A Parallel Simulated Annealing Algorithm for Generating 3D Layouts of Undirected Graphs*

Burkhard Monien,[1] Friedhelm Ramme,[2] Helmut Salmen [1]
[bm | ram | hlmut]@uni-paderborn.de

[1] Department of Computer Science, University of Paderborn, Germany
[2] Paderborn Center for Parallel Computing (PC^2),
University of Paderborn, Germany **

Abstract. In this paper, we introduce a parallel simulated annealing algorithm for generating aesthetically pleasing straight-line drawings. The proposed algorithm calculates high quality 3D layouts of arbitrary undirected graphs. Due to the 3D layouts, structure information is presented to the human viewer at a glance. The computing time of the algorithm is reduced by a new parallel method for exploiting promising intermediate configurations. As the algorithm avoids running into a local minimum of the cost function, it is applicable for the animation of graphs of reasonably larger size than it was possible before.
Subsequent to the discussion of the algorithm, empirical data for the performance of the algorithm and the quality of the generated layouts are presented.

Keywords: 3D graph layout, straight-line drawing, parallel simulated annealing

1 Motivation

During the last few years the problem of generating aesthetically pleasing layouts of a given graph $G = (V, E)$ has received an increasing amount of attention [BETT94]. After transforming an abstract graph description into an appealing drawing, a human viewer can, at a glance, derive additional information about the properties of G which are inherent in the abstract description. Due to the high complexity of the problem, which is $\mathcal{NP}$-hard and the small computational power of the desktop workstations of the past, current layout programs are mostly computing two dimensional straight-line drawings. Only a few authors [FR91, CELR94, FPS94] have considered the challenge of calculating 3D layouts. These kinds of drawings, however, are best suited to present inside information to human viewers.
Modern workstations are equipped with fast graphic interfaces which allow the user to rotate a 3D representation without visible disturbance and to create

* This work was partially supported by the ESPRIT Basic Research Action No. 7141 (ALCOM II)

** WWW: `http://www.uni-paderborn.de/fachbereich/AG/monien/index.html`
WWW: `http://www.uni-paderborn.de/pcpc/pcpc.html`

virtual walk-through animations. However, the calculation of 3D layouts requires a reasonable amount of computational power. For reducing the computing time we will take advantage of a parallel computer system. Criteria like:

- reflect inherent symmetries
- avoid edge crossings
- maximize angle resolution
- keep edge lengths uniform
- distribute vertices evenly

should be optimized in order to get aesthetically pleasing drawings [KK89, FR91, Tu93]. Unfortunately, some of them are competitive and there is no reasonable way to assign absolute weights to them. Most of the known algorithms for solving this problem can be classified into one of three groups:

The *combined heuristics* make use of heavy-duty preprocessing techniques to determine graph characteristics like strongly connected components, planar subgraphs or minimal height breadth-first spanning trees [HS93, Tu93]. The final layout is gradually obtained afterwards.

The *spring embedder model* for drawing undirected graphs was introduced by Eades [Ea84]. Using an analogy to physics, vertices are treated as mutually repulsive charges and edges as springs connecting and attracting the charges. Starting from an initial placement, the spring system is moved to a state with minimal energy. Kamada and Kawai [KK89] refined the model of Eades. They introduced the desirable length, a combined value of the shortest path between two vertices and the desired length of a single edge in the display plane. There, a partial differential equation system must be solved for each vertex in each iteration to determine its new location. Fruchterman and Reingold [FR91], like Eades, made only vertices that are neighbors attract each other, while all other vertices repelled each other. Their layouts were determined during a fixed number of iterations. The proposed algorithm calculates the effect of the attractive and repulsive forces on each vertex and limits the total displacement by a temperature value, afterwards. The temperature and its cooling schedule were borrowed from the simulated annealing method. While all these spring embedders are quite fast, they perform a gradual descent and converge at a local minimum of the energy function. The algorithm presented in [FLM94] avoids this drawback by introducing a local temperature value to each vertex. It terminates after a fixed number of rounds or if all local temperatures are below a certain threshold. Using a new heuristic to detect rotations and oscillations during the computations it was possible to outperform the speed of the algorithms presented in [KK89, FR91]. However, no statements could be made about the convergence behavior.

The third group of algorithms uses randomness to overcome the problem of ending up in local minima. Its model, called *simulated annealing* (SA), is a flexible optimization method, suited for large-scale combinatorial optimization problems [DLS93, JAMS89-91]. To our knowledge, Davidson and Harel [DH93] were the first who used an SA approach for calculating aesthetically pleasing graph layouts. The graph drawing problem, especially, is characterized by large configurations as well as a non-trivial neighborhood relation and a complex cost-function. These properties seem to be highly responsible for the long computing times required by the SA algorithm proposed. However, the resulting quality was comparable to or even better than that of the spring embedders [FR91]. Due to the considerable time requirement, even for graphs of small and medium sizes and the large number of parameters, the SA algorithm of [DH93] is not suited to draw larger graphs. A first parallel approach (implemented at a CM-2) for tackling the drawing problem was presented in [KMS94]. However, it was reported that the convergence rate was slow and that the network diagrams generated had only small sizes.

Within this paper, we will present a parallel SA algorithm for generating 3D layouts of arbitrary undirected graphs. The required time is significantly reduced compared to [DH93], by using the computing facilities of a modern MPP system. For our algorithm, the assignment of four parameters is sufficient. All others are either fixed or were made self-adjusting during runtime. This property is essential for being applicable in practise. In Section 2 of this paper, we will present the PARallel Simulated Annealing algorithm (PARSA) in details. Its properties and some layout results will be discussed in Section 3. Additionally, empirical performance data will be given.

2 The PARSA-Algorithm

SA has been applied successfully to many problems characterized by a large discrete configuration space, too large for an exhaustive search, over which a cost-function is to be minimized (or maximized). After picking some initial configuration, most of the iterative methods continue by choosing a new configuration at each step, evaluating it, and possibly replacing the previous one with it. This action is repeated until some termination condition is satisfied. In general, the procedure ends up in a local minimum, rather than the desired global one. SA tries to prevent this by allowing uphill moves with a certain probability. A sequential SA algorithm[3] consist of two nested loops. Within the interior loop, new configurations are continuously evaluated until an equilibrium of the corresponding cost-function $\mathcal{M}$ is reached. A new configuration C_k is derived from the previous one by choosing an arbitrary vertex of C_{k-1} and moving it to a randomly chosen position. The maximum displacement is limited by a value which is proportional to the temperature ϑ. If an improvement is reached, the new configuration is accepted. In case of deterioration, acceptance is probabilistic. After an equilibrium is detected, the virtual temperature is reduced according to a certain cooling-schedule. If a termination condition is fulfilled the final graph layout is returned.

At the beginning of the SA algorithm, the configurations resembles a muddle. This situation has to be resolved step by step afterwards. To attain a satisfactory convergence behavior, it is important that a single vertex (or only a small number of vertices) are moved at a time, and that each displacement is limited by a temperature dependent factor. When using SA to generate 2D layouts the second statement prevents single vertices to be placed at distant but reasonably better positions if the temperature is already lowered and a potential (or

[3] We assume that the reader is familiar with the basic annealing techniques. Otherwise, a good introduction can be found in [HRS86, JAMS89-91, OG89, DLS93] and the references given.

cost-) barrier must be overcome. Therefore, quite often drawings of unpleasing quality were returned. This effect can be reduced to a certain degree by using a sophisticated cost-function [DH93]. The PARSA-algorithm, however, abolishes this by generating 3D configurations directly. The additional dimension permits single vertices to circumnavigate obstacles and to travel to a distant destination at low temperature values and in situations where the intermediate configuration is relatively stable. The resulting advantage is twofold. Firstly, a simple cost-function speeds up the sequential algorithms. Secondly, the solution-space allows for smooth transitions which results in a convinced convergence behavior with high quality layouts, even if the graphs are going to become large.

To simplify reading, we kept the notation of the PARSA-algorithm (Fig. 1) close to the sequential case. Thus, for the discussion of the common parts one should ignore the lines 10-11, 13-15, 17, 21-23, 25, set $I = 0$, and assume that the remaining algorithm is executed at a single processor.

2.1 Cost-function

The cost-function is one of the most critical parts of any SA algorithm. The quality of a desired solution must be well complemented by the values returned for intermediate solutions rsp. configurations. As this function forms the compute-intensive kernel of the algorithm, it should be possible to calculate its new values $\mathcal{M}(C_k)$ very quickly and if possible incrementally from its previous values $\mathcal{M}(C_{k-1})$. The cost-function we have chosen consists of only three parts: $\mathcal{M} = \mathcal{M}_a + \mathcal{M}_e + \mathcal{M}_p$ with $\mathcal{M}_a$ the *angle-costs,* $\mathcal{M}_e$ the *edge-costs,* and $\mathcal{M}_p$ the *pseudo-edge-costs.* The new values of $\mathcal{M}_i(C_k)$ ($i \in \{a, e, p\}$) are determined by adding a fast computable term $\Delta_i(v)$ to $\mathcal{M}_i(C_{k-1})$, with v being the vertex moved within the transition $C_{k-1} \vdash C_k$.

Angle-costs: Each angle between neighboring edges of a vertex ($\sphericalangle$) in the 3D configuration space is associated with a cost value. The smaller an angle, the higher its cost. An angle α larger than π has costs equivalent to $2\pi - \alpha$. Let λ_a be a scaling factor and k_a a constant. Then, the angle-costs are determined by:

$$\mathcal{M}_a = \lambda_a \cdot \sum_{v \in V} \sum_{a \in A_v} \frac{c_a(a)}{deg(v)} \quad \text{with} \quad c_a(\alpha) = \frac{\pi - \alpha}{k_a \cdot \alpha + \pi}$$

$$\text{and} \quad A_v = \{\sphericalangle(k,l) \mid k = \{v, v_i\},\, l = \{v, v_j\},\, k, l \in E,\, i < j\}$$

Edge-costs: This term evaluates the difference between the desired length L_e of an edge and its Euclidean length Δ in the 3D space. Best results were achieved by using a twofolded function: A hyperbola if the length is below L_e, and a straight line otherwise. Let λ_e be a scaling factor and k_{e1}, k_{e2} some constants. Then, the edge-costs are determined by:

$$\mathcal{M}_e = \lambda_e \cdot \sum_{\substack{e=\{u,v\} \\ e \in E}} \frac{c_e(\Delta)}{deg(u) + deg(v)} \quad \text{with} \quad c_e(\Delta) = \begin{cases} L_e \cdot \frac{L_e - \Delta}{k_{e1} \cdot \Delta + L_e} & \text{if} \quad \Delta < L_e \\ \frac{\Delta - L_e}{k_{e2}} & \text{otherwise} \end{cases}$$

```
-- $k$        : number of accepted configurations
-- $C_k$      : configuration under negotiation
-- $\mathcal{M}$      : value of the cost-function
-- $\overline{E}$      : mean value of $\mathcal{M}$
-- $\overline{V}$      : mean deviation of $\mathcal{M}$
-- $\vartheta$      : temperature value
-- $N[0..h]$ : with $N[i] = (\mathcal{M}, \vartheta, \overline{V})$ the current
--           : cost-function, temperature, mean deviation   at PE $i$
--           : with $i = 0$ being this PE and $1 \leq i \leq h$ the neighbors
--           : initially $\mathcal{M} = \infty$,   $\vartheta_0 = 3d$,   $\overline{V} = \infty$

$PARSA$ $(G = (V, E))$

BEGIN
   $C_0$ := random_initial_configuration () ;
   call_init ($N[0], \overline{E}$);
   $k := 1$;
      REPEAT
         REPEAT
            $C_k$ := create_new_conf ($C_{k-1}, N[0].\vartheta$);
            $m := N[0].\mathcal{M}$;
            $N[0].\mathcal{M} := \mathcal{M}(C_k)$;
            $I := i \in \{0..h\} : N[i].\mathcal{M}$ is minimal ;
            IF  short_subchain ($C_k$)  THEN  $I := 0$;  FI
            IF  ($N[I].\mathcal{M} < m$)  THEN
               IF  ($I > 0$)  THEN  $C_k$ := get_conf_from_neighbor ($I$);
                                    $N[0].\mathcal{M} := N[I].\mathcal{M}$;
               FI
               calc_new ($\overline{E}, N[0].\overline{V}$);
               send_to_all_neighbors ($N[0]$);
               $k := k + 1$;
                              ELSE
               IF  ( $e^{-\frac{m - N[I].\mathcal{M}}{N[0].\vartheta}} \geq$ rnd (0..1) )  THEN
                  IF  ($I > 0$)  THEN  $C_k$ := get_conf_from_neighbor ($I$);
                                       $N[0].\mathcal{M} := N[I].\mathcal{M}$;
                  FI
                  calc_new ($\overline{E}, N[0].\overline{V}$);
                  send_to_all_neighbors ($N[0]$);
                  $k := k + 1$;
               FI
            FI
         UNTIL  equilibrium ();
         $N[0].\vartheta$ := temp_reduction ();
      UNTIL  terminate () ;
      RETURN ($C_{k-1}$);
END ;
```

Fig. 1. The PARSA algorithm

Pseudo-edge-costs: Up to now, only those vertices were considered which are directly connected by an edge. To get aesthetically pleasing layouts, we have to consider all pairs of vertices. Instead of taking the original graph G we take the square-graph $G^2 = (V, E^2)$ with $E^2 = E \cup \{\{i,j\} \mid i,j \in V, \exists k \in V\colon \{i,k\},\{k,j\} \in E\}$. Now, if all edges of its complementary graph $\overline{G^2} = (V, \overline{E^2})$ with $\overline{E^2} = \{\{i,j\} \mid i,j \in V, \{i,j\} \notin E^2\}$ are imposed by additional costs, non-adjacent vertices are placed distantly. When using $\overline{G^2}$ instead of G a double evaluation with the angle-costs at a vertex can be avoided. Especially when drawing large graphs, the muddle resulting from the initial random layout is rectified very fast by using the pseudo-edge-costs. Let λ_p be a scaling factor, k_{p1}, k_{p2} some constants and d the expected diameter of the final graph layout (see 2.2). Then, the pseudo-edge-costs are determined by:

$$\mathcal{M}_p = \lambda_p \cdot \sum_{p \in \overline{E^2}} c_p(\Delta) \quad \text{with} \quad c_p(\Delta) = \begin{cases} L_e \cdot \frac{k_{p1} \cdot d - \Delta}{k_{p2} \cdot L_e + k_{p1} \cdot d} & \text{if} \quad \Delta < 3\,d \\ 0 & \text{otherwise} \end{cases}$$

2.2 Cooling-schedule

The number of intermediate configurations and the decrease of the temperature are controlled according to a cooling-schedule together with the function *equilibrium* () (line 29 in Fig. 1). The choice of the cooling-schedule has great influence on the convergence-behavior of the algorithm and its efficiency. To be applicable in practice, it is essential that the schedule is *self-adapting* to the parameters of a given problem instance [HRS86, DLS93].

Initial temperature: The initial temperature should be chosen depending on the problem instance. It should be large enough for potentially spying out each corner of the solution space. Taking three times the estimated diameter d of the 3D configuration space was found to be sufficient. In order to estimate d, a maximum plane graph with $|V|$ vertices and f faces each having three corners is considered. Using Eulers formula for plane graphs we get $f = 2(|V| - 2)$. With L_e being the desired edge length, the area of a face can be approximated by $\frac{1}{2} L_e^2$. If all faces are placed equally on the surface of a sphere, we get $f \frac{L_e^2}{2} = \pi d^2$ and thus $d = L_e \cdot \sqrt{\frac{|V|-2}{\pi}}$. Now, we can set $\vartheta_0 = 3\,d$ to get the initial value of the virtual temperature.

Temperature reduction: The schedule proposed in [HRS86] reduces the temperature according to the length of the Markov chains, whereby the length of a chain is redetermined at each temperature level. To keep the variations between two consecutive levels lower than the standard deviation σ of the cost-function, the new value ϑ_{k+1} is set to $\vartheta_k \cdot e^{-\frac{\lambda \cdot \vartheta_k}{\sigma}}$ with $0 < \lambda \leq 1$. As the reduction ratio depends on σ, large changes in the standard deviation resulting from short Markov chains will have significant impact on the current schedule. This influence can be alleviated by using the smoothing technique of [OG89] to approximate σ by $\sigma_{k+1}^s = (1 - \omega) \cdot \sigma_{k+1} + \omega \cdot \sigma_k^s \cdot \frac{\vartheta_{k+1}}{\vartheta_k}$ with $0 \leq \omega \leq 1$.

Equilibrium detection: The equilibrium detection is performed depending on the number of accepted configurations and on the standard deviation of the cost-function. It can be assumed [DLS93] that an equilibrium is reached, if the ratio between the number of accepted configurations with cost values in a range of $\pm\delta = \frac{1}{2}\sigma$ around the expected average value of the costs $\overline{E}$, and if the total number of generated configurations is convergent [HRS86].

Termination detection: In addition to the physical network of the MPP systems the PARSA algorithm makes use of a virtual tree network [Parix] for initialization and termination detection. As each Processing Element (PE) of the system computes its own SA algorithm, local annealing temperatures ϑ and local cost-values are provided at each node. Let $\overline{E}$ ($\overline{V}$) be the approximated mean value (mean deviation) of the cost-function. Then, the triple $(\overline{\vartheta}, \overline{E}, \overline{V})$ is determined successively from the leaves to the root of the virtual tree. Initial values are gathered from short local pre-runs. The PARSA algorithm terminates if $\overline{\vartheta}$ falls below of a given threshold or if $\overline{V}$ is less than three percent of $\overline{E}$. The second condition indicates that the single cost-values are in a relatively small range around $\overline{E}$ so that a solution approaching the global minimum is reached with high probability.

2.3 Parallelization

The PARSA algorithm was evaluated on two partitionable MPP systems. A GCPP consisting of 64 nodes with 64 MByte memory and two MPC 601 processors, and a GCel consisting of 1024 nodes with 4 MByte memory and single T8 processors. While T8 nodes can communicate with 20 Mbps per link, the communication speed of the MPC 601 nodes is four times higher. Each node performs the algorithm of Fig. 1. The processors communicate solely by message-passing.

At the beginning, each PE starts with its own random layout configuration. As the mean deviation and the annealing temperature at each PE is high, the different SA runs are nearly independent. Thus, up to # PEs areas of the solution space are explored simultaneously. Later on, this behavior is changed. If one PE detects a promising intermediate configuration now, its neighbors can decide to adopt this configuration for themselves and start an independent Markov subchain on it. Thus, once a promising configuration is found, more and more PEs can join the group to explore the surrounding solution space while others still continue on their local chains. In this way, only little effort is wasted with configurations of unpleasant quality. At the end of the computation, the only Markov chain left is the one which had returned the final layout. This relationship is illustrated in Fig. 2.

Talking about promising layouts: Each PE maintains a local data structure $N[0..h]$ with $N[i] = (\mathcal{M}, \vartheta, \overline{V})$. $N[i]$: $1 \leq i \leq h$ stores the cost-values, temperature, and deviation of the i-th neighboring processor of the physical MPP network. The local values are stored in $N[0]$ and m is used to back-up the values of the previous configuration. In line 10, the neighborhood of a PE is investigated. If $I > 0$ but the local Markov subchain is still too short (line 11), or

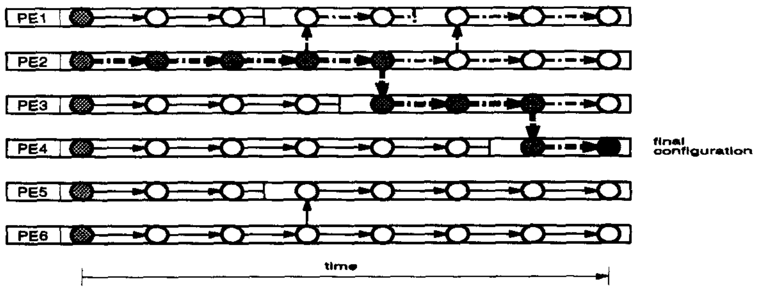


Fig. 2. A sample run with six PEs

the local configuration has currently minimal costs, then the configuration C_k is retained. Otherwise, the configuration from the neighbor PE I is ordered. After some housekeeping, the local data ($\mathcal{N}[0]$) are send to the neighborhood. In case of increased costs (line 19) a configuration is accepted with the probability determined in line 20.

Communication vs. computation: The messages send by the PARSA algorithm can be divided into two classes: Configuration-transfer messages and others, whereby the latter consist of only a small number of bytes. These short messages are send to the physical neighbors if a transition at a PE is completed (line 17, 25). If an equilibrium is reached (line 29), or if the termination condition is fulfilled (line 31), they are send via a virtual tree network (see the virtual topology library of [Parix]). The configuration transfer messages (replies to line 13, 21) depend on G and have a much larger size. In order to reduce the computations necessary when a neighboring configuration is accepted, each vertex in the configuration-transfer message is associated with its partial cost-values. Thus, at the receiving PE, $\mathcal{M}(C_k)$ can be redetermined very fast. To prevent PEs from continuously altering their configurations, a new external one can be accepted only if the local Markov chain is of sufficient length (line 11). If few configurations from neighboring PEs are accepted, many PEs may waste their time by exploring inferior areas of the solution space. On the other hand, if too many configurations from the neighborhood are adopted, the communication overhead becomes dominating, and, because most PEs assume that their neighbors have more promising configurations, significant computation and communication effort is of less value. This behavior is controlled by the procedure *short_subchain*() which checks for the minimum length of the local Markov chain. On our MPP systems, good experiences were made with at least $|V|$ transitions.

3 Properties and Layout Results

Many SA algorithms are characterized by a large number of parameters. Finding the set of values suited best is a highly time-consuming task, even for experts. Thus, a central goal in the design of our algorithm was to keep the number of user-controlled parameters as little as possible. At the end, we were left with only four of those parameters: $\lambda_a, \lambda_e, \lambda_p$ of the cost-function and another one to control the speed of the temperature reduction. The desired edge length L_e has only weak influence on the layout structure. All other parameters were either fixed by a large number of experiments, or were made self-adapting to the problem instance (see 2.2). Initial control values (e.g. $\overline{\vartheta}, \overline{E}, \overline{V}$) are gathered from short pre-runs. This costs additional time, nevertheless, it is indispensable for practical applications.

3.1 Properties

A well designed cost-function is essential for each SA algorithm. It must be calculable very quickly, define a smooth solution space, and approximate the desired aesthetics well. The latter aspect was studied by observing layout sequences of different test-graphs with 'known' optimal layouts. For such graphs, the cost-values of their optimal layouts can be compared to the values of the generated layouts. Fig. 3 shows the result of a sample sequence. The 2% difference from optimum of this sequence, and others, indicates that the algorithm is able to approximate the minimum of the cost-function and the desired aesthetics quite well. This result was confirmed by several other test-sequences, not shown in this report.

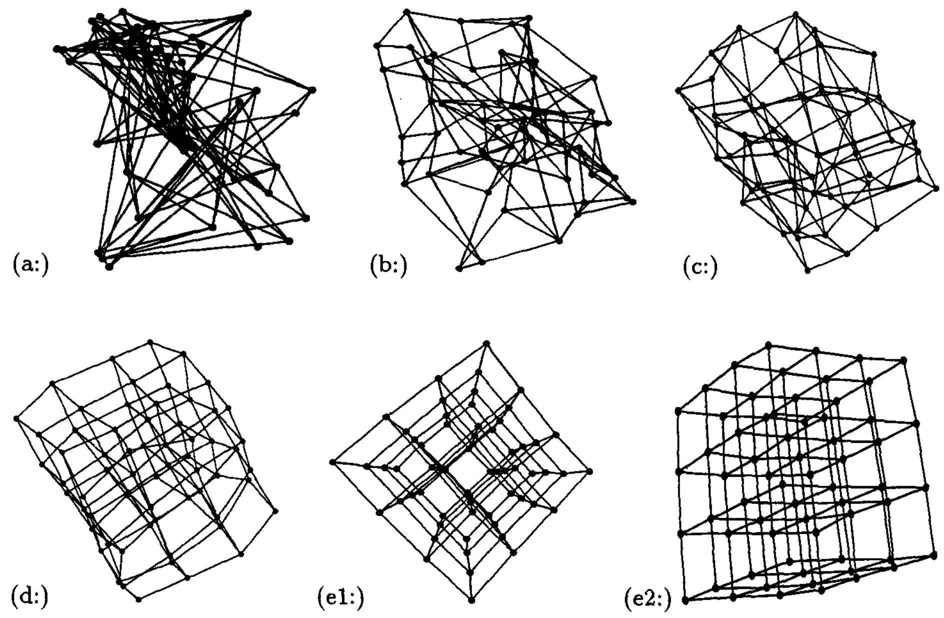
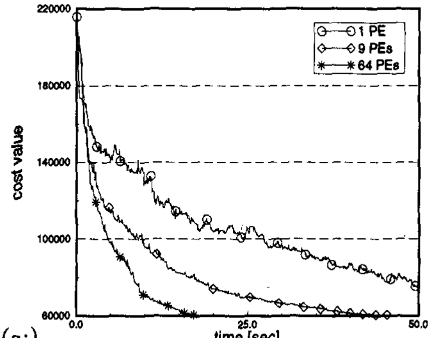


Fig. 3. (a-d:) 3D intermediate states of a 4x4x4-Cube
(e1, e2:) final layout from different angles (2% from optimum)

The evolution of $\mathcal{M}$ and the temperature reduction gives further information on the behavior of the algorithm. To study the former, an 'average' graph

(a:)


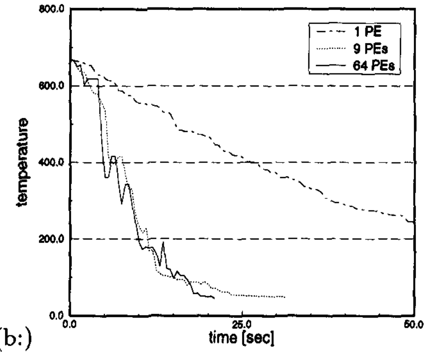
(b:)


Fig. 4. Evolution of the cost-function Temperature reduction

was chosen (see Fig. 5a) and $\mathcal{M}$ was investigated on the GCel system on different network sizes (Fig. 4). During the first 5 seconds, the cost-function drops rapidly. Thus quite often an early impression of the layouts can be obtained very quickly, while it takes much longer to get the required quality. As more PEs can explore a broader area of the solution space simultaneously (Fig. 2), the curves of the larger MPP networks are dropping below the curves of the smaller ones right from the beginning (Fig. 4a). For the test-graph above, a cost-value of about 5 000 is reached when the termination condition is fulfilled. Thus, the advantage of the parallel approach, compared to the sequential case, is obvious. Nevertheless, a limitation on the network size for this graph is clearly indicated. By taking graphs of reasonable larger sizes (e.g. Fig. 8), up to 256 PEs can be exploited successfully.
The limitation for the graph discussed is also confirmed by the temperature reduction curves in Fig. 4b. While the temperature falls slowly at the one PE computation it drops rapidly when exploiting the MPP system. The peaks in the 9 and 64 PE curves result from adoptions of neighboring configurations with a significant difference in the temperature level. These differences, however, become negligible if the cost-function tends to converge.

3.2 Layout results

Due to few objective criteria, it is difficult to evaluate graph drawing algorithms in general. However, a number of test graphs currently substitute as a benchmark suite. They are used to 'measure' the aesthetics and the computation times [KK89, FR91, DH93, Tu93, FLM94, Sa95]. Applying the PARSA algorithm to these graphs, drawings similar to those presented by other authors (Fig. 1, 8, 12 of [DH93], Fig. 17, 18, 47-50, 64, 72 of [FR91] or Fig. 10, 11 of [FLM94]) were obtained. Moreover, the generated 3D layouts of dense graphs can be impressively animated on a computer screen. Our drawing of the common test-graph, mentioned above, is shown in Fig. 5a. Due to the competing aesthetic criteria, vertices were assigned to different levels of the third dimension (e.g. vertices which belong to the inner square were assigned to the top-most level of Fig. 5). Figure 5b shows a 3D layout of a two dimensional grid. These 2D-types of graphs are difficult to draw for any 3D layout algorithm.

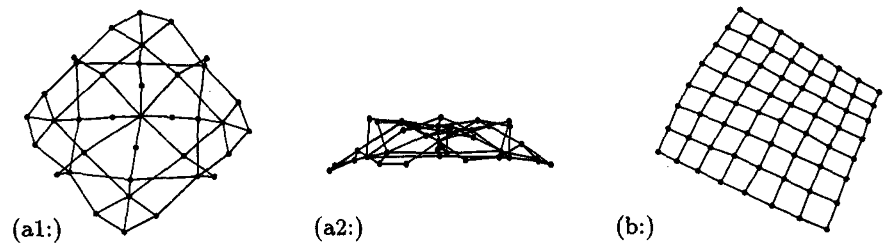


Fig. 5. (a1,a2:) Drawing of Fig. 12 of [DH93] (Fig. 27 of [FR91], Fig. 19 of [FLM94])
(b:) 2D grid, layed out in three dimensions

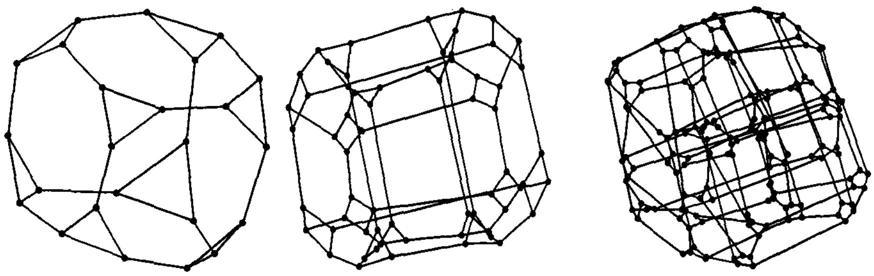


Fig. 6. CCCs of dimension 3, 4 and 5

Fig. 6 shows 3D drawings of Cube-Connected-Circles (CCCs) of dimension three, four and five. These drawings were generated in 7.6, 67 and 620 seconds respectively (on a GCPP with 64 nodes). Within the second figure, the encapsulated cubes with the circles at their corners are easy to recognize. However, for the animation of very complex graphs structures, such as the CCC 5 ($|V| = 160$; $|E| = 240$) the third dimension is indispensable. The CCCs and other more irregular graphs (like the graph 'C' of the GD'95 competition), show the ability of the PARSA algorithm to generate high quality layouts of large graphs with different symmetry-regions. These types of graphs are a challenge for any spring embedder.

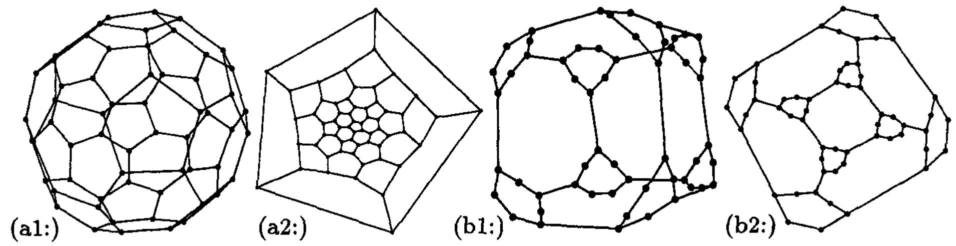


Fig. 7. 3D layouts of soccer ball (a1) and a modified CCC (b1); and its plane projections (a2) and (b2), respectively

Other graphs with inherent 3D structure are the soccer ball (Fig. 7a) with $|V| = 60$; $|E| = 90$) and the modified CCC (Fig. 7b). The former was computed in 58 seconds on average, the latter requires 24 seconds. Its plane projections (Fig. 7 (a2, b2)) were obtained by placing the view-point very close to a face of the 3D layouts of Fig. 7 (a1, b1). Thus, getting a fish-eyes view of the drawing from inside. Due to the 3D layout technique, plane representations for many cubic drawings were attained without any additional effort.

For drawing significantly larger graphs than those presented so far, it is essential that vertices with large distances in G are also placed at distant positions of the layout area. This was achieved by the pseudo-edge-costs of $\mathcal{M}$ (see 2.1). Therefore, it was possible to generate much larger 3D drawings than before. An impression of the aesthetic quality of such large graphs can be obtained from the generated layouts shown in Fig. 8 which was calculated in only 6 minutes. Depending on the characteristics of G speed-up values of up to 20 were measured. This, however, is not the final run-time result. Further potential is expected by moving to the subchain method [DLS93], for the final phase of the parallel SA algorithm.

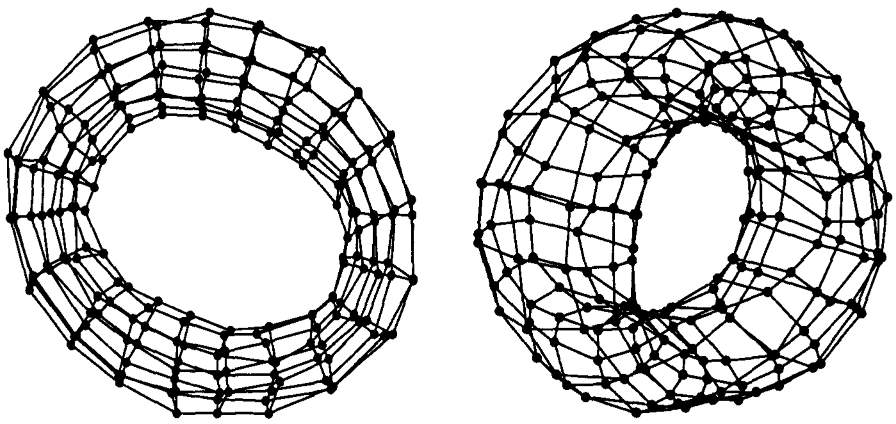


Fig. 8. 3D layout of a 10x20-Torus, computed in 6 minutes ($|V| = 200$; $|E| = 400$)

4 Conclusion

The problem of drawing arbitrary undirected graphs has received an increasing amount of attention during the last few years. However, only few of the published algorithms seem to be applicable in practise. To our knowledge, all of them are either restricted to special graph classes or limited to graphs of relatively small size. In this paper, we have presented a parallel SA algorithm for generating 3D straight-line drawings of arbitrary undirected graphs. Additional structure information is given to human viewers via a simple front-end for 3D graph animations. Due to the cost-function developed and the extended possibilities using a 3D configuration space, it was possible to generate drawings of very different graphs and of much larger size than it was possible before. Because the cooling-schedule and most parameters of the PARSA algorithm were made self-adapting, the presented approach is of high interest for practical applications.

References

[BETT94] G.D. Battista, P. Eades, R. Tamassia, I.G. Tollis : *Algorithms for drawing graphs: An annotated bibliography,* Report, Brown University, June 1994

[CELR94] R.F. Cohen, P. Eades, T. Lin, F. Ruskey : *Three-Dimensional Graph Drawing,* Proc. of Graph Drawing '94, LNCS Springer, Vol. 894, pp. 1-11

[DH93] R. Davidson, D, Harel : *Drawing graphs nicely using simulated annealing,* Technical Report CS89-13, Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Israel 1989, revised July 1993, to appear in Communications of the ACM

[DLS93] R. Diekmann, R. Lüling, J. Simon : *Problem Independent Distributed Simulated Annealing and its Applications,* in: R.V.V. Vidal (ed.): *Applied Simulated Annealing,* Lecture Notes in Economics and Mathematical Systems, Springer 1993, No. 396, pp. 17-44,

[Ea84] P. Eades : *A heuristic for graph drawing,* Congressus Numerantium, 1984, Vol. 42, pp. 149-160

[FPS94] P.W. Fowler, T. Pisanski, J. Shawe-Taylor : *Molecular Graph Eigenvectors for Molecular Coordinates,* Proc. of Graph Drawing '94, LNCS Springer, Vol. 894, pp. 282-285

[FLM94] A. Frick, A. Ludwig, H. Mehldau : *A Fast Adaptive Layout Algorithm for Undirected Graphs,* Proc. of Graph Drawing '94, LNCS Springer, Vol. 894, pp. 388-403

[FR91] T.M.J. Fruchtermann, E.M. Reingold : *Graph drawing by force-directed placement,* Software-Practice and Experience, 1991, Vol. 21, No. 11, pp. 1129-1164

[HS93] D. Harel, M. Sardas : *Randomized Graph Drawing with Heavy-Duty Preprocessing,* Technical Report CS93-16, Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Israel Oct. 1993

[HRS86] M.D. Huang, F. Romeo, A. Sangiovanni-Vincentelli : *An Efficient General Cooling Schedule for Simulated Annealing,* IEEE Int. Conf. on Computer Aided Design 1986, pp. 381-384

[JAMS89-91] D.S. Johnson, C.R. Aragon, L. A. McGeoch, C. Schevon : *Optimization by Simulated Annealing: An Experimental Evaluation,* Part I, "Graph Partitioning", Operations Research Vol. 37, No. 6, pp. 865-892, 1989; *Optimization by Simulated Annealing: An Experimental Evaluation,* Part II, "Graph Coloring and Number Partitioning", Operations Research Vol. 39, No. 3, pp. 378-406, 1991

[KK89] T. Kamada, S. Kawai : *An algorithm for drawing general undirected graphs,* Information Processing Letters, North-Holland 1989, Vol. 31, pp. 7-15

[KMS94] C. Kosak, J. Marks, S. Shieber : *Automating the Layout of Network Diagrams with Specified Visual Organization,* IEEE Trans. on Systems, Man, and Cybernetics, Vol. 24, No. 3, pp. 440-454

[OG89] O.E. Otten, L. van Ginneken : *The Annealing Algorithm,* Kluwer Academic Publishers 1989

[Parix] Parsytec Computer Ltd. : *Parix 1.3: Software Documentation,* Aachen

[Sa95] H. Salmen : *Dreidimensionale Auslegung beliebiger Graphen mittels parallelem Simulated Annealing Methoden,* Master Thesis, Univ. of Paderborn, 1995

[Tu93] D. Tunkelang : *A layout algorithm for undirected graphs,* In Graph Drawing '93, ALCOM Int. Workshop, Paris 1993