Scientia Iranica, Vol. 13, No. 4, pp 387 394

SCIENTIA IRANICA

Drawing Free Trees on 2D Grids Which are Bounded by Simple Polygons

A. Bagheri* **and M. Razzazi**[1]

In this paper, a polyline grid drawing of free trees on two dimensional grids, bounded by simple polygons, is investigated. To the authors' knowledge, this is the first attempt made to develop algorithms for drawing graphs on two dimensional grids bounded by simple polygons.

INTRODUCTION

Graphs are known structures with many applications. Hence, drawing graphs "nicely" has been investigated by many researchers (see [1] for an overview). There are some aesthetics for the nice drawing of graphs that are mentioned in the literature. Some of the most important aesthetics are: Minimizing the number of edge crossings and bends per edge, increasing the symmetry of drawings, maximizing the amount of angle between two incident edges, and distributing vertices uniformly on the drawing regions [1].

While graph drawing on restricted 2D surfaces has many applications, little research has been done on this interesting problem [2]. The existing graph drawing algorithms almost always draw graphs on unbounded planes and a few of them simply draw graphs in regions bounded by rectangles. However, there are applications in which it is desirable or required to draw graphs in regions which are bounded by general polygons. For example, consider a graphical user interface where one would like to show a graph inside a prescribed polygon, or designing PCB of an electronic device which should be T-shaped.

Drawing graphs on a 2D grid with a prescribed size is NP-Hard [3]. In this paper, a polyline grid drawing of free trees on 2D grids bounded by simple polygons is investigated. The straight skeletons of the given polygons and the Simulated Annealing (SA) method is used in the heuristic algorithm, and it is attempted to achieve nice drawings, with respect to the number of edge crossings and uniform node distribution aesthetics. The edges of the given trees may have some bends in the drawings, where the given bounding polygons are non-convex. Davidson and Harel introduced an algorithm for drawing graphs inside given rectangles using the SA method [4], thus called throughout this paper. In order to make possible a comparison of their results with the authors', their algorithm was extended in order to give it the ability of drawing graphs inside simple polygons, which is called the extended SA algorithm throughout this paper.

The experimental results show that the authors' algorithm produces more symmetry, less edge crossings and much more uniform node distribution than the extended SA algorithm. It is believed that this is because of using the geometric properties of the given drawing regions using the authors' algorithm. This paper is an extension of the authors' previous paper [5], which investigates the drawing of free trees on 2D surfaces bounded by rectilinear polygons. The algorithm introduced in [5] also works for simple polygons, but, to make this paper self-contained, it is mentioned here with some minor changes. The simulated annealing method is briefly explained in the following section. Then, the straight skeleton is briefly described and the authors' algorithm is stated. After that, some drawing results of the algorithm inside simple polygons are illustrated and compared to the drawing results of the extended SA algorithm; the experimental results are also analyzed. Finally, a conclusion is stated.

SIMULATED ANNEALING METHOD

The Simulated Annealing (SA) method is a flexible optimization method, suitable for large scale combinatorial optimization problems. The simulated annealing method was originated in statistical mechanics by Metropolis et al. [6], and was formulated in general

*. *Corresponding Author, Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, I.R. Iran.*

1. *Department of Computer Engineering, Institute for Studies in Theoretical Physics and Mathematics (I.P.M.), Tehran, I.R. Iran.*

terms by Kirkpatrick et al. [7]. It has been applied successfully to classical combinatorial optimization problems, such as the traveling salesman problem and problems concerning the design of VLSI. The SA method differs from standard iterative improvement methods by allowing "uphill" moves - moves that spoil, rather than improve, the temporary solution [4].

The problems for which the SA method is useful are characterized by a very large discrete configuration space, too large for an exhaustive search, over which an objective cost function is to be minimized (or maximized). After picking an initial configuration, most 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 (e.g., no move reduces the objective function). The procedure ends in a minimum configuration, but, generally, it is a local minimum rather than the desired global minimum. The SA method tries to escape these local minima by using rules that are derived from an analogy to the process in which liquids are cooled to a crystalline form; a process called annealing [4].

It is well known that when a liquid is cooled slowly, it reaches a totally ordered form, called crystal, which represents the minimum energy state of the system. In contrast, rapid cooling results in amorphous structures that have higher energy, representing local minima. The difference lies in the fact that when a liquid is cooled slowly, the atoms have time to reach a thermal equilibrium at each and every temperature. In this state, the system obeys the Boltzman distribution:

$$P(E) \simeq e^{-E/kT}.$$

Here, $P(E)$ specifies the probability distribution of the energy values of the states, E, as a function of temperature T and the Boltzman constant, k. On the one hand, for every temperature, each energy E has nonzero probability and, thus, the system can change its state to one with higher energy. On the other hand, at a low temperature, the system tends to be in states with very low energy, with the global minimum achieved at temperature zero. Metropolis et al. [6] devised an algorithm for simulating this annealing procedure by a series of sequential moves. The basic rule for when the probability with which the system changes its state from one with energy E_1 to one with energy E_2 is:

$$e^{-(E_2 - E_1)/kT}.$$

This rule implies that whenever the energy, E_2, of the new candidate state is smaller than the current energy, E_1, the system will take the move and if it is larger, the state change is probabilistic. Kirkpatrick et al. [7] were apparently the first to realize that the above procedure could be used for general optimization problems. Several entries must be determined whenever the SA method is applied. These include the following:

- The set of configuration or state of the system, including an initial configuration (which is often chosen at random);
- A generation rule for new configurations, which is usually obtained by defining the neighborhood of each configuration and choosing the next configuration randomly from the neighborhood of the current one;
- The target or cost function, to be minimized over the configuration space (this is an analogue of the energy);
- The cooling schedule of the control parameter, including initial values and rules for when and how to change it. (This is the analogue of the temperature and its decrease.)
- The termination condition, which is usually based on the time and values of the cost function and/or the control parameter.

Having defined all of these, the schematic form of the SA method is as follows. In clause 2(b), random stands for a real number between 0 and 1, selected randomly [4].

1. Choose an initial configuration, S_1, and an initial temperature, T;
2. Repeat the following (usually some fixed number of times):
   a. Choose a new configuration, S_2, from the neighborhood of S_1;
   b. Let E_1 and E_2 be the values of the cost function at S_1 and S_2, respectively. If $E_2 < E_1$ or random $< e^{-(E_1 - E_2)/T}$ then, set $S_1 \leftarrow S_2$.
3. Decrease the temperature, T;
4. If the termination rule is satisfied, stop; otherwise, go back to step 2.

STRAIGHT SKELETONS

There are two types of skeleton for simple polygons, the medial axis and the straight skeleton. The medial axis of a given simple polygon, P, consists of all interior points whose closest point on the boundary of P is not unique [8]. While the medial axis is a Voronoi-diagram-like concept, the straight skeleton is not defined using a distance function, but, rather, by an appropriate shrinking process. The straight skeleton is defined as the union of the pieces of angular bisectors traced out by the polygon vertices during the shrinking process.

Imagine that the boundary of P is contracted towards P's interior, in a self-parallel manner and at the same speed for all edges. Lengths of edges might decrease or increase in this process. Each vertex of P moves along the angular bisector of its incident edges. This situation continues as long as the boundary does not change topologically. There are two possible types of change:

1. Edge Event: An edge shrinks to zero, making its neighboring edges adjacent.
2. Split Event: An edge is split, i.e. a reflex vertex runs into this edge, thus, splitting the whole polygon. New adjacencies occur between the split edge and each of the two edges incident to the reflex vertex.

After either type of event, one is left with one or two new polygons which are shrunk recursively if they have a non-zero area. The straight skeleton, in general, differs from the medial axis. If P is convex, then, both structures are identical; otherwise, the medial axis contains parabolically curved segments around the reflex vertices of P, which are avoided by the straight skeleton. In this paper, the drawing of free trees on 2D grids bounded by general simple polygons is considered and to avoid parabolically curved segments, the straight skeletons are used as the skeletons of polygons. In the sequel, the skeleton means the straight skeleton. The skeleton of a given n-gon P partitions the interior of P into n connected regions, which are called faces. Each face is related to just one edge of P. Bisector pieces are called arcs (or sometimes edges), and their endpoints are called nodes of the skeleton. When P is simple, the structure is a tree [9,10]. Figure 1 shows the straight skeleton of a simple polygon. In this figure, the 4-gon abcd is a face; a, b, c and d are the nodes; and (b,c), (c,d) and (d,a) are the arcs of the skeleton.

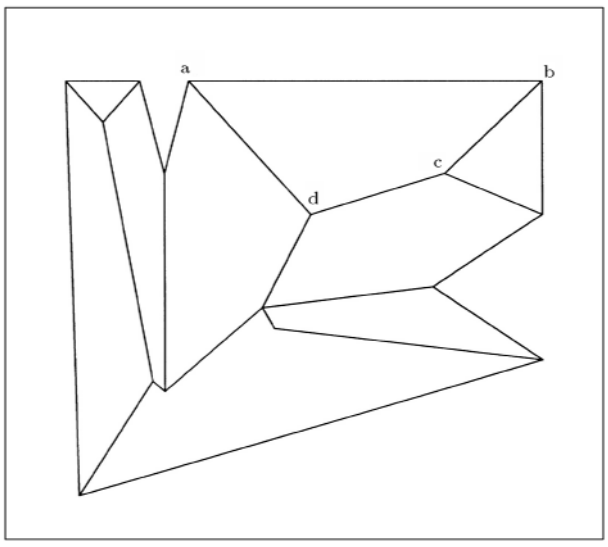

Figure 1. The straight skeleton of a simple polygon.

DRAWING ALGORITHM

In this section, the authors' algorithm is explained and some new definitions used in the algorithm are introduced. (From now on, for simplicity, trees are used for free trees and polygons for simple polygons.) The algorithm produces a polyline grid drawing of the given tree, which is bounded by the given polygon. All the vertices and bends are put on the grid points, but the edge crossings are not restricted to being on the grid points.

Definition 1

For a Tree, T, and its node, i, let Neighbor Nodes $(i;T)$ be the set of all the nodes of T which are adjacent to i.

Considering the tree structure of skeletons, the above definition is also applicable to skeletons.

Definition 2

For a skeleton, S, and its node i, let Neighbor Faces $(i;S)$ be the set of all the faces which have node i as a vertex on their boundary.

Definition 3

For a skeleton, S, and its arc, (i,j), let Neighbor Faces Area (i,j) be the sum of the areas of the faces which lie on the sides of arc (i,j).

Definition 4

For a skeleton, S, and its node, i, $W_{NF}(i;S)$ is defined as follows:

$$W_{NF}(i;S) = \sum_{f \in \text{Neighbor Faces}(i;S)} \text{Area}(f),$$

Area (f) denotes the area of face f.

Definition 5

For a skeleton, S, and its arc, (i,j), $W_{CN}(i|j;S)$ is defined as follows:

$$W_{CN}(i|j;S) = \sum_{m \in \text{Neighbor Nodes}(i;S), m \neq j} W_{CN}(m|i;S) + W_{NF}(i;S)$$

$$\text{Neighbor Faces Area}(i,j).$$

Definition 6

For a Tree, T, and its edge, (i,j), $W_{CN}(i|j;T)$ is defined as follows:

$$W_{CN}(i|j;T) = \sum_{m \in \text{Neighbor Nodes}(i;T), m \neq j} W_{CN}(m|i;T) + 1.$$

Definition 7

For a skeleton, S, and its arc, (i,j), $W_A((i,j);S)$, the weight of arc (i,j), is defined as follows:

$$W_A((i,j);S) = |W_{CN}(i|j;S) \quad W_{CN}(j|i;S)|.$$

The above definition is also applicable to the edges of trees. In the following, the drawing algorithm is described. The pseudo-code of the algorithm is given in Figure 2. First, the straight skeleton of the given polygon and the areas of its faces are computed. Let the boundary of a face be called Pface. The straight skeleton, S, of the given polygon, P, is obtained using [10]. Since all the Pfaces are simple polygons, one can use the formula given in [11] to compute the areas of the faces. Then, for each arc (i,j) of the skeleton, $W_A((i,j);S)$ is computed as the weight of arc (i,j) of S, and, for each edge (i,j) of the given tree, T, $W_A((i,j);T)$ is computed as the weight of edge (i,j) of T.

Definition 8

The middle edge of a skeleton, S, (a tree T) means an edge of the skeleton (the tree) that has the minimum weight of all the other edges of the skeleton (the tree).

Definition 9

A skeleton (a tree) may have more than one middle edge. It can be shown that, in this case, these edges share an endpoint, which is called the middle node.

Definition 10

The middle-connected node means a node that is connected to the middle node by an edge.

The tree is mapped onto the skeleton using the recursive mapping procedure. The pseudo-code of the recursive mapping procedure is given in Figure 3. Using this, the vertices of the tree will be distributed all over the given region. The recursive mapping procedure maps the middle edge or the middle node of the tree onto a proper point of the skeleton in each recursion. This mapping procedure provides a mapping list of the nodes of the tree, which are mapped onto the corresponding points of the skeleton. This mapping list is used by the SA method to spread the vertices of the tree all over the given region. The termination condition of the procedure is satisfied when the number of the nodes of the given tree or the number of the edges of the skeleton becomes less than one. Considering the weighted skeleton and the weighted tree, four cases are possible at each call of the mapping procedure.

Input: A free tree, T, and a simple polygon, P.
Output: A polyline grid drawing of tree T bounded by polygon P.

Begin

1. Compute the polygon skeleton and the area of the faces.
2. Compute the weights of the arcs of the skeleton.
3. Compute the weights of the edges of the tree.
4. Call the mapping procedure for the tree and the skeleton.
5. Remove the possible crossings between the edges of the tree and the sides of the polygon.
6. Draw the tree using the SA method.

End.

Figure 2. The authors' drawing algorithm.

Input: A weighted free tree, T, and a weighted skeleton, S.
Output: A mapping list.
Begin

1. Find the middle edges and the middle nodes of T and S.
2. Add to the mapping list the middle edges and the middle nodes.
3. If Both T and S have just middle edges, then, do the following:
  3.1. Omit these edges from T and S. This divides T into two sub-trees, T_1 and T_2, and S into two sub-skeletons, S_1 and S_2.
  3.2. Update the weights of the sub-trees and the sub-skeletons.
  3.3. Call the mapping procedure for T_1 and S_1.
  3.4. Call the mapping procedure for T_2 and S_2.
4. If Both T and S have middle nodes, then, do the following:
  4.1. Omit the middle node and its incident edges from T and, for S, just disconnect the arcs which are connected at the middle node. This divides T and S into two or more sub-trees and sub-skeletons.
  4.2. Update the weights of the sub-trees and the sub-skeletons.
  4.3. Divide the sub-trees and the sub-skeletons into the same number of balanced groups.
  4.4. Call the mapping procedure for each group of sub-skeletons and its related group of sub-trees.
5. If T has a middle node and S has just a middle edge, then, do the following:
  5.1. Omit the middle node and its incident edges from T and the middle edge from S. This divides T into two or more sub-trees and S into two sub-skeletons.
  5.2. Update the weights of the sub-trees and the sub-skeletons.
  5.3. Divide the sub-trees into two balanced groups.
  5.4. Call the mapping procedure for each group of sub-trees and its related sub-skeleton.
6. If S has a middle node and T has just a middle edge, then, do the following:
  6.1. Disconnect the arcs connected to the middle node of S and omit the middle edge from T. This divides S into two or more sub-skeletons and T into two sub-trees.
  6.2. Update the weights of the sub-trees and the sub-skeletons.
  6.3. Divide the sub-skeletons into two balanced groups.
  6.4. Call the mapping procedure for each group of sub-skeletons and its related sub-tree.

End.

Figure 3. The recursive mapping procedure.

Case 1

In this case, both the tree and the skeleton have middle edges but no middle nodes. The middle edge of the skeleton is mapped onto the middle edge of the tree and these two middle edges are omitted from the skeleton and the tree. This divides the tree and the skeleton, respectively, into two sub-trees and two sub-skeletons.

A record should be added to the mapping list to show that the middle edge of the tree is mapped onto the middle edge of the skeleton. First, the middle

edge of the tree is substituted with a path of length two, whose extreme vertices are the endpoints of the middle edge and whose internal vertex is a dummy vertex. It is recorded in the mapping list that this newly added dummy vertex of the tree is mapped onto the middle point of the related middle edge of the skeleton. After termination of the algorithm, these dummy vertices may appear as bends of the edges of the tree.

The weights of the edges of the two sub-skeletons and the two sub-trees are updated. To do this, consider each sub-skeleton (sub-tree) as a directed tree whose root is the endpoint of the omitted middle edge that is attached to this sub-skeleton (sub-tree) and the edges are directed from the root toward the leaves. Suppose edge (v, u) is the omitted middle edge of skeleton S (tree T). The following pseudo code shows how the weights of the edges of each sub-skeleton of skeleton S are updated. This pseudo-code can be used to update the weights of the edges of each sub-tree of tree T, by replacing S with T:

for each directed edge (i, j) of each sub-skeleton rooted at u, do:

$$\{ \quad W_{CN}(i|j;S) = W_{CN}(i|j;S) \quad W_{CN}(v|u;S)$$
$$\quad W_A((i,j);S) = |W_{CN}(i|j;S) \quad W_{CN}(j|i;S)|$$
$$\}$$

for each directed edge (i, j) of each sub-skeleton rooted at v, do:

$$\{ \quad W_{CN}(i|j;S) = W_{CN}(i|j;S) \quad W_{CN}(u|v;S)$$
$$\quad W_A((i,j);S) = |W_{CN}(i|j;S) \quad W_{CN}(j|i;S)|$$
$$\}$$

The mapping procedure is applied recursively for each sub-skeleton and its related sub-tree.

Case 2

In this case, both the tree and the skeleton have a middle node. The middle node and its incident edges are omitted from the tree. But, for the skeleton, one just disconnects the edges that are connected at the middle node. So, the skeleton and the tree are divided, respectively, into two or more sub-skeletons and sub-trees. It is recorded in the mapping list that the middle node of the tree is mapped onto the related middle node of the skeleton.

The weights of the edges of the sub-skeletons and the sub-trees are updated. To do this, consider each sub-skeleton as a directed tree whose root is the middle node and each sub-tree as a directed tree whose root is a middle-connected node and the edges are directed from the root toward the leaves. Suppose node u is the middle node of skeleton S (tree T). The following pseudo code shows how the weights of the edges of the sub-trees and the sub-skeletons are updated.

$$SUM_T = \sum_{(u,i)\epsilon T} W_{CN}(i|u;T),$$

for each directed edge (i, j) of each sub-tree rooted at middle-connected node v, do:

$$\{ \quad W_{CN}(i|j;T) = W_{CN}(i|j;T) \quad SUM_T + W_{CN}(v|u;T) \quad 1$$
$$\quad W_A((i,j);T) = |W_{CN}(i|j;T) \quad W_{CN}(j|i;T)|$$
$$\}$$

$SUM_S = \sum_{(u,i)\epsilon S} W_{CN}(i|u;S)$

for each directed edge (i, j) of each sub-skeleton, which is rooted at middle node u and includes middle-connected node v, do:

$$\{ \quad W_{CN}(i|j;S) = W_{CN}(i|j;S) \quad SUM_S + W_{CN}(v|u;S)$$
$$\quad W_A((i,j);S) = |W_{CN}(i|j;S) \quad W_{CN}(j|i;S)|$$
$$\}$$

In this case, one may have more than two sub-trees and two sub-skeletons. The sub-skeletons and the sub-trees are divided into the same number of groups of sub-skeletons and sub-trees, which are balanced, as much as possible, with respect to $W_{CN}(v|u;S)$ and $W_{CN}(p|q;T)$, where u is the middle node and v is a middle-connected node of the skeleton, q is the middle node and p is a middle-connected node of the tree. For each group of sub-skeletons and its related group of sub-trees, the mapping procedure is applied recursively.

Case 3

In this case, the tree has a middle node but the skeleton doesn't. The skeleton is divided into two sub-skeletons and the weights of the edges of the sub-skeletons are updated, as in Case 1. The tree is divided into two or more sub-trees and the weights of the edges of the sub-trees are updated, as in Case 2. It is recorded in the mapping list that the middle node of the tree is mapped onto the middle point of the related middle edge of the skeleton. If there are more than two sub-trees, the sub-trees are divided into two groups of sub-trees, which are balanced, as much as possible, with respect to $W_{CN}(v|u;T)$, where u is the middle node of the tree and v is a middle-connected node of the tree. The mapping procedure is applied, recursively, for each group of sub-trees and the related sub-skeleton.

Case 4

In this case, the skeleton has a middle node but the tree doesn't. The tree is divided into two sub-trees and the weights of the edges of the sub-trees are updated, as in Case 1. The skeleton is divided into two or more sub-skeletons and the weights of the edges of the sub-skeletons are updated, as in Case 2. The middle edge of the tree is substituted with a path of length two, whose extreme vertices are the endpoints of the

middle edge and whose internal vertex is a dummy vertex. It is recorded in the mapping list that this newly added dummy vertex of the tree is mapped onto the middle point of the related middle edge of the skeleton. If there are more than two sub-skeletons, the sub-skeletons are divided into two groups of sub-skeletons, which are balanced, as much as possible, with respect to $W_{CN}(v|u;S)$, where u is the middle node of the skeleton and v is a middle-connected node of the skeleton. The mapping procedure is applied, recursively, for each group of sub-skeletons and the related sub-tree.

After mapping of the tree onto the skeleton, the possible crossings between the edges of the tree and the sides of the polygon are removed. Before applying the SA method, all the nodes of the tree that are included in the mapping list, are placed at the related points of the skeleton. Assuming the closest located node of a tree node is an already located node of the tree with the shortest graph-theoretic distance from the given node among all the previously located nodes of the tree, the remaining nodes of the tree are placed at the location of their closest located node. When all the nodes of the tree are initially located, if the given polygon is non-convex, then, there is the possibility of crossing between the edges of the tree and the border of the polygon. If this is the case, these crossings are removed by introducing some dummy nodes and bending the crossing edges of the tree. All the coordinates are made integral by truncating or rounding the coordinates. The resulting configuration is the starting configuration of the SA method.

At the end, the SA method is used to draw the tree inside the given polygon. It is attempted to keep the nodes of the tree close to their corresponding points of the skeleton by adding some new virtual nodes and edges to the given tree. The authors' algorithm guides the SA method, so, the drawing results have fewer edge crossings and show more symmetries than the results of the extension of the algorithm introduced in [4], the extended SA algorithm.

DRAWING RESULTS

In this section, the results obtained from the experiments are presented and to provide an intuitive sample, an example of the drawing results of the algorithm and the extension of the SA algorithm are illustrated. The same parameters and factors are used for the cost function of the SA method as the extended SA algorithm, which is also used by [4].

Figures 4 and 5, respectively, illustrate the drawings of a 31-node complete binary tree inside a W-shaped polygon by the extended SA algorithm and by the authors' algorithm. The size of the bounding rectangle, which includes the W-shaped polygon, is

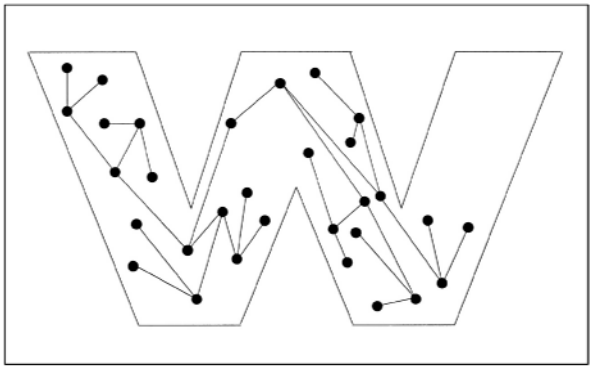

Figure 4. Drawing of a 31-node complete binary tree by the extended algorithm inside a W-shaped polygon.

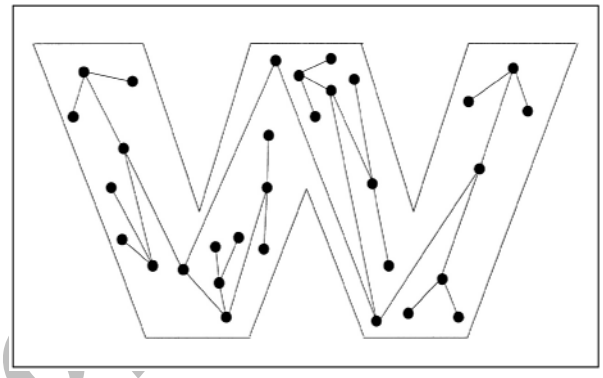

Figure 5. Drawing of a 31-node complete binary tree by the authors' algorithm inside a W-shaped polygon.

200×400. The experimental results show that the algorithm produces much more symmetry and less edge crossings than the extended SA algorithm. The algorithm divides the given tree into some clusters of nodes by means of the recursive mapping procedure and distributes the nodes on the different parts of the given polygon. Because of this, the authors' algorithm produces fewer edge crossings than the extended SA algorithm.

Let the experimental results about the number of edge crossings and the quality of node distribution of the authors' algorithm and the extended SA algorithm be presented. The tests were performed on a PC with a 500 MHZ Intel MMX processor with 64 MB of RAM, running Windows 98. 10 groups of random free trees were generated, each group consisting of 10 trees, 100 trees in total. The trees belonging to group i, $i = 1, \cdots, 10$, had $10 \times i$ nodes. The maximum node degree of the trees was bounded by 10. The five different simple polygons were also considered as the bounding polygons of the drawing regions. The authors' algorithm and the extended SA algorithm were applied to draw these 100 free trees inside these 5 polygons. The tests were repeated 10 times and the average number of edge crossings and the average quality of node distribution were computed. For

evaluating the quality of the node distribution, the reverse of $\sum_{i,j} 1/D_{i,j}^2$ was used, where $D_{i,j}$ is the Euclidean distance between the vertices i and j. This term has been used as the node distribution term in the cost function of the SA method in [4].

Since the results of the experiments on different polygons were similar, for brevity, only one sample is presented here. As an example of the experimental results, one can see Figures 6 and 7, which compare the average number of edge crossings and the average quality of the node distribution of the two algorithms for drawing of a 31-node complete binary tree inside a C-shaped polygon (see Figure 8). Figure 6 presents the ratio of the average number of edge crossings of the extended SA algorithm to the average number of edge crossings of the authors' algorithm. Figure 7 presents the ratio of the average quality of the node distribution of the authors' algorithm to the average quality of the node distribution of the extended SA algorithm. As can be seen from the diagrams, the authors' algorithm produces fewer edge crossings and much more uniform node distribution than the extended SA algorithm.

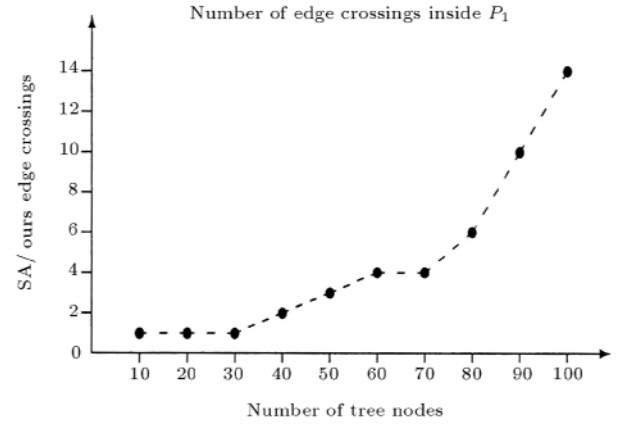

Figure 6. The average number of edge crossings inside polygon P1.

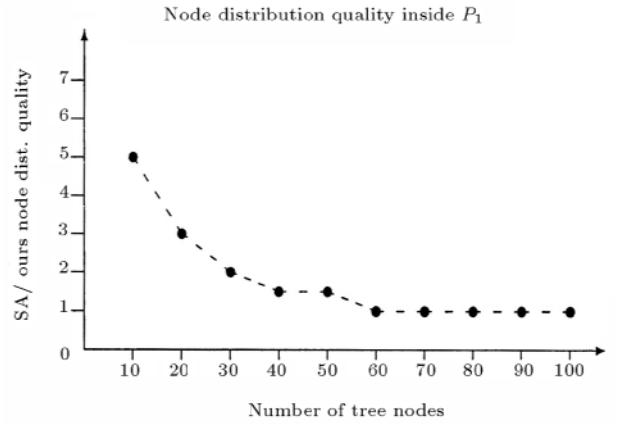

Figure 7. The average quality of node distribution inside polygon P1.

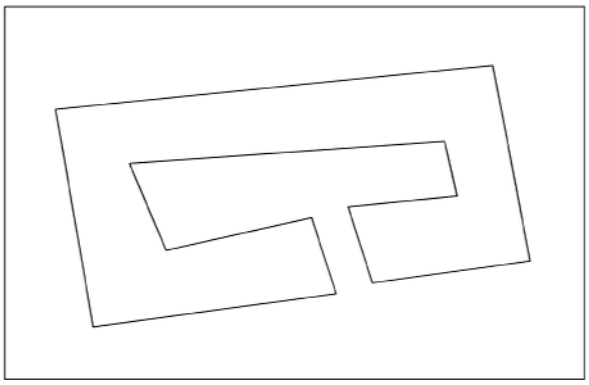

Figure 8. Polygon P1.

This is due to utilizing the geometrical properties of the bounding polygons using the authors' algorithm.

Figure 6 shows that by increasing the number of nodes of given trees while the drawing region is fixed, the ratio of the number of edge crossings of the extended SA algorithm to the number of edge crossings of the authors' algorithm increases. This behavior is expected because the authors' algorithm distributes the nodes of the given trees more intelligently than the extended SA algorithm. In fact, the authors' algorithm clusters the nodes of the tree and draws each cluster on a different part of the given drawing region, which helps to reduce the number of edge crossings. Also, it is obvious that the work will be more difficult for the extended SA algorithm when the congestion of the nodes increases.

Figure 7 shows that by increasing the number of nodes of given trees while the drawing region is fixed, the ratio of the quality of the node distribution of the authors' algorithm to the quality of the node distribution of the extended SA algorithm approaches one. This behavior is also expected, because it is obvious that when the congestion of the nodes increases, the choices for the node placements decrease and both algorithms produce a more similar node distribution.

CONCLUSION

In this paper, the polyline grid drawing of free trees on 2D grids bounded by simple polygons is investigated. The authors' algorithm uses the straight skeletons of the bounding polygons and the simulated annealing method. The drawing results show that the algorithm draws trees nicer than previous known algorithms [4], with respect to the number of edge crossings, symmetry and uniform node distribution. This result is due to utilizing the geometrical properties of the bonding polygons using the authors' algorithm. To the authors' knowledge, these works are the first attempts to develop algorithms for drawing graphs on two dimensional grids bounded by simple polygons [5,12].

ACKNOWLEDGMENT

This research was in part supported by a grant from I.P.M. (No. CS1384-4-05).

REFERENCES

1. Battista, G.D., Eades, P., Tamassia, R. and Tollis, I.G. "Algorithms for drawing graphs: An annotated bibliography", *Comput. Geom. Theory Appl.*, **4**, pp 235-282 (1994).
2. Felsner, S., Liotta, G. and Wismath, S. "Straight-line drawings on restricted integer grids in two and three dimensions", P. Mutzel, M. Junger and S. Leipert, Eds., *Proc. 9th Int. Symp. on Graph Drawing (GD'01), Lecture Notes in Computer Science*, **2265**, Springer, pp 328-342 (2002).
3. Formann, M. and Wagner, F. "The VLSI layout problem in various embedding models", in *Proc. Internat. Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Computer Science*, **484**, pp 130-139 (1991).
4. Davidson, R. and Harel, D. "Drawing graphs nicely using simulated annealing", *ACM Transactions on Graphics*, **15**(4), pp 301-331 (Oct. 1996).
5. Bagheri, A. and Razzazi, M. "How to draw free trees inside bounded rectilinear polygons", *International Journal of Computer Mathematics*, **81**(11), pp 1329-1339 (Nov. 2004).
6. Metropolis,, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. "Equation of state calculations by fast computing machines", *J. Chem. Phys.*, **21**(6), pp 1087-1091 (1953).
7. Kirkpatrick, S., Gelatt, Jr., C.D. and Vecchi, M.P. "Optimization by simulated annealing", *Science*, **220**(4598), pp 671-680 (1983).
8. Chin, F., Snoeyink, J. and Wang, C.A. "Finding the medial axis of a simple polygon in linear time", *Proc. 6th Ann. Int. Symp. Algorithms and Computation (ISAAC 95), Lecture Notes in Computer Science 1004*, pp 383-391 (1995).
9. Aichholzer, O., Aurenhammer, F., Alberts, D. and Gartner, B. "A novel type of skeleton for polygons", *Journal of Universal Computer Science*, **1**(12), pp 752-761 (1995).
10. Felkel, P. and Obdrzalek, S. "Straight Skeleton implementation", *Proceedings of Spring Conference on Computer Graphics*, ISBN 80-223-0837-4, pp 210-218 (1998).
11. Bourke, P. "Calculating the area and centroid of a polygon", http://www.swin.edu.au/astronomy/pbourke/geometry/polyarea/ (1988).
12. Bagheri, A. and Razzazi, M. "Drawing free trees inside convex regions using polygon skeleton", *Pakistan Journal of Applied Sciences*, **2**(1), pp 17-23 (2002).