Journal of Universal Computer Science, vol. 11, no. 6 (2005), 804-829
submitted: 8/1/04, accepted: 16/4/05, appeared: 28/6/05 © J.UCS

How to Draw Free Trees Inside Bounded Simple Polygons

Alireza Bagheri
(Software Systems R&D Laboratory
Department of Computer Engineering & IT
Amirkabir University of Technology, Tehran, Iran
bagheri@ce.aut.ac.ir)

Mohammadreza Razzazi
(Software Systems R&D Laboratory
Department of Computer Engineering & IT
Amirkabir University of Technology, Tehran, Iran
Institute for Studies in Theoretical Physics and Mathematics (I.P.M.)
razzazi@ce.aut.ac.ir)

Abstract: In this paper we investigate polyline grid drawing of free trees on $2D$ grids which are bounded by simple polygons. We focus on achieving uniform node distribution while we also try to achieve minimum edge crossings. We do not consider achieving symmetry as a mandatory task, but our algorithm can exploit some symmetries present in both the given trees and the given polygons. To our knowledge, our work is the first attempt for developing algorithms that draw graphs on regions which are bounded by simple polygons.

Key Words: Computational Geometry, Graph Drawing, Simulated Annealing, Free Trees, Straight Skeleton
Category: I.3.5, I.6.3, G.2.2

1 Introduction

Graph Drawing has many applications, including the design of VLSI layouts, software engineering, database systems, and graphical user interfaces. Hence drawing graphs “nicely” has been investigated by many researchers. There are some aesthetics for nice drawing of graphs that are mentioned in the literature. Some of the most important aesthetics are: *minimizing the number of edge crossings*, *minimizing the number of bends per edge*, *increasing the symmetry of drawing*, *maximizing the angular resolution*, and *distributing the vertices uniformly* [Chan 1999, Purchase 1997].

Most of the graph drawing algorithms draw graphs on unbounded planes, and few of them draw graphs on regions which are bounded by rectangles. But there are some applications in which it is required or desired to draw graphs on regions which are bounded by general polygons. For example, consider a graphics software by which one would like to draw a graph inside a star-shaped polygon, or consider designing PCB of an electronic device which should be U-shaped. In this paper, we investigate polyline grid drawing of free trees inside simple polygons by means of the straight skeletons of polygons [Aichholzer and Aurenhammer 1996, Aichholzer et al. 1995, Felkel and Obdrzalek 1998] and the simulated annealing (SA) method.

More precisely, given a free tree and a simple polygon, we should produce a polyline grid drawing of the given tree which is bounded by the given polygon. The surface which is bounded by the given polygon is called the drawing region. The most important aesthetic which should be satisfied by our drawings is uniform node distribution. The other aesthetic which should be satisfied is minimum edge crossings. Although we do not consider symmetry as a mandatory aesthetic, but our algorithm can exploit some symmetries present in both the tree and the polygon. The edges of the given tree may get some bends in our drawings, if the drawing region is bounded by a non-convex polygon.

A rooted tree is a tree in which a special node is singled out. This node is called the "root" of the tree. Rooted trees are equivalent to oriented trees. A tree which is not rooted is called a free tree. Rooted trees are often used to represent hierarchies such as family trees, organization charts, and search trees, so placing parents above their children is a traditional rule in drawing rooted trees [Di Battista et al. 1994]. Considering this rule when drawing rooted trees inside simple polygons make the task too difficult, for this reason we focus only on drawing of free trees. Ignoring this rule, there is no difference between rooted trees and free trees in our algorithm.

When drawing large graphs, the drawings of the algorithms that use clustering ([Brandenburg 1997], [Brandenburg and Sen 1999], [Edachery et al. 1999], and [Roxborough and Sen 1997]) usually have much fewer edge crossings than the drawing results of those do not use it [Eades et al. 1996, Eades et al. 1999, Eades and Feng 1996]. Our algorithm uses a special kind of clustering and tries to uniformly distribute the vertices of the given tree over the given region by means of the straight skeleton of the bounding polygon.

The simulated annealing method has been already used to draw graphs nicely in the literature. Davidson and Harel in [Davidson and Harel 1996] introduced an algorithm to draw graphs nicely using the SA method, which we call the SA algorithm. The SA algorithm draws graphs inside a given rectangle and similar to ours uses the SA method. To compare our drawing results to those of the SA algorithm, we modified the SA algorithm in order to enable it to draw graphs inside simple polygons. Let us call it the extended SA algorithm. Because of employing geometrical properties of drawing regions, our drawings show the following advantages compared to the drawings of the extended SA algorithm:

- More symmetries

- Less edge crossings
- More uniform node distribution

In section 2, the simulated annealing method is stated. The straight skeleton is briefly described in section 3. In section 4, our algorithm is introduced. In section 5, some drawings of our algorithm are illustrated and compared to the drawings of the extended SA algorithm. The conclusion is stated in section 6.

2 The Simulated Annealing Method

The simulated annealing (SA) method is a flexible optimization method, suited for large scale combinatorial optimization problems. 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. The SA method tries to escape from 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*.

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. 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 low temperature, the system tends to be in states with very low energy, with the global minimum achieved at temperature zero. Metropolis et al. devised an algorithm for simulating this annealing procedure by a series of sequential moves. The basic rule is that 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. 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 states 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 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 the 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 [Davidson and Harel 1996].

```
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 to step 2
```

The SA algorithm uses the above procedure for drawing graphs inside rectangles. It "randomly" places the vertices of the given graph on the surface of the given rectangle, and takes it as the initial configuration of the SA method. For each aesthetic criterion, it defines a term such that minimizing that term satisfies the criterion. It defines the cost function of the SA method as sum of these terms (see [Davidson and Harel 1996] for more details).

3 The Straight Skeletons

There are two types of skeletons for simple polygons, *the medial axis*, and *the straight skeleton*. The medial axis of a given simple polygon P consists of all the interior points whose closest point on the boundary of P is not unique [Chin et al. 1995]. While the medial axis is a voronoi-diagram-like concept, the straight skeleton is not defined by 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. The lengths of the 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 changes:

Edge event: An edge shrinks to zero, making its neighboring edges adjacent.
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, we are left with a new, or two new, polygons which are shrunk recursively if they have 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, we consider drawing of free trees inside general simple polygons, and to avoid parabolically curved segments we use the straight skeletons as the skeletons of polygons. In the sequel, by the skeleton we mean the straight skeleton. The skeleton of a given n-gon P partitions the interior of P into n connected monotone regions which are called *faces*. Each face is swept by just one edge of P during the shrinking process. The bisector pieces are called *arcs* (or sometimes *edges*), and their endpoints are called *nodes*. When P is simple the structure is tree. The skeleton of n-gon P consists of $2n - 2$ nodes and $2n - 3$ arcs ([Aichholzer and Aurenhammer 1996], [Aichholzer et al. 1995], and [Felkel and Obdrzalek 1998]). Figure 1 shows the straight skeleton of a rectangle.

4 Our Algorithm

In this section after introducing some definitions, we explain our algorithm. In the sequel we use term tree for free tree and term polygon for simple polygon. Our algorithm produces a polyline grid drawing of the given free tree which is

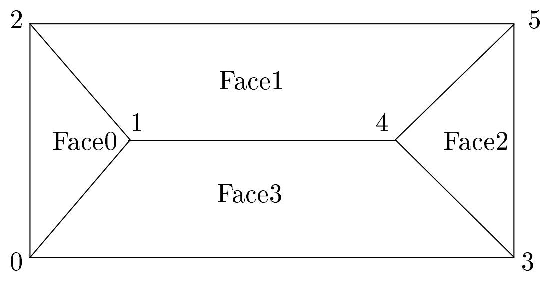

Figure 1: The straight skeleton of a rectangle

bounded by the given polygon. All the vertices and the bends are positioned on the grid points, but this restriction does not apply to the edge crossings.

Definition 1. For skeleton S and its node i, let $FaceSet(i;S)$ be the set of all the faces which have node i as a vertex on their boundary.

Example 1. Consider skeleton S of figure 1, we have $FaceSet(0;S) = \{F0, F3\}$ and $FaceSet(1;S) = \{F0, F1, F3\}$.

Definition 2. For tree T and its edge (i,j), let $CloserSet(i|j;T)$ be the set of all the nodes of the tree whose graph-theoretic distance from node i is shorter than from node j.

Example 2. Consider skeleton S of figure 1, $CloserSet(1|4;S) = \{0,1,2\}$ and $CloserSet(4|1;S) = \{3,4,5\}$.

Considering the tree structure of the skeletons, this definition is also applicable to the skeletons. In the following we describe each step of our algorithm in details. The pseudo-code of our algorithm is as follows.

```
Free Trees Drawing Algorithm
input: an $m$-node free tree $T$ and a simple $n$-gon $P$.
output: a polyline grid drawing of $T$ which is bounded by $P$.

Step a. Computing the polygon skeleton and the area of the faces.
Step b. Computing the weights of the nodes of the skeleton.
Step c. Computing the weights of the edges of the skeleton.
Step d. Computing the weights of the edges of the tree.
Step e. Mapping the tree onto the skeleton.
Step f. Removing the crossings between the tree edges and the polygon sides.
Step g. Drawing the tree using the SA method.
```

Step a. Computing the polygon skeleton and the area of the faces.
We compute straight skeleton S of given polygon P by using the algorithm of [Felkel and Obdrzalek 1998], whose time complexity is $O(n \times r + n \times \log n)$, where r is the number of reflex vertices of P. If P is convex then the time complexity will be $O(n \log n)$, but for general simple polygons it can be as big as $O(n^2)$. Let us call the boundary of faces $Pface$. Since all the Pfaces are simple polygons, we can use the following formula to compute the area of a face f [Bourke 1988].

$$Area(f) = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

Here x_i and y_i are the coordinates of vertex i $(i = 0..n-1)$ of the $Pface$ of face f; $x_n = x_0$ and $y_n = y_0$. To use the above formula, the vertices of the Pfaces should be ordered clockwise or counter clockwise. The time complexity of computing the areas of the faces is $O(n)$, since we have $2n - 3$ arcs and each arc is common between two faces.

Step b. Computing the weights of the nodes of the skeleton.
For each node i of skeleton S we compute the following sum as the weight of node i.

$$W_{IA}(i; S) = \sum_{f \in FaceSet(i;S)} Area(f)$$

$W_{IA}(i; S)$ represents the total amount of the area of the faces which are incident to node i of skeleton S. The time complexity of this step is also $O(n)$, since we have $2n - 3$ arcs and each arc is common between two faces.

Step c. Computing the weights of the edges of the skeleton.
To each endpoint i of every edge (i, j) of skeleton S, we assign a weight which is denoted by $W_{CS}(i|j; S)$. This weight is defined as sum of the weights of the nodes of $CloserSet(i|j; S)$, and shows the approximate area of the polygon which is laid on side i of edge (i, j). The difference of the weights of endpoints i and j is considered to be the weight of edge (i, j) and is denoted by $W_E((i, j); S)$. We can compute the weights of the edges of the skeleton by applying the DFS method twice. So the time complexity of this step is $O(n)$.

$$W_{CS}(i|j; S) = \sum_{m \in CloserSet(i|j;S)} W_{IA}(m; S)$$

$$W_{CS}(j|i; S) = \sum_{m \in CloserSet(j|i;S)} W_{IA}(m; S)$$

$$W_E((i, j); S) = |W_{CS}(i|j; S) - W_{CS}(j|i; S)|$$

Step d. Computing the weights of the edges of the tree.
To each endpoint i of every edge (i, j) of given tree T, we assign a weight which is denoted by $W_{CS}(i|j;T)$. This weight is defined as sum of the weights of the nodes of $CloserSet(i|j;T)$, and shows the number of nodes of the tree which is laid on side i of edge (i, j). The difference of the weights of endpoints i and j is considered to be the weight of edge (i, j). By $|S|$ we mean the cardinality of a set S. For each node i of tree T we define $W_{IA}(i;T) = 1$. The time complexity of this step is also $O(n)$.

$$W_{CS}(i|j;T) = \sum_{m \in CloserSet(i|j;T)} W_{IA}(m;T) = |CloserSet(i|j;T)|$$

$$W_{CS}(j|i;T) = \sum_{m \in CloserSet(j|i;T)} W_{IA}(m;T) = |CloserSet(j|i;T)|$$

$$W_E((i,j);T) = |W_{CS}(i|j;T) - W_{CS}(j|i;T)|$$

Definition 3. By *the middle edge* of skeleton S (tree T) we mean the edge of the skeleton (tree) that has the minimum weight among all the other edges of the skeleton (tree).

Definition 4. A skeleton (tree) may have more than one middle edge. In this case, it is guaranteed by lemma 1 that these edges share an endpoint. This common endpoint is called *the middle node.*

Definition 5. By *the middle-connected node* we mean a node that is connected to the middle node by an edge.

Definition 6. For tree T and its two nodes i and j, let $PathSet((i, j);T)$ be the set of all the nodes of the tree which lie on the path between nodes i and j. This definition is also applicable to the skeletons.

Lemma 1. *If skeleton S (tree T) has more than one middle edge, then these edges share an endpoint.*

Proof. We prove the lemma for skeleton S, a similar proof applies to tree T. Suppose edges (a, b) and (c, d) are two middle edges of skeleton S. If these two middle edges share an endpoint, the lemma is proved. Otherwise, there is at least one edge (e, f) which lies on the path between nodes b and c (see figure 2). Two cases are possible, case I in which $W_{CS}(e|f;S) \geq W_{CS}(f|e;S)$ and case II in which $W_{CS}(e|f;S) < W_{CS}(f|e;S)$. We prove the lemma for case I, the proof is similar for case II.

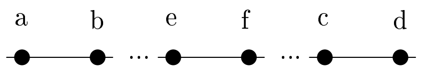

Figure 2: Proof of lemma 1

From the definition we have:

$$W_{CS}(c|d;S) \geq W_{IA}(c;S) + \sum_{i \in PathSet((c,f);S)} W_{IA}(i;S) \\ + W_{IA}(f;S) + W_{CS}(e|f;S) \qquad \Longrightarrow$$

$$W_{CS}(c|d;S) > W_{CS}(e|f;S) \qquad (1)$$

$$W_{CS}(f|e;S) \geq W_{IA}(f;S) + \sum_{i \in PathSet((f,c);S)} W_{IA}(i;S) \\ + W_{IA}(c;S) + W_{CS}(d|c;S) \qquad \Longrightarrow$$

$$W_{CS}(d|c;S) < W_{CS}(f|e;S) \qquad (2)$$

From relations (1) and (2) we have

$$W_{CS}(c|d;S) - W_{CS}(d|c;S) > W_{CS}(e|f;S) - W_{CS}(f|e;S) \geq 0 \qquad \Longrightarrow$$

$$|W_{CS}(c|d;S) - W_{CS}(d|c;S)| > |W_{CS}(e|f;S) - W_{CS}(f|e;S)| \qquad \Longrightarrow$$

$$W_E((c,d);S) > W_E((e,f);S)$$

This contradicts the assumption that edge (c,d) is a middle edge. *Q.E.D.*

Step e. Mapping the tree onto the skeleton.
In this step, we are going to uniformly distribute the vertices of the tree over the given region. To achieve uniform node distribution, we explore the given tree and the given skeleton and try to place the tree nodes on the appropriate parts of the polygon, such that the number of nodes which are laid on a part of the polygon is proportional to the area of that part. We do this by means of a recursive mapping procedure.

The input of the mapping procedure is a set of weighted trees and a set of weighted skeletons, these sets may contain just one member, at least in the first and the last calls. Each input tree (skeleton) has a weight which will be precisely defined in the following, let $W(T)$ $(W(S))$ denotes the weight of input tree T (input skeleton S). The output of the procedure is a mapping list that specifies which nodes of the given tree(s) should be mapped onto which points of the given skeleton(s). This mapping list is used by the SA method to spread the vertices of the given tree over the given region. The termination condition of the procedure is satisfied when the number of nodes of the given tree or the number of edges of the given skeleton is equal to one.

The mapping procedure is as follows. If the input set of trees (skeletons) contains just one tree (skeleton) then we check if it has a middle edge or a middle node, else we suppose the trees (skeletons) of this set belong to a larger tree (skeleton) which has a middle node. In the other words, we suppose the multi-member sets of trees and skeletons have middle nodes. So we are facing with four cases:

Case 1. *The input tree and the input skeleton both have a middle edge.*
Case 2. *The input tree(s) and the input skeleton(s) both have a middle node.*
Case 3. *The input tree(s) has/have a middle node, but the input skeleton has a middle edge.*
Case 4. *The input tree has a middle edge, but the input skeleton(s) has/have a middle node.*

Although we try to distribute the nodes of the given tree uniformly on the surface of the given polygon, but our mapping procedure is not so successful in some cases. In the following we describe how the mapping procedure works in each case, and how much successful it is to satisfy the uniform node distribution criterion.

CASE 1: In this case the input sets of trees and skeletons both have a single member, and their members both have middle edges. Let T and S be the input tree and the input skeleton, respectively. Also let (u, v) and (k, l) denote the middle edge of T and S, respectively. We should record in the mapping list that (u, v) is mapped onto (k, l). To do this, we substitute edge (u, v) with path $u - w - v$, where w is a dummy vertex. We record in the mapping list that w is mapped onto the middle point of edge (k, l). After termination of the algorithm, dummy vertex w may appear as a bend in edge (u, v) of the tree.

Let T_u (T_v) and S_k (S_l) denote the sub-tree and the sub-skeleton induced by $CloserSet(u|v;T)$ $(CloserSet(v|u;T))$ and $CloserSet(k|l;S)$ $(CloserSet(l|k;S))$, respectively. We update the weights of the edges of T_u, T_v, S_k and S_l. To do this, consider T_u, T_v, S_k and S_l as directed trees whose roots are u, v, k and l, respectively, and the edges are directed from parents to children. The following pseudo-code shows how the weights of the edges of sub-tree T_u are updated. We can also use this pseudo-code for updating the weights of the edges of sub-tree T_v by replacing u with v, and for updating the weights of the edges of sub-skeletons S_k and S_l by replacing T, u and v with S, k and l, respectively.

```
for each directed edge $(i, j)$ of sub-tree $T_u$ rooted at $u$ do:
  $W_{CS}(j|i;T_u) = W_{CS}(j|i;T)$
  $W_{CS}(i|j;T_u) = W_{CS}(i|j;T) - W_{CS}(v|u;T)$
  $W_E((i,j);T_u) = |W_{CS}(i|j;T_u) - W_{CS}(j|i;T_u)|$
```

$W(T_u) = W_{CS}(u|v;T)$ ($W(T_v) = W_{CS}(v|u;T)$) and $W(S_k) = W_{CS}(k|l;S)$ ($W(S_l) = W_{CS}(l|k;S)$) denote the weights of sub-tree T_u (T_v) and sub-skeleton S_k (S_l). Suppose $W(T_u) \leq W(T_v)$ and $W(S_k) \leq W(S_l)$. We call the mapping procedure recursively once for T_u and S_k, and once for T_v and S_l. The less value $|W(T_u)/W(S_k) - W(T_v)/W(S_l)|$ is, the more successful the mapping procedure is in satisfying the uniform node distribution criterion and the nicer drawing is yield.

CASE 2: In this case the tree(s) and the skeleton(s) both have a middle node. If the input sets of trees and skeletons "both" have a single member, say T and S then we record in the mapping list that middle node u of T is mapped onto middle node k of S.

If the input set of trees (skeletons) has a single member, say T (S), then we update the weights of the edges of its sub-trees (sub-skeletons) as follows. Let T_v denotes the sub-tree induced by $CloserSet(v|u;T)$, where u is the middle node and v is a middle-connected node of T and $S_{k,l}$ denotes the sub-skeleton induced by $CloserSet(l|k;S) \cup \{k\}$, where k is the middle node and l is a middle-connected node of S. So we have $S_{k,l} = S_l \cup \{(k,l)\} \cup \{k\}$. For each middle-connected node v of T, we update the weights of the edges of sub-tree T_v; and for each middle-connected node l of S, we update the weights of the edges of sub-skeleton $S_{k,l}$. To do this, consider each sub-tree T_v and each sub-skeleton $S_{k,l}$ as a directed tree whose root is v and k, respectively, and the edges are directed from parents to children. The following pseudo-code shows how the weights of the edges of the sub-trees and the sub-skeletons are updated.

```
for each directed edge $(i, j)$ of every sub-tree $T_v$ rooted at
middle-connected node $v$ do:
  $W_{CS}(j|i;T_v) = W_{CS}(j|i;T)$
  $W_{CS}(i|j;T_v) = W_{CS}(i|j;T) - W_{CS}(u|v;T)$
  $W_E((i,j);T_v) = |W_{CS}(i|j;T_v) - W_{CS}(j|i;T_v)|$
```

for each directed edge (i, j) of every sub-skeleton $S_{k,l}$ which is rooted at middle node k and includes middle-connected node l do:

$W_{CS}(j|i; S_{k,l}) = W_{CS}(j|i; S)$
$W_{CS}(i|j; S_{k,l}) = W_{CS}(i|j; S) - W_{CS}(k|l; S) + W_{IA}(k; S)$
$W_E((i, j); S_{k,l}) = |W_{CS}(i|j; S_{k,l}) - W_{CS}(j|i; S_{k,l})|$

Let $W(T_v) = W_{CS}(v|u; T)$ and $W(S_{k,l}) = W_{CS}(l|k; T) + W_{IA}(k; S)$ denote the weights of sub-tree T_v and sub-skeleton $S_{k,l}$, respectively. If the input set of trees (skeletons) has a single member then we assign its sub-trees (sub-skeletons), as defined above, to two sets else we assign the input trees (skeletons) to two sets, then we call the mapping procedure for each set. Precisely speaking, we assign the trees or the sub-trees and the skeletons or the sub-skeletons to two sets of trees, say GT_1 and GT_2, and to two sets of skeletons, say GS_1 and GS_2, respectively, which are balanced as much as possible with respect to sum of the weights of their members. Let $W(GT) = \sum_{T_v \in GT} W(T_v)$ and $W(GS) = \sum_{S_{k,l} \in GS} W(S_{k,l})$. Suppose $W(GT_1) \leq W(GT_2)$ and $W(GS_1) \leq W(GS_2)$. We call the mapping procedure once for GT_1 and GS_1, and once for GT_2 and GS_2. Cleary in some cases, we may get better results if we assign the trees and the skeletons to more than "two" sets, which for simplicity we do not discuss it here. The less value $|W(GT_1)/W(GS_1) - W(GT_2)/W(GS_2)|$ is, the more successful the mapping procedure is in satisfying the uniform node distribution criterion and the nicer drawing is yield.

CASE 3: In this case the tree(s) has/have a middle node, but the skeleton S has a middle edge, say (k, l). Let S_k (S_l) denotes the sub-skeleton induced by $CloserSet(k|l; S)$ ($CloserSet(l|k; S)$). The weights of the edges of sub-skeletons S_k and S_l are updated as in Case 1. Let $W(S_k) = W_{CS}(k|l; S)$ denotes the weight of sub-skeleton S_k. If the input set of trees has a single member, say T, then the weights of the edges of its sub-trees are updated as in Case 2, and we record 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. Let T_v denotes the sub-tree induced by $CloserSet(v|u; T)$, where u is the middle node and v is a middle-connected node of T. Also, let $W(T_v) = W_{CS}(v|u; T)$ denotes the weight of sub-tree T_v.

If the input set of trees has a single member then we assign its sub-trees to two sets else we assign the input trees to two sets, then we call the mapping procedure for each set. Precisely speaking, we assign the trees or the sub-trees to two sets of trees, say GT_1 and GT_2, which are balanced as much as possible with respect to sum of the weights of their members. Let $W(GT) = \sum_{T_v \in GT} W(T_v)$. Suppose $W(GT_1) \leq W(GT_2)$ and $W(S_k) \leq W(S_l)$. We call the mapping procedure once for GT_1 and S_k, and once for GT_2 and S_l. The less

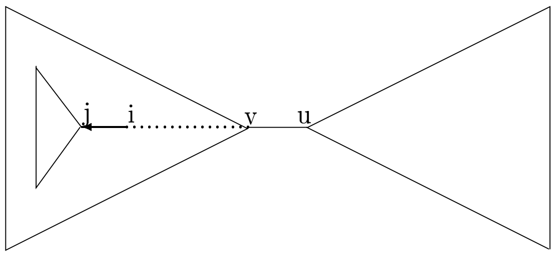

Figure 3: Proof of lemma 2

value $|W(GT_1)/W(S_k) - W(GT_2)/W(S_l)|$ is, the more successful the mapping procedure is in satisfying the uniform node distribution criterion and the nicer drawing is yield.

CASE 4: In this case the skeleton(s) has/have a middle node, but the tree T has a middle edge, say (u, v). Let T_v (T_u) denotes the sub-tree induced by $CloserSet(v|u; T)$ $(CloserSet(u|v; T))$. The weights of the edges of sub-trees T_v and T_u are updated as in Case 1. Let $W(T_u) = W_{CS}(u|v; S)$ denotes the weight of sub-tree T_u. If the input set of skeletons has a single member, say S, then the weights of the edges of its sub-skeletons are updated as in Case 2. Also, we substitute middle edge (u, v) of T with path $u - w - v$, where w is a dummy vertex, and we record in the mapping list that w is mapped onto the middle node of S. Let $S_{k,l}$ denotes the sub-skeleton induced by $CloserSet(l|k; T) \cup \{k\}$, where k is the middle node and l is a middle-connected node of S. Also, let $W(S_{k,l}) = W_{CS}(l|k; S) + W_{IA}(k; S)$ denotes the weight of sub-skeleton $S_{k,l}$.

If the input set of skeletons has a single member then we assign its sub-skeletons to two sets else we assign the input skeletons to two sets, then we call the mapping procedure for each set. Precisely speaking, we assign the skeletons or the sub-skeletons to two sets of skeletons, say GS_1 and GS_2, which are balanced as much as possible with respect to sum of the weights of their members. Let $W(GS) = \sum_{S_{k,l} \in GS} W(S_{k,l})$. Suppose $W(GS_1) \leq W(GS_2)$ and $W(T_u) \leq W(T_v)$. We call the mapping procedure once for T_u and GS_1, and once for T_v and GS_2. The less value $|W(T_u)/W(GS_1) - W(T_v)/W(GS_2)|$ is, the more successful the mapping procedure is in satisfying the uniform node distribution criterion and the nicer drawing is yield.

The following lemma shows that the weights of the edges of the sub-skeletons and the sub-trees are updated correctly in step e. The time complexity of step e is stated by lemma 3.

Lemma 2. *The weights of the edges of the sub-skeletons and the sub-trees are updated correctly in step e.*

Proof. We show that the updating pseudo-codes are correct. In all the four cases of step e, if the input sets of trees or skeletons have more than one member then there is nothing to do, otherwise we are facing with a tree T or a skeleton S, a middle edge (u, v) or a middle node u and a middle-connected node v, and some sub-trees or sub-skeletons whose weights should be updated. We can summarize all the possible cases for updating the weights of the edges of the sub-trees and the sub-skeletons into two cases.

Case I, in which a sub-tree (sub-skeleton) is attached to the other parts of tree T (skeleton S) through edge (u, v) and includes node v but not node u. Considering figure 3, let T_v (S_v) denotes the directed sub-tree (sub-skeleton) induced by $CloserSet(v|u;T)$ $(CloserSet(v|u;S))$ whose root is v and its edges are directed from parents to children. For each directed edge (i, j) of sub-tree T_v we have $CloserSet(j|i;T) = CloserSet(j|i;T_v)$ and $CloserSet(i|j;T) = CloserSet(i|j;T_v) \cup CloserSet(u|v;T)$. Since sets T_v and T_u are distinct, from the definition we have $W_{CS}(i|j;T) = W_{CS}(i|j;T_v) + W_{CS}(u|v;T)$ and $W_{CS}(j|i;T_v) = W_{CS}(j|i;T)$. Because of symmetry, the above statement is correct if edge (i, j) belongs to sub-tree T_u. Also it is correct for sub-skeletons S_v and S_u, since $CloserSet$ and W_{CS} definitions are defined the same for trees and skeletons.

Case II, in which a sub-skeleton is attached to the other parts of skeleton S through edge (u, v) and includes both nodes u and v. Considering figure 3, let $S_{u,v}$ denotes the directed sub-skeleton induced by $CloserSet(v|u;S) \cup \{u\}$ whose root is u and its edges are directed from parents to children. For each directed edge (i, j) of sub-skeleton $S_{u,v}$ we have $CloserSet(j|i;S) = CloserSet(j|i;S_{u,v})$ and $CloserSet(i|j;S) = CloserSet(i|j;S_v) \cup CloserSet(u|v;S)$. We have $S_{u,v} = S_v \cup \{(u, v)\} \cup \{u\}$, so $CloserSet(i|j;S_{u,v}) = CloserSet(i|j;S_v) \cup \{u\}$. Since sets S_v and S_u are distinct, from the definition we have $W_{CS}(i|j;S) = W_{CS}(i|j;S_v) + W_{CS}(u|v;S)$ and $W_{CS}(j|i;S_v) = W_{CS}(j|i;S)$. Then we have $W_{CS}(i|j;S) = W_{CS}(i|j;S_{u,v}) - W_{IA}(u;S) + W_{CS}(u|v;S)$. *Q.E.D.*

Lemma 3. *The time complexity of step e is $O((n+m) \times \log(min\{m, n\}))$, where n is the number of vertices of the given polygon and m is the number of nodes of the given tree.*

Proof. In the four cases of step e, it may be needed to find the middle nodes or the middle edges of the input skeleton and the input tree, which can be done in $O(n + m)$ time. Also, it may be needed to update the weights of the edges of the sub-skeletons and the sub-trees, which can be done in $O(n+m)$ by applying the DFS method once for each sub-skeleton and each sub-tree. We assign the sub-skeletons and the sub-trees to two sets which are balanced and we call the mapping procedure recursively, so the time complexity of step e can be stated by the recurrence equation $T(n,m) = O(n+m) + 2 \times T(n/2, m/2)$. Solving this equation, we have $O((n+m) \times \log(min\{m,n\}))$ as the time complexity of step e. *Q.E.D.*

Step f. Removing the crossings between the edges of the tree and the sides of the polygon.
Before applying the SA method, all the nodes of the tree, which are included in the mapping list are placed at the related points of the skeleton. Let the closest located node of a tree node be an already located node of the tree which has the shortest graph-theoretic distance from the given node, among all the previously located nodes of the tree. All the unlocated nodes of the tree are placed at the locations of their closest located nodes. After all the nodes of the tree are initially located, if the given polygon is non-convex, there is the possibility of crossing between the edges of the tree and the border of the polygon. If this is the case, we remove these crossings by introducing some dummy nodes and bending the crossing edges of the tree. In other words, we lay those segments of the crossing edges of the tree which lie outside the polygon on the boundary of the polygon by adding some dummy nodes and bending the crossing edges of the tree. By applying rounding or truncation we obtain integer coordinates. The resulting configuration is the initial configuration of the SA method.

Step g. Drawing the tree using the SA method.
We use the SA method to draw the tree inside the given polygon. We try to keep the nodes of the tree close to their corresponding points of the skeleton by introducing some virtual fixed nodes and virtual edges. As our algorithm guides the SA method, we could achieve fewer edge crossings, more uniform node distribution, and usually more symmetries than the extended SA algorithm.

5 Drawing Results

In this section, we compare the drawings of our algorithm to those of the extended SA algorithm. In the examples illustrated here, some random free trees which are generated by our random free tree generator program are drawn by our algorithm and the extended SA algorithm. These trees are drawn on a $2D$ grid of size 480×640 which is bounded by some convex, rectilinear and concave polygons. Our algorithm and the extended SA algorithm both use the same terms and factors for the cost function of the SA method. Minimization of this cost function leads to minimization of the number of edge crossings, and uniform node distribution. Let us illustrate some drawings of our algorithm and the extended SA algorithm.

Figure 4 shows the drawings of a 7-node complete binary tree by the extended SA algorithm (4-a) and by our algorithm (4-b). The size of the bounding

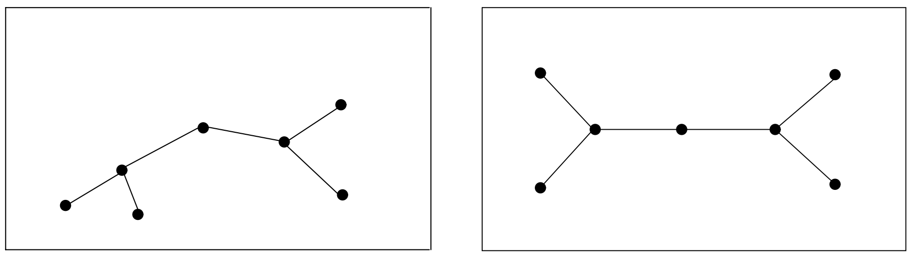

Figure 4: Drawing of a 7-node complete binary tree by the extended SA algorithm (a), and by our algorithm (b).

rectangle is 100×200. As can be seen, our drawing is nicer than the drawing of the extended SA algorithm. This is due to the symmetry and the uniform node distribution of our drawing which is achieved by using the geometrical properties of the bounding polygon by our mapping procedure. In this example, the structure of the given tree completely matches the structure of the skeleton of the given polygon, and because of this we get such a nice drawing.

Although we do not consider symmetry as a predefined goal, but our algorithm can exploit some symmetries present in both the given trees and the given polygons. The straight skeleton of a symmetric polygon is normally symmetric. Also, the definitions of the weights of the edges of the skeleton and the tree reflect the symmetries of the skeleton and the tree. The mapping procedure mapps the nodes of the tree onto the points of the skeleton using these weights, and exploits the symmetries of the polygon and the tree.

Suppose the given polygon and the given tree are symmetrical. If all the nodes of the tree are mapped onto the points of the skeleton of the polygon then our algorithm produces a symmetric drawing. Otherwise, some nodes of the tree remain unmapped and the SA method is free to determine their positions. In this case, the drawing of the mapped nodes are symmetric, but we can not say anything about the symmetry of the drawing of the unmapped nodes.

The drawing of a 63-node complete binary tree by the extended SA algorithm inside a square is shown in figure 5. As can be seen, although the tree is planar, the drawing of it by the extended SA algorithm is not planar. The drawing of this tree by our algorithm is shown in figure 6. The size of the bounding square in these examples is 400×400. Our algorithm divides the given tree into some clusters of nodes and distributes the nodes on different parts of the drawing region. So our drawings usually have fewer edge crossings than the drawings of the extended SA algorithm. In this example, the skeleton has 5 nodes and the

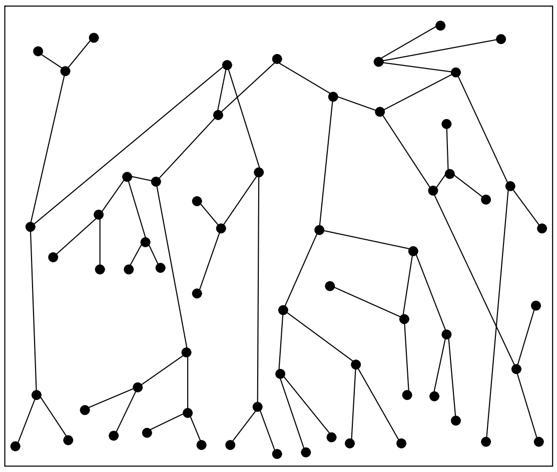

Figure 5: Drawing of a 63-node complete binary tree by the extended SA algorithm inside a square

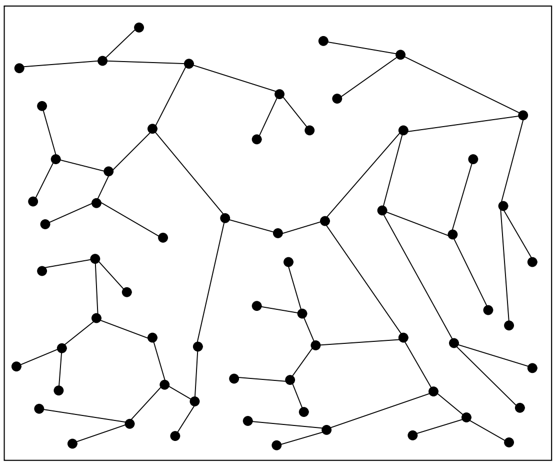

Figure 6: Drawing of a 63-node complete binary tree by our algorithm inside a square

tree has 63 nodes, and just 5 nodes of the tree are mapped onto the skeleton. As can be seen in figure 6, the darwing of the 5 mapped nodes of the tree is symmetric while the drawing of the other unmapped nodes is asymmetric.

Figures 7, 8, 9 and 10 illustrate the drawings of a 31-node complete binary tree inside U-shaped and W-shaped polygons by our algorithm and by the extended SA algorithm. The sizes of the bounding rectangles which includes the

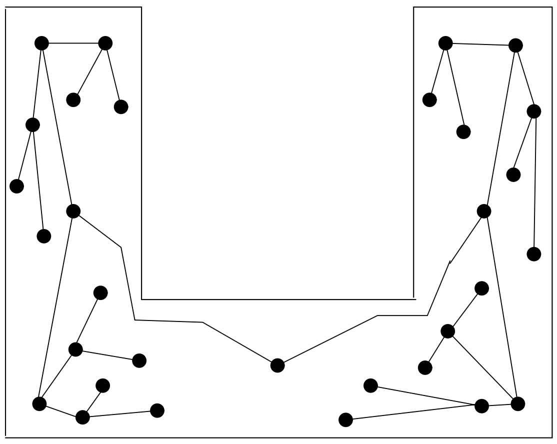
Figure 7: Drawing of a 31-node complete binary tree by our algorithm inside a U-shaped rectilinear polygon

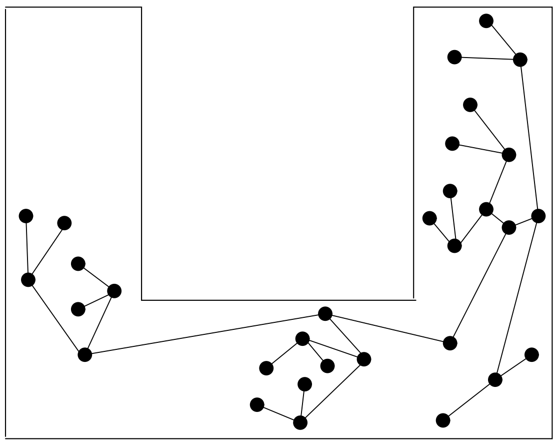
Figure 8: Drawing of a 31-node complete binary tree by the extended SA algorithm inside a U-shaped rectilinear polygon

U-shaped and the W-shaped polygons are 300×400 and 200×400, respectively.

Let us present our experimental results about the running time, the number of edge crossings and the node distribution of our algorithm and the extended SA algorithm. We performed our experiment on a PC with a 500MHZ Intel MMX processor and a 64MB of RAM, running Windows 98. We generated 10 groups of random free trees, each group consisting of 10 trees. The trees belonging to group i, $i = 1, ..., 10$, have $10 \times i$ nodes. The degree of the nodes of the trees is

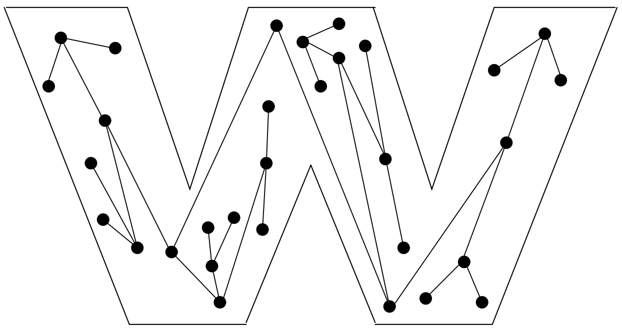
Figure 9: Drawing of a 31-node complete binary tree by our algorithm inside a W-shaped polygon

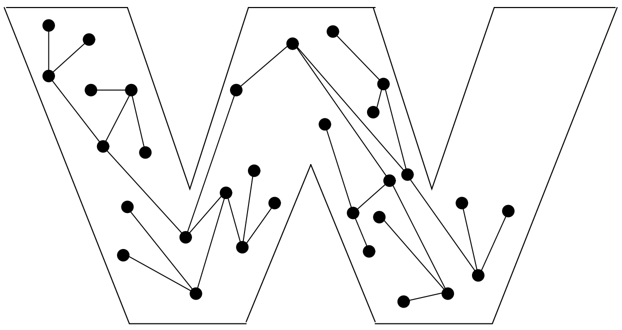
Figure 10: Drawing of a 31-node complete binary tree by the extended algorithm inside a W-shaped polygon

at most 10. We also considered five concave polygons as the bounding polygons.

These polygons are shown in figure 11. We executed our algorithm and the extended SA algorithm to draw these 100 free trees inside these 5 polygons. We repeated the test 10 times and computed the average running time, the average number of edge crossings and the average node distribution. We used the term $\sum_{i,j} 1000/D^2_{i,j}$ for evaluating the node distribution, where $D_{i,j}$ is the Euclidean distance between nodes i and j. This term also has been used to evaluate the node distribution in [Davidson and Harel 1996].

To present a summary of the test results, the results of the test for polygons P_2 and P_5 are shown numerically in tables 5, and 5 and graphically in diagrams 12 through 17. Let R.F.T., A.C.N., A.N.D. and A.R.T. denote Random Free Tree, Average Crossing Number, Average Node Distribution and Average Running Time in seconds, respectively. As can be seen from diagrams 12 through 17,

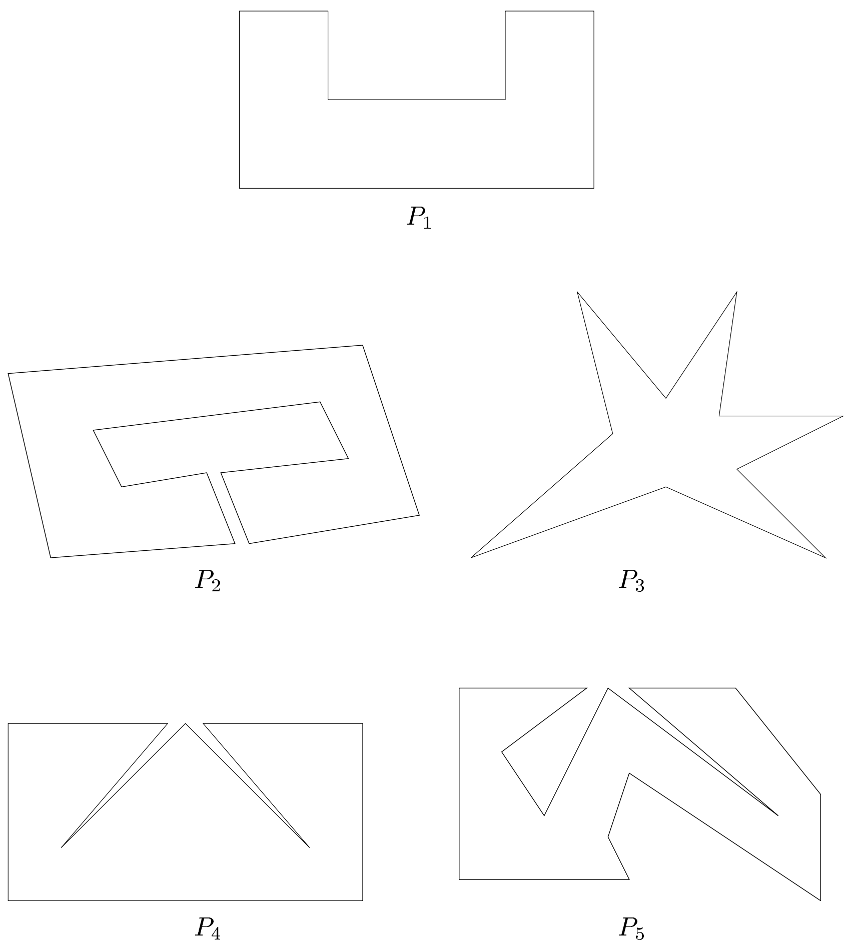

Figure 11: Bounding polygons P_1, P_2, P_3, P_4, and P_5

our drawings have fewer edge crossings and better node distribution than the drawings of the extended SA algorithm. This is due to utilizing the geometrical properties of the bounding polygons by our algorithm. In return, our algorithm uses negligible extra time to do the job, which is due to the more computations that is forced by dummy and virtual vertices and edges which our algorithm adds to the original trees.

	Our Algorithm			Ext. SA Algorithm		
R.F.T.	A.C.N.	A.N.D.	A.R.T	A.C.N.	A.N.D.	A.R.T
10-node	0	3.71	526.14	0	15.53	8.99
20-node	0	34.85	238.74	0	39.10	40.16
30-node	0	56.50	241.70	0	79.71	108.99
40-node	0	105.10	452.81	0	148.38	240.39
50-node	0	182.20	761.86	0.6	266.03	434.83
60-node	0	302.74	1166.71	0.6	388.44	714.51
70-node	0	442.41	1550.43	1.6	731.38	1128.26
80-node	0.2	560.98	2078.19	1.4	971.03	1692.21
90-node	0.2	684.48	3156.20	2	1299.68	2318.37
100-node	0.4	871.30	4637.83	2.2	1545.79	3124.28

Table 1: Experimental results of running our algorithm and the extended SA algorithm for drawing free trees inside polygon P_2

	Our Algorithm			Ext. SA Algorithm		
R.F.T.	A.C.N.	A.N.D.	A.R.T	A.C.N.	A.N.D.	A.R.T
10-node	0.4	3.09	194.57	0	19.35	10.83
20-node	0	33.15	73.31	0	33.08	43.15
30-node	0.6	83.47	185.521	0	99.60	116.75
40-node	0	133.37	425.78	0	160.85	248.80
50-node	0	221.64	629.02	0	267.83	445.22
60-node	0.2	377.26	1025.00	0.2	330.81	769.20
70-node	0	516.97	1411.29	0.6	610.40	1177.23
80-node	0.2	687.66	2244.41	1.6	804.09	1663.45
90-node	0.4	868.57	2798.951	1	1241.22	2445.96
100-node	0.4	1122.46	4386.61	3.8	1508.86	2999.48

Table 2: Experimental results of running our algorithm and the extended SA algorithm for drawing free trees inside polygon P_5

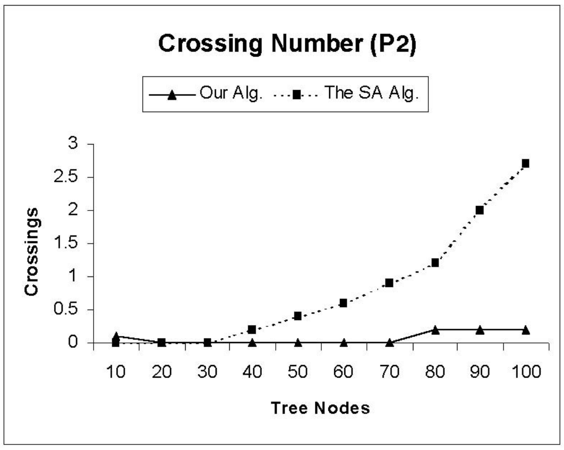

Figure 12: Average Crossing Number of Our Algorithm and the Extended SA Algorithm inside Polygon P_2

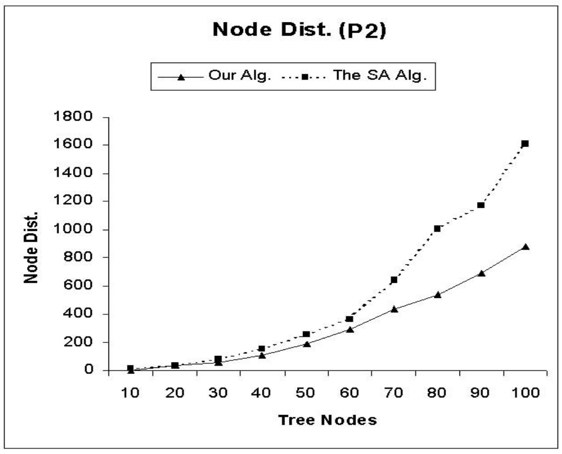

Figure 13: Average Distribution of Our Algorithm and the Extended SA Algorithm inside Polygon P_2

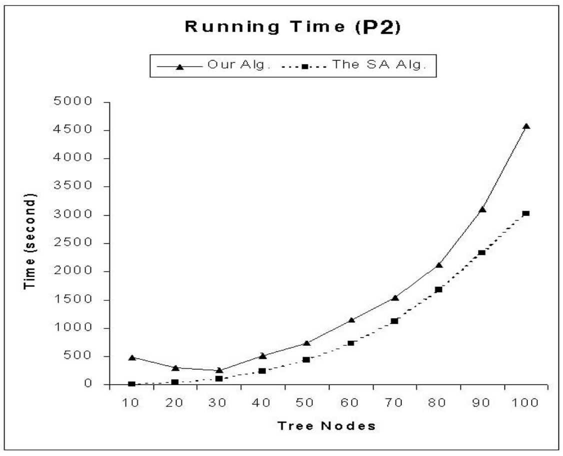

Figure 14: Average Running Time of Our Algorithm and the Extended SA Algorithm inside Polygon P_2

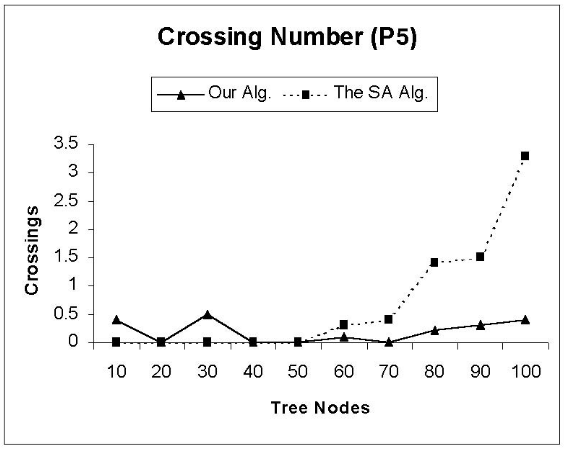

Figure 15: Average Crossing Number of Our Algorithm and the Extended SA Algorithm inside Polygon P_5

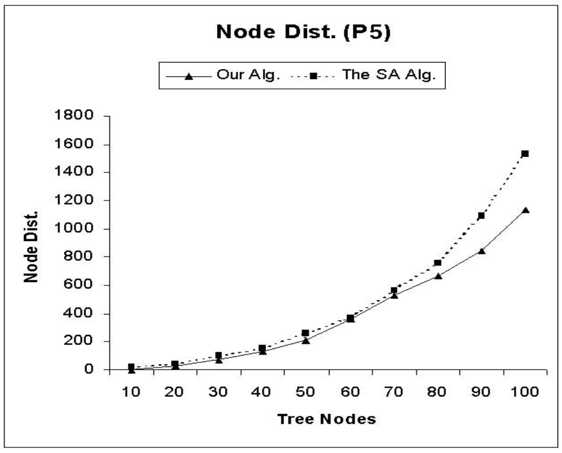

Figure 16: Average Node Distribution of Our Algorithm and the Extended SA Algorithm inside Polygon P_5

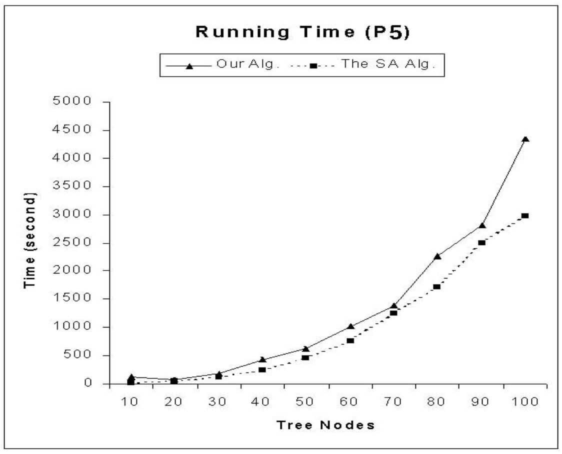

Figure 17: Average Running Time of Our Algorithm and the Extended SA Algorithm inside Polygon P_5

6 Conclusion

In this paper, we investigated polyline grid drawing of free trees on surfaces of general simple polygons by means of the straight skeletons and the simulated annealing method. We focused on achieving uniform node distribution while we also tried to achieve minimum edge crossings. Although, we did not consider achieving symmetry as a mandatory task, but our algorithm could exploit some symmetries presented in both the given trees and the given polygons.

To our knowledge, our work is the first attempt to develop algorithms that draw graphs on $2D$ grids which are bounded by simple polygons. Our experimental results, obtained by running our algorithm and the extended SA algorithm for 100 random free trees, showed that our algorithm produces fewer edge crossings, more uniform node distribution, and usually exploits more symmetries than the extended SA algorithm. The cost of these improvements is the negligible more running time of our algorithm.

Acknowledgement

This research was in part supported by a grant from I.P.M.(No. CS1383-4-1). We would like to thank Oswin Aichholzer (oaich@igi.tu-graz.ac.at), Stepan Obdrzalek (xobdrzal@fel.cvut.cz) and Petr Felkel (felkel@fel.cvut.cz) for allowing us to use their straight skeleton implementations. We also would like to thank Barzan Mozaffari for helping us to implement the algorithms.

References

[Aichholzer and Aurenhammer 1996] Aichholzer, O., and Aurenhammer, F.:"Straight Skeletons for General Polygonal Figures in the Plane", Proc. 2nd Ann. Int. Conf. Computing and Combinatorics (COCOON-96), Lecture Notes in Computer Science (LNCS)1090, Springer , pp. 117-126, 1996.

[Aichholzer et al. 1995] 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.

[Bourke 1988] Bourke, P.:"Calculating the area and centroid of a polygon", http://www.swin.edu.au/astronomy/pbourke/geometry/ polyarea/, 1988.

[Brandenburg 1997] Brandenburg, F. J.:"Graph Clustering I: Cycles of Cliques", Proceedings of Graph Drawing 97, LNCS 1353, 1997.

[Brandenburg and Sen 1999] Brandenburg, F. J. and Sen, A.:"Graph Clustering II: Trees of Cliques with Size Bounds", http://www.infosun.fmi.uni-passau.de/br/lehrstuhlMitarbeiter/Brandenburg/publikationen/, 1999.

[Chan 1999] Chan, T. M.:"A Near-Line Area Bound for Drawing Binary Trees", In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, pp. 161-168, 1999.

[Chin et al. 1995] 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.

[Davidson and Harel 1996] Davidson, R. and Harel, D.: "Drawing Graphs Nicely Using Simulated Annealing", ACM Transactions on Graphics 15(4), pp. 301-331, October 1996.

[Di Battista et al. 1994] Di Battista, G., Eades, P., Tamassia, R. and Tollis, I. G.: "Algorithms for Drawing Graphs: an Annotated Bibliography", Computational Geometry: Theory and Applications 4(5), pp. 235-282, 1994.

[Eades and Feng 1996] Eades, P. and Feng, Q. W.: "Multilevel Visualization of Clustered Graphs", Symposium on Graph Drawing GD-96, ed. S. C. North, LNCS 1190, pp. 101-112, 1996.

[Eades et al. 1996] Eades, P., Feng, Q. W. and Lin, X.: "Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs", Symposium on Graph Drawing GD-96, ed. S. C. North, LNCS 1190, pp. 113-128, 1996.

[Eades et al. 1999] Eades, P., Feng, Q. W. and Nagamochi, H.: "Drawing Clustered Graphs on an Orthogonal Grid", Journal of Graph Algorithms and Applications 3(4), http://www.cs.brown.edu/publications/jgaa/, pp. 3-29, 1999.

[Edachery et al. 1999] Edachery, J., Sen, A. and Brandenburg, F. J.: "Graph Clustering Using Distance-K Cliques", 7th International Symposium on Graph Drawing (GD-99), LNCS 1731, pp. 98-106, September 1999.

[Felkel and Obdrzalek 1998] Felkel, P. and Obdrzalek, S.: "Straight Skeleton Implementation", Proceedings of Spring Conference on Computer Graphics, ISBN 80-223-0837-4, pp. 210-218, http://www.cgg.cvut.cz/Publications/felkel-sccg-98.ps.gz, 1998.

[Purchase 1997] Purchase, H.: "Which Aesthetic Has the Greatest Effect on Human Understanding?", 5th Symposium on Graph Drawing, Rome, Italy, LNCS 1353, pp. 248-261, September 18-20 1997.

[Roxborough and Sen 1997] Roxborough, T. and Sen, A.: "Graph Clustering Using Multiway Ratio Cut", Proceedings of Graph Drawing Symposium GD-97, LNCS 1353, Rome, pp. 291-296, September 1997.