weizmann Inst. Sci., Faculty Maths. Comp. Sci., Jan, 1999.
9. D. Harel and Y. Koren. A Fast Multi-Scale Algorithm for Drawing Large Graphs. Tech. Rep. CS99-21, Weizmann Inst. Sci., Faculty Maths. Comp. Sci., Nov, 1999.
10. B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In S. Karin, editor, *Proc. Supercomputing '95*. ACM Press, 1995.
11. C. H. Papadimitriou and K. Stieglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.
12. R. Sablowski and A. Frick. Automatic Graph Clustering. In *Proc. 6th Int. Symp. Graph Drawing*, volume 1190 of *LNCS*, pages 395–400. Springer, 1996.
13. D. Tunkelang. JIGGLE: Java Interactive General Graph Layout Environment. In S. H. Whitesides, editor, *Proc. 6th Int. Symp. Graph Drawing*, volume 1547 of *LNCS*, pages 413–422. Springer, 1998.
14. C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. Tech. Rep. 00/IM/60, Univ. Greenwich, London SE10 9LS, UK, April 2000.
15. C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. Tech. Rep. 00/IM/63, Univ. Greenwich, London SE10 9LS, UK, Aug. 2000.

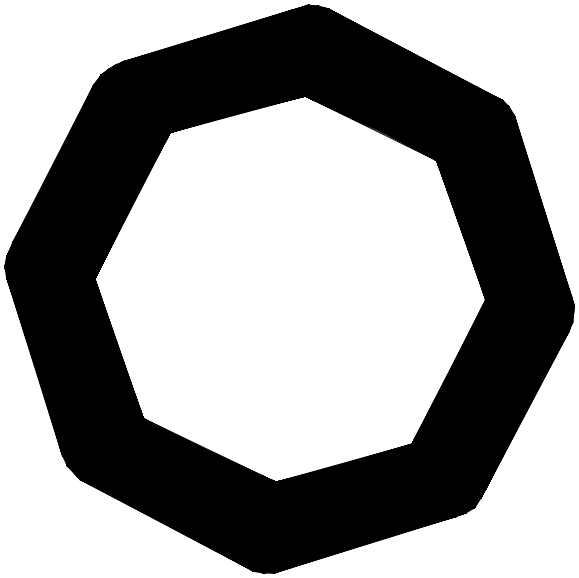

Fig. 2. The layout of c-fat500-10 computed with the multilevel placement algorithm

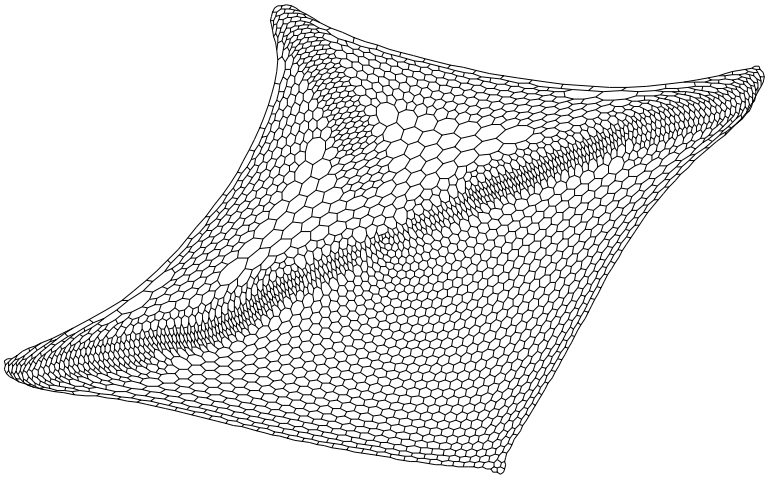

Fig. 3. The layout of 4970 computed with the multilevel placement algorithm

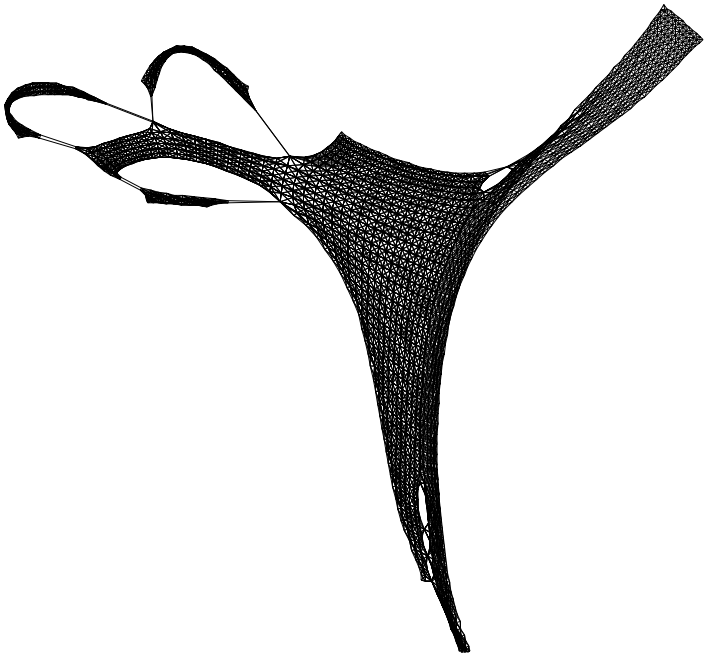

Fig. 4. The layout of shuttle computed with the multilevel placement algorithm

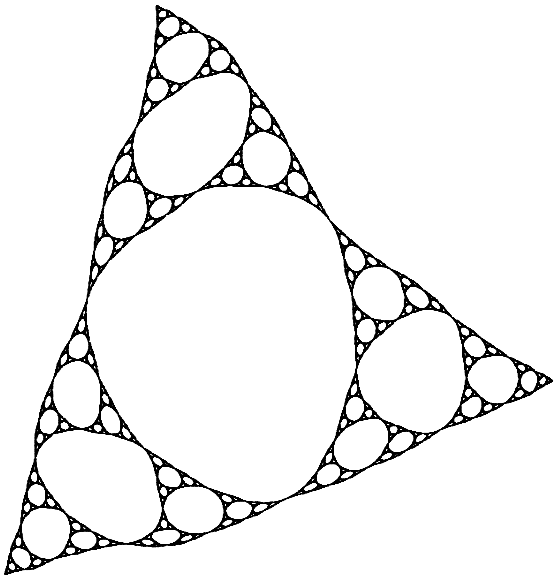

Fig. 5. The layout of sierpinski10 computed with the multilevel placement algorithm

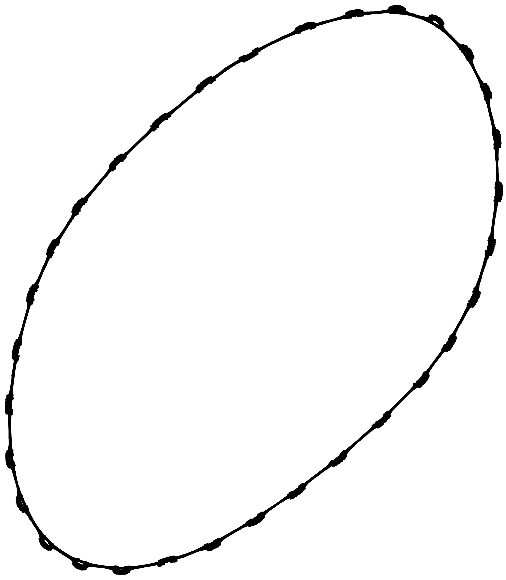

Fig. 6. The layout of finan512 computed with the multilevel placement algorithm

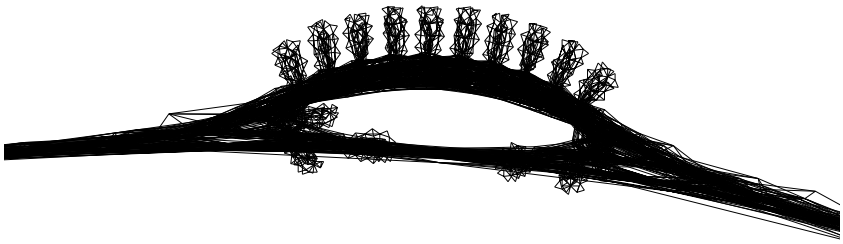

Fig. 7. finan512 computed with the multilevel placement algorithm: detail of the micro structure