A Fast Adaptive Layout Algorithm for Undirected Graphs (Extended Abstract and System Demonstration)

Arne Frick*, Andreas Ludwig, Heiko Mehldau

Universität Karlsruhe, Fakultät für Informatik, D-76128 Karlsruhe, Germany

Abstract. We present a randomized adaptive layout algorithm for nicely drawing undirected graphs that is based on the spring-embedder paradigm and contains several new heuristics to improve the convergence, including local temperatures, gravitational forces and the detection of rotations and oscillations. The proposed algorithm achieves drawings of high quality on a wide range of graphs with standard settings. Moreover, the algorithm is fast, being thus applicable on general undirected graphs of substantially larger size and complexity than before [9, 6, 3]. Aesthetically pleasing solutions are found in most cases. We give empirical data for the running time of the algorithm and the quality of the computed layouts.

1 Introduction

The problem of obtaining an aesthetically pleasing drawing of a given graph $G = (V, E)$ is receiving increasing attention in the literature [2, 1]. One way of dealing with this problem is the construction of *straight-line drawings*, in which each edge is mapped into a straight-line segment in the plane. The problem then reduces to the problem of *positioning* the vertices $v \in V$ by determining a mapping $\phi : V \rightarrow \mathbb{R}^2$.

There are many criteria for judging the aesthetics of a graph drawing [12]. Among the most influential are the display of symmetries existing in the graph and the minimization of edge crossings. Furthermore, edges should have as few bends as possible, and the deviation on their lengths should be small. The area used for drawing should be as small as possible, while the vertices and edges should be evenly distributed in the area. Connected vertices should be close to each other.

Straight-line drawings avoid bends in edges by definition. The remaining criteria, however, cannot be matched optimally in polynomial time unless $\mathcal{P} = \mathcal{NP}$. Therefore, only good approximations to an optimal solution appear to be feasible given the current state of the art. Even more so, simultaneous optimization for several criteria can involve quality tradeoffs as there exist incompatible combinations. Examples and references are given in the full paper. After introducing some notation in Sect. 2, we briefly describe heuristics to obtain good approximations to the simultaneous optimization problem in Sect. 3.

We continue with a detailed discussion of the proposed algorithm called GEM in Sect. 4. Empirical data on the convergence and the time complexity of the proposed

* EMail: frick@informatik.uni-karlsruhe.de

algorithm on different types of graphs is presented in Sect. 5. The results are compared to those for publicly available implementations of other force-directed algorithms [9, 6].

We conclude with some suggestions for further investigation before we complement the numerical data with graphical output from the GEM algorithm. Appendix A describes GEMDRAW, the algorithm development environment, while App. B contains drawings for several types of graphs, including well-known ones from the literature.

2 Notation

Throughout this paper, we use the following notation for graph sizes and density. We shall use the following conventions to classify graphs into different groups. Graphs of size $|V| \leq 16, 32, 64, 128$ and $V \geq 128$ are called *tiny*, *small*, *medium*, *large*, *huge*, respectively. We acknowledge that this partition is somewhat arbitrary, but it does reflect the fact that almost all methods discussed below focus on tiny and small graphs, while few handle medium-sized sparse graphs, not to mention large or huge ones. We further distinguish between *sparse*, *normal* and *dense* graphs ($|E| < |V|, |V| \leq |E| < 3|V|, |E| \geq 3|V|$). This is motivated by the fact that trees should be considered sparse, while meshes, tori and hypercubes of small dimension represent the normal case.

3 Previous Approaches

The known heuristics for constructing a straight-line drawing of an undirected graph can be classified into three groups according to the computational model employed.

The *spring embedder model* for drawing undirected graphs is due to Eades [4]. It generalizes previously known algorithms for the layout of PCB's [5, 8]. Using an analogy to physics, vertices are treated as mutually repulsive charges and edges as springs connecting and attracting the charges. Starting with an arbitrary initial placement of vertices, the algorithm iterates the system in discrete time steps by computing the forces between vertices and updating their position accordingly. The algorithm stops after a fixed number of time steps. An obvious drawback with this approach is that the mass-spring system may not have converged after the fixed number of steps while on the other hand, time is wasted unnecessarily if this number is chosen too large. Although the algorithm does not explicitly support the detection and display of symmetries, it turned out to display symmetries if any exist. Kamada and Kawai [9] refined the model subsequently. They introduce a *optimal edge length* k. Vertices are updated sequentially by moving only one vertex at each time step. The algorithm performs a *gradient descent* and converges deterministically to a local minimum. The time complexity of the approach cannot be expressed in terms of $|V|$ and $|E|$.

In optimization theory, researchers have introduced randomness to overcome the problem of ending up in local minima. They use a technique from statistical mechanics called *simulated annealing* [13, 10] allowing for changes into states with higher energy. An arbitrary state change is computed. Any downhill move is accepted, while "uphill moves" are accepted with a probability depending on a current *temperature*. Initially the system has the ability to perform arbitrary moves because the temperature is still

high. Later, the probability of choosing a next state with more energy approaches zero as the temperature is lowered. Davidson and Harel [3] employ simulated annealing to achieve aesthetically pleasing results on small and medium-sized graphs. The approach is flexible in that it can easily be adapted to incorporate other quality measures or weights in the cost function, but unfortunately it is very slow. Independently, Fruchterman and Reingold [6] modified Eades' algorithm by introducing a simple cooling schedule. Their algorithm is deterministic in that it only performs local optimizations. The distance a vertex can travel at a given time is limited depending on the current temperature.

The third group of heuristics is based on preprocessing of the graph to get a good initial placement [15, 14]. The full paper contains reviews of these.

4 The GEM algorithm

The algorithm proposed in this paper is called GEM (short for *graph embedder*). It contains several novel algorithmic ideas. These include the concept of a local temperature, the attraction of vertices towards their barycenter and the detection of oscillations and rotations.

The major design goal was that interactive speed should be achieved even for medium-sized graphs. We consider a drawing to be interactive if it takes less than $2\ s$ to compute. To this end, we designed GEM to rely on fast integer arithmetic.

We hypothesized that randomization works in the domain of force-directed graph drawing. Therefore, randomization plays an important role in several places of the algorithm.

The discussion of the GEM algorithm starts with the observation that cooling schedules appear to give better results than methods relying solely on a gradient descent, but their running time is unsatisfactory. Temperatures as used in GEM indicate the maximum distance a vertex can travel when being updated. The temperature scale has a direct influence on a suitable choice of other parameters, i.e. the constants used in the formulae for the attractive and repulsive forces.

Fruchterman and Reingold [6] conjectured that better cooling schedules should allow much more efficient algorithms than theirs. Davidson and Harel [3] first suggested an adaptive cooling schedule but did not explore on the idea further. As to our knowledge, the algorithm proposed in this paper is the first approach using this idea, although we do not employ a cooling schedule in the strict sense. Rather, the algorithm adapts to the data locally and does not require global cooling as assumed by a schedule. For each vertex, a *local temperature* is defined that depends on its old temperature and the likelihood that the vertex oscillates or is part of a rotating subgraph. Local temperatures raise if the algorithm determines that a vertex is probably not close to its final destination. The *global temperature* is defined as the average of the local temperatures over all vertices. Thus, it indicates how stable the drawing of the graph is.

The proposed algorithm consists of two stages, an initialization stage and an iteration stage. The initialization consists of the assignment of an initial position, impulse and temperature to each vertex. The main loop updates vertex positions until the global temperature is lower than a desired minimal temperature or the time allowance has expired. An abstract algorithm for GEM is depicted in Fig. 1. Although we found in practice that

most graphs would easily cool down to $T_{\min}$, we cannot exclude the possibility of a graph moving chaotically between rounds. This can be solved by choosing $T_{\min}$ larger.

```
procedure GEM is                                                          1
    -- Input:                                                             2
    --     G = (V, E) graph where                                         3
    --     V = set of record                                              4
    --           ξ  -- current position                                   5
    --           p  -- last impulse                                       6
    --           t  -- local temperature                                  7
    --           d  -- skew gauge                                         8
    --     R_max    maximal number of rounds [4|V|]                       9
    --     T_max    upper bound on local temperature [256]                10
    --     T_min    desired minimal temperature [3]                       11
    -- Output: for each v ∈ V, a position is computed                     12
                                                                          13
    forall v ∈ V do                                                       14
      initialize v                                                        15
    while T_global > T_min and #rounds < R_max do                         16
      choose a vertex v to update;                                        17
      compute v's impulse;                                                18
      update v's position and temperature;                                19
    end; -- GEM                                                           20
```

Fig. 1. Main loop of the GEM algorithm

Vertices are moved sequentially according to a *choice function*. Assuming that vertex v was chosen to be updated, the attractive and repulsive forces acting on v are computed. In addition, a *gravitational force* pulling the vertex towards the barycenter of the vertex cluster is assumed. The use of gravitation accelerates the convergence of GEM. In addition, it helps to keep disconnected graphs and loosely connected components together.

The resulting force is scaled with v's current temperature to form the *impulse* of v such as to reflect the algorithm's "knowledge" of the state of computation. A low temperature indicates either that the layout is almost stable (at least locally) or that there exist oscillations or rotations. In each case, movements should be short. GEM has the ability to leave wells containing local energy minima, since local temperature increases change the global energy distribution. Relative to the old distribution, uphill moves become possible. Unfortunately, this feature makes proofs of convergence hard, if not impossible.

We now discuss in turn the initialization stage (Sect. 4.1), the choice mechanism (Sect. 4.2), the impulse computation (Sect. 4.3) and the adjustment of the local temperature (Sect. 4.4).

4.1 Initialization

Every vertex is initialized with a zero impulse vector $\mathbf{p} = \mathbf{0}$, a direction skew gauge $d = 0$ and an initial temperature T_{init}. In general, we can confirm the claim of Kamada/Kawai [9] that initial positions do not have great influence on the resulting pictures but on the number of rounds to be performed to reach equilibrium. A random initial placement usually suffices for convergence, but we observed that certain structures like binary trees and meshes may gain from the computation of an initial placement by inserting the vertices one-by-one in an initial round.

4.2 Choice of a vertex to update

GEM uses a sequential update strategy, i.e. a single vertex is updated in each iteration. Our hypothesis suggests a random choice. We found indeed that GEM converges faster with a random choice mechanism than with a deterministic schedule.

To achieve a random selection, we proceed as follows. Iterations are grouped into *rounds*. At the beginning of each round, a random permutation is determined according to which the vertices are chosen. The complexity of choosing a permutation can be amortized to $\mathrm{O}(1)$ per iteration as there is an $\mathrm{O}(n)$ algorithm to compute a random permutation on n elements[11].

4.3 Impulse Computation

GEM memorizes the last movement for each vertex. In analogy to physics, we refer to this information as the *impulse* of the vertex. The computation of the impulse is governed by several global constants, a *desired edge length*[2] and a gravitational constant factor γ determining how strongly a vertex is driven towards the barycenter[3] of the current layout. This additional attractive force has two important effects: unconnected and loosely connected components are not separated too far, and it may lead to a 30% increase in convergence speed.

The algorithm for computing the new impulse of a single vertex v is given in Fig. 2. The function Φ mentioned therein is a scaling factor giving vertices with many edges more inertia. This improves the layout quality in some cases, e.g. Fig. 18.

4.4 Temperature adjustment

After computing the current impulse for vertex v, its position is updated. If v's impulse was non-negligible, we update v's internal data structures (see Fig. 3).

A new local temperature for v is computed based on the last temperature, the last and current movement and the skew gauge $v.d$ indicating the likeliness of v oscillating or being part of a rotation. The detection of rotations and oscillations requires knowledge of $\sin(\beta)$ and $\cos(\beta)$ where $\beta = \angle \mathbf{p}, v.\mathbf{p}$.

[2] This is the same as the "optimal" edge length of [6].

[3] It is important to use the barycenter as opposed to the center of the layout area, as the latter would force a finished graph to move until its barycenter coincides with the center of the layout area.

```
-- Input:                                                                    1
--   v        vertex to be updated                                          2
--   c        sum_u u.xi; the barycenter of G is computed as c/|V|          3
--   Phi      function growing with deg(v)[1 + deg(v)/2]                    4
-- Output:                                                                   5
--   p        current impulse of v                                          6
-- Constants and Functions:                                                  7
--   E_des    desired edge length [128]                                     8
--   gamma    gravitational constant [1/16]                                 9
                                                                           10
-- attraction to center of gravity                                         11
p:=(c/|V| - v.pos) · gamma · Phi(v);                                       12
-- random disturbance                                                      13
delta:=small random vector;  -- default range: [-32, 32] x [-32, 32]       14
p:=p + delta;                                                              15
forall u ∈ V \ {v} do                                                      16
    -- repulsive forces                                                    17
    Delta:=v.xi - u.xi;                                                    18
    if Delta ≠ 0 then p:=p + Delta · E_des^2/|Delta|^2;                    19
forall (u, v) ∈ E do                                                       20
    -- attractive forces                                                   21
    Delta:=v.xi - u.xi;                                                    22
    p:=p - Delta · |Delta|^2/(E_des^2 · Phi(v));                           23
```

Fig. 2. Impulse computation

Vertex rotations occur for example if the final layout has been found, but the temperature is still too high. Under rare circumstances, a graph in rotation never converges, so cooling down is an appropriate thing to do whenever non-negligible rotations are detected.

A rotation is assumed when repeatedly β had the same sign and was within a rotation sensitive range as depicted in Fig. 4, in which $v.\mathbf{p}$ is the last impulse of v, while $\mathbf{p_1}, \mathbf{p_2}$ and $\mathbf{p_3}$ are possible directions for v's current movement. $\mathbf{p_1}, \mathbf{p_2}$ and $\mathbf{p_3}$ are examples of a movement in the right direction, a rotation and an oscillation, respectively.

This situation triggers the skew gauge $v.d$. The more $|v.d|$ approaches 1, the more unbalanced it is. $v.t$ is scaled down by v's unbalancedness as this is a measure of how likely v is part of a rotation. The choice of a scale factor σ_{r} allows for necessary rotations.

A vertex is subject to an *oscillation* if its last and current impulse point in opposite directions, which is detected by testing whether $\cos(\beta) < 0$. If this is the case, GEM assumes that the vertex has just passed its right position and lowers the temperature according to a sensitivity factor σ_{o} assuming that it had passed its optimal position and will turn around again in the next round. Subsequent oscillations will therefore finally freeze the vertex.

To the opposite, if v's current impulse has approximately the same direction as the

```
-- Input:                                                                    1
--   v          vertex to be updated                                         2
--   p          current impulse of v                                         3
-- Output:                                                                   4
--   v          with updated ξ, t, d, p                                      5
-- Constants:                                                               6
--   T_max      maximal temperature [256]                                    7
--   α_o        opening angle for oscillation detection; α_o ∈ [0, π/2]  [π] 8
--   α_r        opening angle for rotation detection; α_r ∈ [0, π]  [π/3]    9
--   σ_o        sensitivity towards oscillation; σ_o ≥ 1  [1/3]              10
--   σ_r        sensitivity towards rotation; σ_r ∈ (0, 1]  [1/2|V|]         11
                                                                            12
if p ≠ 0 then                                                               13
    p:=v.t · p/|p|; -- scale with current temperature                       14
    v.ξ:=v.ξ + p;                                                           15
    c:=c + p; -- save the division at this point                            16
if v.p ≠ 0 then                                                             17
    β:=∠p, v.p;                                                             18
    if sin β ≥ sin(π/2 + α_r/2) then                                        19
        -- rotation                                                         20
        v.d:=v.d + σ_r · sgn(sin β);                                        21
    if |cos β| ≥ cos(α_o/2) then                                            22
        -- oscillation                                                     23
        v.t:=v.t · σ_o · cos β;                                             24
    v.t:=v.t · (1 − |v.d|);                                                 25
    v.t:=min(v.t, T_max);                                                   26
    v.p:=p;                                                                 27
```

Fig. 3. Temperature update algorithm

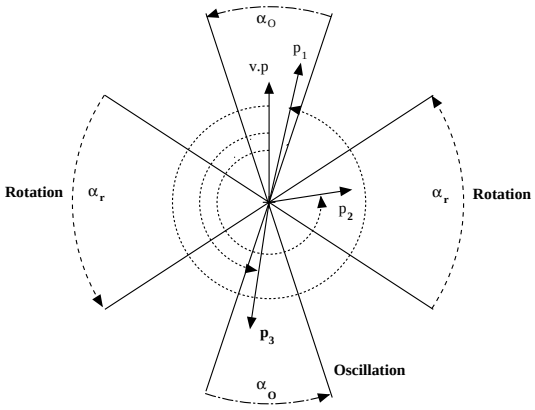

Fig. 4. Detection of rotations and oscillations

last one ($\cos\beta \approx 1$), GEM interprets this as a move in the "right" direction and raises the temperature somewhat to accelerate v's next movement. An opening angle α_{o} controls the sensitivity towards this situation.

Of course, the opening angles α_{o} and α_{r} have to be chosen carefully. Otherwise, they might not be sensitive enough or cause overreactions.

After computing the new local temperature and perhaps adjusting the skew gauge, some housekeeping is done in order to update the barycenter. Having done so, the iteration is finished.

5 Measurements

In this section, we compare the runtimes of the GEM algorithm with those of publicly available implementations of the Kamada/Kawai and Fruchterman/Reingold algorithms contained in the GraphEd tool [7] subsequently called KK and FR, respectively. Our test suite consists of 30 graphs of different types, sizes and densities. All measurements have been conducted on a SparcStation 10 using the GraphEd test suite procedure. Due to the unavoidable overhead incurred by the UNIX operating system, these timings should be interpreted as a relative comparison and not as a performance benchmark.

An important determining factor are the parameters to the algorithms. In each case, we used the default values. As mentioned in the source code for FR, the implementation is heavily optimized. GEM was run in single-insertion mode for the initial placement. Hard-coded animation output in the KK code was commented out to achieve competitive test conditions.

A priori, we expected GEM to outperform KK and FR in terms of runtime as we rely on integer arithmetic only, but it was unclear how randomization and the inherent imprecision would influence the quality of the drawings. The resulting runtime and quality measurements are given in Fig. 1. The quality data measured for each drawing includes the number of edge crossings χ, the mean edge length $\bar{E}$ (used for scaling of the results), the edge length deviation $\tilde{E}$, the minimal vertex distance $D_{\min}$ and the maximal vertex distance $D_{\max}$. The values for $\tilde{E}, D_{\min}$ and $D_{\max}$ are scaled by $\bar{E}$ as to ensure comparability of the results.

6 Results

As all compared algorithms strive to satisfy the same aesthetic criteria, it is not surprising to find that the quality measurements yield similar results. On the average, we observed that GEM performs slightly better than FR. On small sparse graphs KK gives slightly better results but becomes considerably worse on large graphs.

The quality of GEM drawings is good. Considering that the heuristics used in the quality function have no notion of a crossing-free drawing, the algorithm surprisingly often results in planar embeddings. In comparison to FR and KK, GEM can resolve cycles and foldings in parts of the graph easily (see Fig. 11) as indicated by the low number of edge crossings.

In our experience, GEM can cool every graph down to $5°$. In most cases, the drawings are not significantly worse than those for the default $T_{\min} = 3°$.

	graph				GEM					KK					FR				
#	Name	$\|V\|$	$\|E\|$	Density	time[s]	χ	$\tilde{E}$	$D_{\min}$	$D_{\max}$	time[s]	χ	$\tilde{E}$	$D_{\min}$	$D_{\max}$	time[s]	χ	$\tilde{E}$	$D_{\min}$	$D_{\max}$
1	Binary Tree	15	14	sparse	0.13	0	0.145	0.851	6.117	0.75	0	0.030	0.952	5.550	0.54	0	0.216	0.771	6.353
2	Path	16	15	sparse	0.19	0	0.082	0.821	10.961	0.97	0	0.048	0.903	13.057	0.95	0	0.130	0.735	12.148
3	Cycle	16	16	normal	0.19	0	0.018	0.955	5.361	0.99	0	0.037	0.949	5.483	0.70	0	0.003	0.995	5.171
4	Square Grid	16	24	normal	0.17	0	0.048	0.943	4.300	1.15	0	0.019	0.962	4.231	0.59	0	0.063	0.935	4.184
5	Wheel	13	24	normal	0.12	0	0.319	0.643	2.681	0.62	4	0.253	0.522	2.647	0.10	0	0.318	0.675	2.643
6	Hypercube 4D	16	32	normal	0.15	24	0.413	0.337	2.004	1.08	22	0.181	0.323	2.738	0.06	24	0.064	0.478	2.685
7	$K_{8,8}$	16	64	dense	0.23	596	0.257	0.298	1.911	1.57	686	0.236	0.214	1.670	1.75	624	0.691	0.782	4.235
8	K_{12}	12	66	dense	0.15	406	0.371	0.431	1.619	0.73	402	0.371	0.427	1.577	1.05	407	0.371	0.443	1.608
9	Star	24	23	sparse	0.28	0	0.132	0.326	2.213	2.88	0	0.187	0.139	2.346	1.83	0	0.185	0.412	2.263
10	Binary Tree	31	30	sparse	0.52	0	0.205	0.611	8.677	6.44	1	0.131	0.510	7.947	2.92	0	0.315	0.535	9.554
11	Dodecahedron	20	30	normal	0.22	6	0.149	0.403	3.603	1.84	10	0.130	0.452	3.539	0.69	10	0.137	0.567	3.511
12	Hypercube 5D	32	80	normal	0.42	177	0.049	0.221	3.189	7.42	168	0.119	0.237	3.215	1.43	177	0.062	0.000	3.150
13	Triangular Grid	28	63	normal	0.37	0	0.099	0.809	6.109	5.14	0	0.040	0.902	6.079	1.31	0	0.144	0.659	6.419
14	K_{24}	24	276	dense	0.59	8129	0.417	0.305	1.761	6.15	8347	0.416	0.266	1.75	4.47	7962	0.418	0.344	1.765
15	Path	48	47	sparse	1.63	0	0.054	0.803	17.83	24.48	3	0.103	0.135	13.276	6.51	2	0.180	0.423	14.964
16	Binary Tree	63	62	sparse	1.90	0	0.261	0.437	12.00	49.23	1	0.115	0.286	9.960	10.58	1	0.418	0.302	14.142
17	Fibonacci Tree	54	53	sparse	1.77	0	0.267	0.543	13.84	31.90	3	0.131	0.283	10.657	8.12	1	0.407	0.352	15.365
18	Cycle	48	48	normal	1.73	0	0.034	0.911	15.28	22.54	1	0.116	0.364	14.408	6.49	0	0.107	0.666	16.667
19	Square Grid	49	84	normal	1.11	0	0.095	0.832	8.257	26.88	0	0.056	0.886	8.705	6.21	0	0.126	0.808	8.326
20	Torus	64	128	normal	1.98	46	0.311	0.473	7.894	63.67	54	0.326	0.220	7.880	11.7	44	0.454	0.481	8.341
21	Triangular Grid	55	135	normal	1.55	0	0.115	0.718	9.227	35.32	3	0.094	0.552	9.645	9.18	0	0.173	0.547	9.654
22	Hypercube 6D	64	192	dense	1.20	1004	0.062	0.211	3.871	55.86	977	0.201	0.119	3.777	12.27	1000	0.069	0.216	3.806
23	Binary Tree	127	126	sparse	9.19	0	0.311	0.298	16.125	1.26	2044	0.497	0.000	2.324	41.28	2	0.521	0.228	20.311
24	Hexagonal Grid	96	132	normal	4.65	0	0.149	0.764	11.784	192.75	3	0.274	0.288	12.585	24.26	0	0.189	0.715	11.830
25	Triangular Grid	120	315	normal	5.99	0	0.124	0.618	14.314	388.00	0	0.156	0.588	16.028	42.17	0	0.199	0.426	15.072
26	Path	128	127	sparse	9.54	0	0.053	0.704	30.189	432.73	21	0.212	0.034	20.030	44.14	5	0.265	0.148	26.017
27	Binary Tree	255	254	sparse	45.04	0	0.367	0.257	21.62	> 1000					186.21	37	0.635	0.000	20.751
28	Path	256	255	sparse	37.93	2	0.086	0.572	41.61	> 1000					181.65	32	0.519	0.000	27.785
29	Triangular Grid	210	570	normal	30.88	0	0.127	0.563	19.57	2110.06	435	0.266	0.045	19.162	131.63	4	0.219	0.196	20.465
30	Square Grid	256	480	normal	71.78	0	0.118	0.701	20.79	> 1000					197.41	89	0.250	0.000	18.421

Table 1. Test suite: Graphs 1–8 are tiny, 9–14 small, 15–22 medium, 23–26 large, 27–30 huge.

The overall result is that GEM is consistently faster than the other methods. FR is about four times slower, and KK deteriorates rapidly as the graph sizes increase. It remains unclear if this is a feature of the implementation or inherent to the algorithm.

One iteration of the GEM algorithm takes time O($|V|$) since the forces exerted on v by the remaining $|V| - 1$ vertices have to be recomputed.

The PC version of GEM runs as fast as the GraphEd version. This was achieved by optimizations at the data structure level and may also be attributed to the fact that the environment has no GUI and multi-user overhead. On an amd486DX40 PC we measured a speed of approximately $120,000$ iterations per second. A heuristic we found by experimenting says that we need approximately $|V|$ rounds. Since a round consists of $|V|$ iterations and each iteration considers $|V|$ vertices, this implies an estimated time complexity of O($|V|^3$), where the constant is small and depends on G.

7 Conclusions

In this paper, we have presented an adaptive algorithm to compute a layout for undirected graphs based on local temperatures. We introduced several algorithmic improvements to previous approaches, including local temperatures, attraction towards the barycenter and the detection of rotations and oscillations. This represents a departure from the conventional gradient descent methods narrowing into the nearest local minimum of the quality function.

We were able to match or improve the quality of the results obtained by widely used implementations of the Fruchterman/Reingold and Kamada/Kawai algorithms while running consistently faster than these. The results also confirmed the hypothesis that randomization can be successfully used for the force-directed layout of undirected graphs.

While other published work presents mainly small-sized graphs and occasionally medium-sized graphs, GEM can easily handle large and complex graphs. As graphs get larger and larger, the drawing area becomes too small very quickly. Also, the human eye will have difficulty comprehending larger graphs as a whole, so graph partitioning schemes should be taken into consideration.

Although we chose not to explicitly minimize edge crossings, GEM can often avoid crossings (see Fig. 18).

8 Open Questions

Several problems remain to be solved. First of all, nothing is known about the theoretical behavior of the proposed algorithm. Although we were able to experimentally confirm a time complexity of O($|V|^3$) with small constants, we cannot give a formal proof. Even worse, although the GEM algorithm almost always terminates with satisfactory results, no proof of convergence is apparent.

Further research is necessary to determine the importance of each single factor in the GEM algorithm and their interplay. Experiments in this direction could investigate in leaving out single or several factors contributing to the success of GEM (randomization, oscillation, rotation, gravitation).

There exist two interesting connections between the GEM algorithm and artificial neural network theory. Having seen animations of GEM runs on mesh-like structures, similarities to Kohonen feature maps are apparent. The question comes to mind whether the connections between these domains are deeper. A second, even more intriguing connection can be made to improved backpropagation learning algorithms (Rprop, QuickProp) which have a notion of local learning rates, which is very similar to the GEM idea of having local temperatures. An interesting experiment would therefore be to devise a neural network for graph drawing.

Unfortunately, these connections do not help at present to settle the complexity and stability questions raised above as researchers in these areas are themselves actively investigating these questions.

A major problem in the area of graph drawing is the non-existence of a standard set of graphs by which to judge on the quality of drawings. A first step in this direction would be the definition of such a set for the restricted domain of straight-line drawings. We hope to have contributed by the distinction between graphs of several sizes and densities for our test suite.

9 Acknowledgements

W. Zimmermann gave valuable comments on the presentation of our results. We would like to thank the anonymous referees who pointed out earlier work in force-directed placement.

References

1. G. Di Battista, P. Eades, H. de Fraysseix, P. Rosenstiehl, and R. Tamassia, editors. *Proceedings of the ALCOM International Workshop on Graph Drawing 1993*. ALCOM, 1993.
2. G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. Report, Brown University, June 1994.
3. R. Davidson and David Harel. Drawing graphs nicely using simulated annealing. Technical Report CS89-13, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1989. revised July 1993, to appear in Communications of the ACM.
4. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
5. C. J. Fisk, D. L. Caskey, and L. E. West. ACCEL: Automated circuit card etching layout. *Proceedings of the IEEE*, 55(11):1971–1982, November 1967.
6. T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Software–Practice and Experience*, 21, 1991.
7. M. Himsolt. Graphed: A graphical platform for the implementation of graph algorithms. In *Proceedings of Graph Drawing '94*, LNCS, Princeton, New Jersey, October 10-12 1994. DIMACS Workshop on Graph Drawing, Springer. this volume.
8. N. R. Quinn Jr. and M. A. Breuer. A forced directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and Systems*, CAS-26(6):377–388, 1979.
9. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31, 1989.

10. Scott Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
11. Donald E. Knuth. *Seminumerical Algorithms*, volume 2. Addison-Wesley, 2nd edition, 1981.
12. J.B. Manning. *Geometric symmetry in graphs*. PhD thesis, Purdue University, December 1990.
13. N. Metropolis, W. Rosenbluth, M.N. Rosenbluth, and A.H. Teller. Equation of state calculations by fast computer machines. *J. Chem. Phys.*, 21:1087, 1953.
14. D. Tunkelang. A layout algorithm for undirected graphs. In *Graph Drawing '93, ALCOM International Workshop PARIS 1993 on Graph Drawing and Topological Graph Algorithms*, September 1993.
15. H. Watanabe. Heuristic graph displayer for G-Base. *International Journal of Man-Machine Studies*, 30:287–302, 1989.

A GemDraw

This appendix describes GEMDRAW, the environment in which we developed GEM. GEMDRAW is written for a PC in Borland C using the Borland Graphics Interface (BGI). It turned out early in the design stage that we needed a specialized tool for visualizing vertex temperatures and for quickly producing large test graphs. We decided to select and implement mechanisms for extending small graphs in a regular way, which are to be described. In addition, GEMDRAW knows several types of parameterized graphs (mesh-structures, trees, hypercubes, completeley connected) and allows for interactive creation and modification of graphs.

Using the concepts of *iteration* and *duplication*, complex structures can be created easily. For each graph $G = (V, E)$, an iterated graph can be defined by cloning the original graph $|V|$ times and connecting each vertex of the original graph to all vertices of its assigned clone. An iterated K_{10} is shown in Fig. 5. The duplication of G is defined by cloning G once and adding edges between each original vertex and its clone. Figure 6 shows a duplicated K_{15}. As an example of the usefulness of the concept, Fig. 7 shows a torus defined by duplicating a cycle several times. A hypercube of dimension n can quickly be produced as follows. Create a vertex and repeat n times: duplicate the graph produced so far.

B Visual examples

In this appendix we present several examples for the quality of the graphs drawn by GEM, including well-known examples from the literature.

We first turn our attention to sparse structures and give a sequence of drawings for cycles in Fig. 8. This is a good example of the adaptive nature of the GEM algorithm as a folding causes local temperatures to remain high. In Fig. 9 the heuristics of evenly spacing vertices is illustrated using paths as an example.

A sequence of intermediate steps in the drawing of a triangular mesh (see Fig. 11) shows that GEM can handle foldings well, as opposed to [6]. Observe that the neighborhood of a folded area remains hot.

Drawings of square (Fig. 12) and hexagonal (Fig. 13) meshes of different sizes are further examples demonstrating that GEM is not focused towards certain structures as

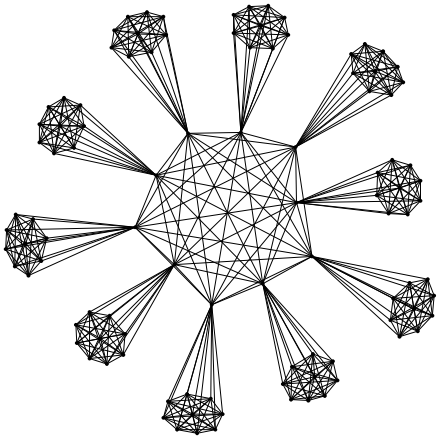

Fig. 5. Iterated K_{10}

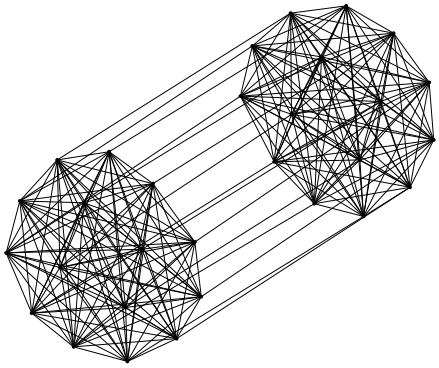

Fig. 6. Duplicated K_{15}; the even vertex distance heuristic forces vertices to be placed inside the hull

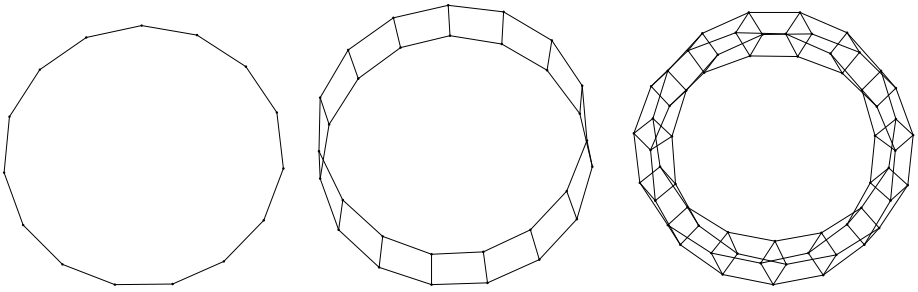

Fig. 7. Torus defined by duplicating a cycle

all runtimes are a function of $|V|$. It can be observed that some huge graphs become distorted. This is not a consequence of the gravitational force used to compute vertex impulses, but merely of the random vectors added to the impulse: Once perfect symmetry is disturbed, perturbations of this kind will occur.

Since GEM does not optimize for the number of edge crossings, the resulting drawing will often be a projection of a 3D-picture. This can be exemplified with Fig. 14– 16. In Fig. 16, the left layout is produced much more often than the middle one, which in turn is computed more often than the right one.

We finish with GEM output for several examples from the papers of Fruchterman/Reingold and Davidson/Harel and compare the results. While GEM manages to draw the graph in Fig. 18 planar as do Davidson/Harel, the drawing of the graph in Fig. 19 is basically the Fruchterman/Reingold version.

This article was processed using the LaTeX macro package with LLNCS style

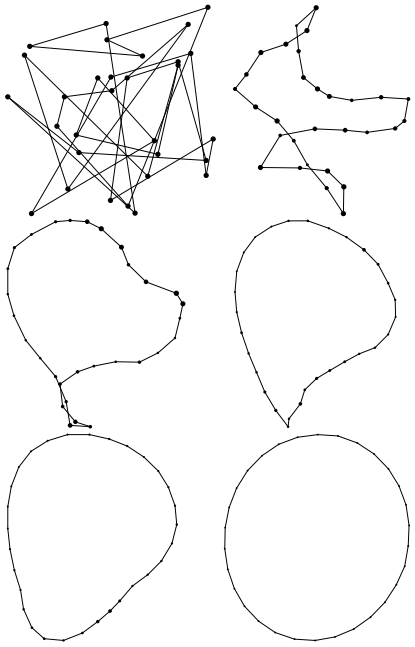

Fig. 8. Cycle with $|V| = 30$ vertices in different stages of development

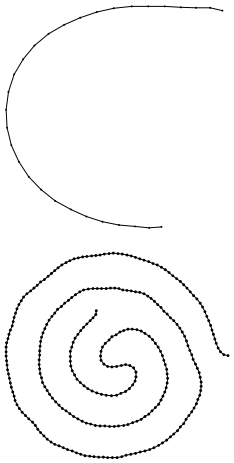

Fig. 9. Paths of size $|V| = 30, 300$ after $3000, 20700$ iterations

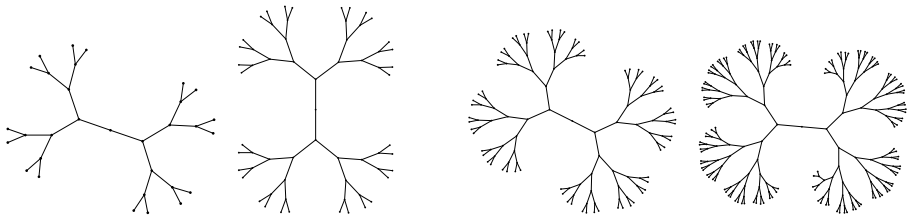

Fig. 10. Binary trees of size $|V| = 31, 63, 127, 255$ after $1178, 3276, 9906, 34935$ iterations

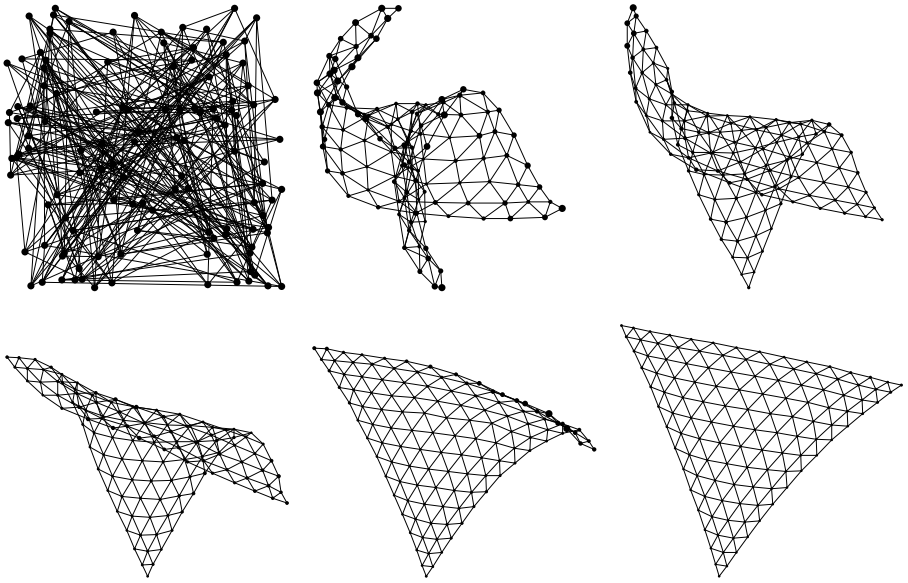

Fig. 11. Intermediate states of a triangular mesh with foldings

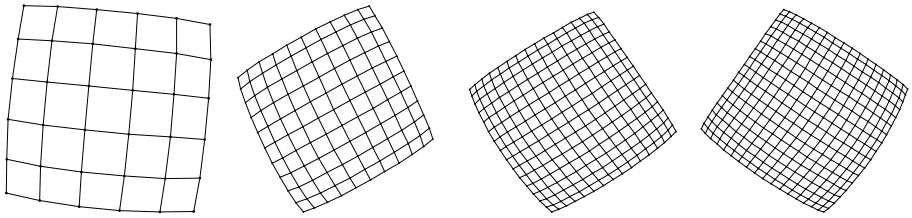

Fig. 12. Square grids of size $|V| = 36, 121, 256, 324$ after $972, 6534, 26880, 42472$ iterations

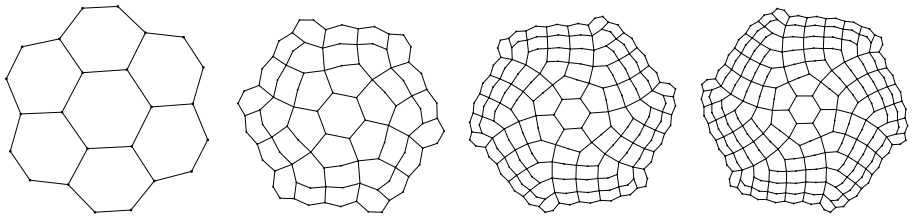

Fig. 13. Hexagonal grids of size $|V| = 24, 96, 216, 294$ after $720, 5184, 15120, 29106$ iterations

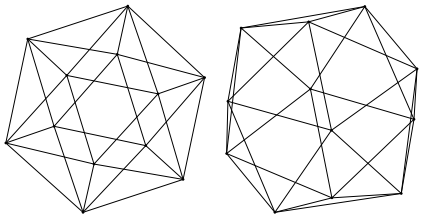

Fig. 14. Two different layouts of an icosahedron ($|V| = 12, |E| = 30$) after ≈ 220 iterations (see also figure 29 from Fruchterman and Reingold)

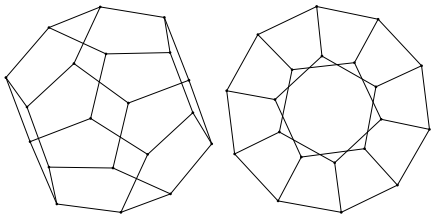

Fig. 15. Two different layouts of an dodecahedron ($|V| = 20, |E| = 30$) after ≈ 700 iterations (see also figure 57 from Fruchterman and Reingold)

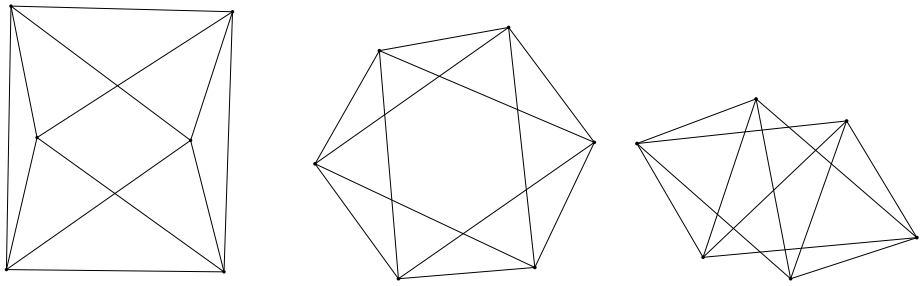

Fig. 16. Three different layouts of an octahedron ($|V| = 6, |E| = 12$) after ≤ 200 iterations

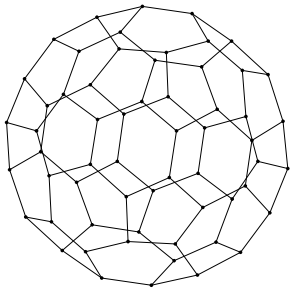

Fig. 17. Drawing of a soccer ball (also known in chemistry as the C_{60} molecule or *Buckminster Fulleren*) with $|V| = 60, |E| = 90$ after ≤ 5000 iterations at $3°$

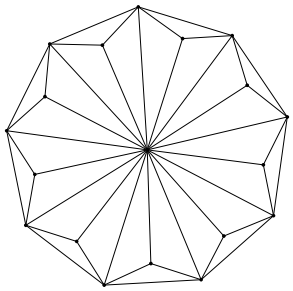

Fig. 18. GEM drawing of the graph in [6, fig. 24]; this drawing is similar to [3, fig. 1]

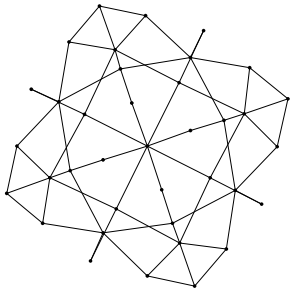

Fig. 19. GEM drawing of the graph in [3, fig. 12]; this drawing is similar to [6, fig. 26]