Journal of Graph Algorithms and Applications
http://www.cs.brown.edu/publications/jgaa/
vol. 4, no. 3, pp. 19–46 (2000)

Balanced Aspect Ratio Trees and Their Use for Drawing Large Graphs

Christian A. Duncan
Max-Planck-Institut für Informatik
Saarbrücken, Germany
http://www.mpi-sb.mpg.de/~ duncan
christian.duncan@mpi-sb.mpg.de

Michael T. Goodrich   *Stephen G. Kobourov*
Center for Geometric Computing
The Johns Hopkins University
Baltimore, MD 21218
http://www.cs.jhu.edu/labs/cgc/
goodrich@cs.jhu.edu   kobourov@cs.jhu.edu

Abstract

We describe a new approach for cluster-based drawing of large graphs, which obtains clusters by using binary space partition (BSP) trees. We also introduce a novel BSP-type decomposition, called the *balanced aspect ratio* (BAR) tree, which guarantees that the cells produced are convex and have bounded aspect ratios. In addition, the tree depth is $O(\log n)$, and its construction takes $O(n \log n)$ time, where n is the number of points. We show that the BAR tree can be used to recursively divide a graph embedded in the plane into subgraphs of roughly equal size, such that the drawing of each subgraph has a *balanced aspect ratio*. As a result, we obtain a representation of a graph as a collection of $O(\log n)$ layers, where each succeeding layer represents the graph in an increasing level of detail. The overall running time of the algorithm is $O(n \log n + m + D_0(G))$, where n and m are the number of vertices and edges of the graph G, and $D_0(G)$ is the time it takes to obtain an initial embedding of G in the plane. In particular, if the graph is planar each layer is a graph drawn with straight lines and without crossings on the $n \times n$ grid and the running time reduces to $O(n \log n)$.

Communicated by G. Liotta and S. H. Whitesides: submitted November 1998; revised November 1999.

Research supported in part by ARO grant DAAH04–96–1–0013 and NSF grant CCR-9732300.

1 Introduction

In the past decade hundreds of graph drawing algorithms have been developed (e.g., see [7, 8]), and research in methods for visually representing graphical information is now a thriving area with several different emphases. One general emphasis in graph drawing research is directed at algorithms that display an entire graph, with each vertex and edge explicitly depicted. Such drawings have the advantage of showing the global structure of the graph. A disadvantage, however, is that they can be cluttered for drawings of large graphs, where details are typically hard to discern. For example, such drawings are inappropriate for display on a computer screen any time the number of vertices is more than the number of pixels on the screen. For this reason, there is a growing emphasis in graph drawing research on algorithms that do not draw an entire graph, but instead partially draw a graph, either by showing high-level structures and allowing users to "zoom in" on areas of interest, or by showing substructures of the graph and allowing users to "scroll" from one area of the graph to another. Such approaches are well suited for displaying large graphs, such as significant portions of the world wide web graph, where every web page is a vertex and every hyper-link is an edge.

A common technique used for scrolling viewpoints is the *fish-eye view* [16, 18, 27], which shows an area of interest quite large and detailed (such as nodes representing a user's web pages) and shows other areas successively smaller and in less detail (such as nodes representing a user's department and organization web pages). Fish-eye views allow a user to understand the structure of a graph near a specific set of nodes, but they often do not display global structures.

An alternate technique displays the global structure present in a graph by clustering smaller subgraphs and drawing these subgraphs as single nodes or filled-in regions. By grouping vertices together into *clusters*, we can recursively divide a given graph into layers of increasing detail. These layers can then be viewed in a top-down fashion or even in fish-eye view by following a single path in a cluster-based recursion tree. If clusters of a graph are given as input along with the graph itself, then several authors give various algorithms for displaying these clusters in two or three dimensions [10, 11, 13, 14, 24, 31]. If, as will often be the case, clusters of a graph are not given *a priori*, then various heuristics can be applied for finding clusters using properties such as connectivity, cluster size, geometric proximity, or statistical variation [1, 17, 23, 25]. Once a clustering has been determined, we can generate the layers in a hierarchical drawing of the graph, with the layer depth (i.e., number of layers) being determined by the depth of the recursive clustering hierarchy. This approach allows the graph to be represented by a sequence of drawings of increasing detail. As illustrated by Eades and Feng [10], this hierarchical approach to drawing large graphs can be very effective. Thus, our interest in this paper is to further the study of methods for producing good graph clusterings that can be used for graph drawing purposes.

We feel that a good clustering algorithm and its associated drawing method should come as close as possible to achieving the following goals:

1. *Balanced clustering*: in each level of the hierarchy the size of the clusters should be about the same.
2. *Small cluster depth*: there should be a small number of layers in the recursive decomposition.
3. *Convex cluster drawings*: the drawing of each cluster should fit in a simple convex region, which we call the *cluster region* for that subgraph.
4. *Balanced aspect ratio*: cluster regions should not be too "skinny".
5. *Efficiency*: computing the clustering and its associated drawing should not take too long.

In this paper we study how well we can achieve these goals for large graph drawings using clustering. Previous algorithms optimize one or more of the above criteria at the expense of some of the rest. Our goal is to simultaneously satisfy all of them. Our approach relies on creating the clusters using binary space partition (BSP) trees, defined by recursively cutting regions with straight lines.

1.1 BSP Tree Based Clustered Graph Drawing

The main idea behind the use of a BSP tree in $\mathbb{R}^2$ to define clusters is very simple. Given a graph $G = (V, E)$, where $n = |V|$ and $m = |E|$, we can use any existing method to embed it in the plane, provided that method places vertices at distinct points in the plane (e.g., see [7, 20, 32]). For example, if G is planar we can use any existing method for embedding G in the plane such that vertices are at grid points, and edges of the graph are straight lines that do not cross [6, 12, 28, 30, 33]. Once the graph drawing is defined, we build a binary space partition tree on the vertices of this drawing. Each node v in this tree corresponds to a convex region R of the plane, and associated with v is a line that separates R into two regions, each of which are associated with a child of v. Thus, any such BSP tree defined on the points corresponding to vertices of G naturally defines a hierarchical clustering of the nodes of G. Such a clustering could then be used, for example, with an algorithm like that of Eades and Feng [10], who present a technique for drawing a 3-dimensional representation of a clustered graph.

The main problem with using BSP trees to define clusters for a graph drawing algorithm is that previous methods for constructing BSP trees do not give rise to clustered drawings that achieve the design goals listed above. For example, the standard *k-d* tree and its variants (e.g., see [15, 26]), which use axis-parallel lines to recursively divide the number of points in a region in half, maintain every criteria but the balanced aspect ratio. Likewise, quad-trees and fair-split trees (e.g., see [4, 26]), which always split by a line parallel to a coordinate axis to recursively divide the area of a region more or less in half, maintain balanced aspect ratio but can have a depth that is $\Theta(n)$.

In graph drawing, aesthetics are very important, and while "fat" regions appear rounder, a series of skinny regions can be distracting. But depth is also important, for a deep hierarchy of clusterings would be computationally expensive to traverse and would not provide very balanced clusters. The balanced box-decomposition tree of Arya et al. [3, 2] has $O(\log n)$ depth and has regions with good aspect ratio, but it sacrifices convexity by introducing holes into the middle of regions, which makes this data structure less attractive for use in clustering for graph drawing applications. Indeed, to our knowledge, there is no previous BSP-type hierarchical decomposition tree that achieves all of the above design goals.

1.2 The Balanced Aspect Ratio (BAR) Tree

In this paper we present a new type of binary space partition tree that is better suited for the application of defining clusters in a large graph. Our data structure, which we call the *balanced aspect ratio* (BAR) tree, is a BSP-type decomposition tree that has $O(\log n)$ depth and creates convex regions with bounded aspect ratio (also called "fat" regions). In this paper we present the BAR tree in $\mathbb{R}^2$. The generalized BAR tree in $\mathbb{R}^d$ is presented in [9]. The construction of the BAR tree is very similar to that of a *k*-*d* tree, except for two important differences:

1. In addition to axis-aligned cuts, the BAR tree allows for one more cut direction: a 45°-angled cut.

2. Rather than insisting that the number of points in a region be cut in half at every level, the BAR tree guarantees that the number of points is cut roughly in half every two levels, which is something that does not seem possible to do with either a *k*-*d* tree or a quadtree (or even a hybrid of the two) while guaranteeing regions with bounded aspect ratios.

In short, the BAR tree is an $O(\log n)$-depth BSP-type data structure that creates fat, convex regions. Thus, the BAR tree is "balanced" in two ways: on the one hand, clusters on the same level have roughly the same number of points, and, on the other hand, each cluster region has a bounded aspect ratio.

We show that a BAR tree achieves this combined set of goals by proving the existence of a cut, which we call a *two-cut*. A two-cut might not reduce the point size by any amount but maintains balanced aspect ratio and ensures the existence of a subsequent cut, which we call a *one-cut*, that both maintains good aspect ratio *and* reduces the point size by at least two-thirds. In Section 3, we formally define one- and two-cuts and describe how to construct a BAR tree.

1.3 Our Results for Cluster-Based Graph Drawing

In Section 4, we show how to use the BAR tree in a cluster-based graph drawing algorithm. The Large Graph Drawing (`LGD`) algorithm runs in $O(n \log n + m + D_0(G))$ time, where n and m are the number of vertices and edges in the graph G and $D_0(G)$ is the time to embed G in the plane. If the graph is planar,

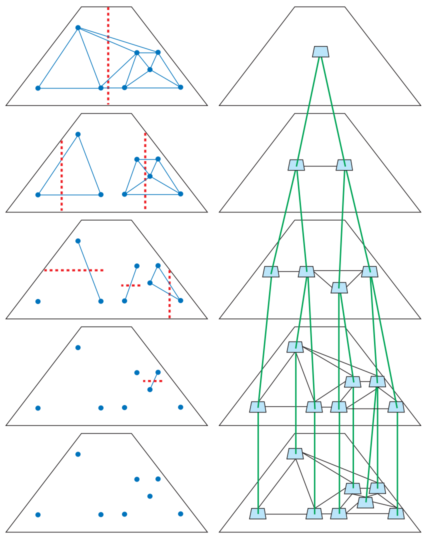

Figure 1: A clustered graph $C = (G, T)$. The underlying graph G is at the lowest level on the right. The clustering of G on the right is obtained from the BSP cuts on the left. Each cluster is represented by a single node. Edges between layers on the right are edges of the tree T.

the algorithm introduces no edge crossings and the running time reduces to $O(n \log n)$.

The algorithm creates a hierarchical cluster representation of a graph, with balanced clusters at each layer and with cluster depth $O(\log n)$. Each cluster region has a *balanced aspect ratio*, guaranteed by the BAR tree data structure. In the actual display of the clustered graph we represent the clusters either by their convex hulls, or by a larger region defined by the BSP tree, or simply by a single node, see Figure 1.

2 Using a BSP Tree for Cluster Drawing

Let $G = (V, E)$ be the graph that we want to draw, where $|V| = n$ and $|E| = m$. Note that graph G is given combinatorially, i.e., defined by the order of the neighbors around each vertex. An embedding of G also assigns distinct coordinates in $\mathbb{R}^2$ for every vertex $v \in V(G)$. The edges of the graph are drawn as straight lines. For the rest of this paper, we assume that the vertices of G have integer coordinates, that is, the graph is embedded on the integer grid.

The goal of our `LGD` algorithm is to produce a representation of the graph G given a BSP tree T, see Figure 1. Similar to [10] we define the *clustered graph* $C = (G, T)$ to be the graph G, and the BSP tree T, such that the vertices of G coincide with the leaves of T. An internal node of T represents a cluster, which

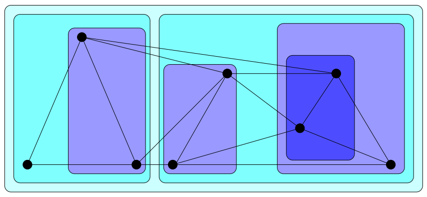

Figure 2: A 2-dimensional representation of a clustered graph $C = (G, T)$. The underlying graph G and the clustering are the same as in Figure 1. a simple closed curve.

consists of all the vertices in its subtree. All the nodes of T at a given depth i represent the clusters of that level.

A *view at level* i, $G_i = (V_i, E_i)$, consists of the nodes of depth i in T and a set of representative edges, for $0 \leq i \leq \texttt{depth}(T)$. An edge (u, v) belongs to E_i if there is an edge between a and b in G, where a is in the subtree of u and b is in the subtree of v. In addition, each node $u \in T$ has an associated region, corresponding to the partition given by T. In Figure 1 we show an example of a 3-dimensional representation of a graph G and in Figure 2 we show a 2-dimensional representation of the same graph.

We create the graphs G_i in a bottom-up fashion, starting with G_k and going all the way up to G_0, where $k = \texttt{depth}(T)$. Define the combinatorial graph $H = (V(H), E(H))$, where initially $V(H) = \{u \in T : \texttt{depth}(u) = k\}$ and $E(H) = E(G)$. Notice that H is well defined since the leaves of T are exactly the vertices of G.

At each new level i we perform a *shrinking* of H. Suppose $u, v \in V(H)$, and $\texttt{parent}(u) = \texttt{parent}(v)$. We replace the pair by their parent and remove the edge (u, v) if it exists. We also remove any multiple edges that this operation may have created and maintain for each surviving edge a pointer to the original edge in G. Thus a *shrinking* of the graph H consists of all such operations, necessary to transform H into a representation of G at one higher level in the tree T.

At each level G_i is a subgraph of G with certain edges removed. Since we are producing a representation of G in 3-dimensions, every vertex must have three coordinates. The first two coordinates correspond to the location of the vertex on the integer grid. The third coordinate of a vertex $v \in V_i$ is equal to i, that is, all the vertices in G_i are embedded in the plane given by $z = i$. To obtain G_i from G_{i+1}, for $i = 0, \ldots, k-1$, we use the combinatorial graph H from level $i+1$. Initially $E_i = E_{i+1}$. We then perform a shrinking of H and while removing an edge from H we remove its associated edge from E_i.

Thus the algorithm on Figure 3 runs in $O(n \cdot \texttt{depth}(T) + m)$ time. Using any of the previous known types of BSP trees, we can maintain most but never all of the desired properties. For example, if T is a *k-d* tree the cluster regions do not have balanced aspect ratios. We next describe how to construct a BSP tree which satisfies all of our goal criteria.

```
create_clustered_graph($T, G$)
    $H \leftarrow G$
    $k \leftarrow$ depth($T$)
    for $i = k$ downto $0$
        obtain $G_i$ from $H$
        shrink $H$
    return $C$
```

Figure 3: Given graph G embedded in the plane and BSP tree T create clustered graph C. Here H is a combinatorial graph initially the same as G. The operations of obtaining G_i from H and shrinking of H are defined in Section 2.

3 The BAR tree

Let us now discuss in detail the definition of our particular BSP-type decomposition tree, the BAR tree, and its construction. We begin with some general definitions.

Definition 1 *The following terms relate to various potential cuts:*

- *A canonical cut direction is any of the following three vectors:*

$$\vec{v}_x = (1,0), \vec{v}_y = (0,1), \vec{v}_z = (1,-1).$$

- *A canonical cut is any line whose normal is a canonical cut direction. For example, the line $x - y = 3$ has normal $\vec{v}_z$.*
- *A canonical region is any convex polygon such that each side is a segment of a canonical cut.*

Since there are three cut directions[1], a canonical region can have at most six sides. For convenience, we define six labels representing the six sides of the polygon. Notice that some of these sides may have zero length. For a canonical region R, we let x^l and x^r represent the corresponding left and right sides of R with normal $\vec{v}_x$. Similarly, we define y^l, y^r, z^l, and z^r, see Figure 4.

Definition 2 *For a canonical region R, let $\texttt{diam}_i(R)$ be the L_m metric distance between the two sides of R with normal $\vec{v}_i$. For a side l in R, we define $|l|$ to be the length of the line segment l measured in the L_m metric.*

For simplicity in our arguments and notation, we use the L_∞ metric although any of the standard L_m metrics is acceptable. In the L_∞ metric the distance between two lines normal to $\vec{v}_z$ and the length of a line segment normal to $\vec{v}_z$ are

[1]Note the assymetry of not having the canonical direction $\vec{v}_w = (1,1)$. The arguments that rely on the three canonical directions above also hold if we add this fourth direction, or any others.

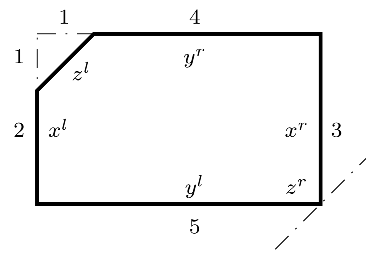

Figure 4: A labelling of the various sides of a canonical region R.

defined differently than in the L_2 metric. In particular, for a canonical region R with sides z^l and z^r, the length $|z^l|$ (or $|z^r|$) is the vertical distance between the two endpoints. The distance between the lines associated with z^l and z^r is one half the vertical distance between the two lines.

Definition 3 *The aspect ratio of a canonical region R is*

$$\mathtt{ar}(R) = \max(\mathtt{diam}_i(R))/\min(\mathtt{diam}_j(R)), \forall i,j \in \{x,y,z\}.$$

Given an aspect ratio parameter α, a region R is α-balanced if $\mathtt{ar}(R) \leq \alpha$.

This definition is valid only for canonical regions. Since all of the regions that appear in this section are canonical regions, whenever we refer to any region we mean a canonical region. When the term α is understood, we refer to α-balanced regions as simply *balanced* regions and refer to non-α-balanced regions as *unbalanced* regions. Throughout the paper, we also call balanced and unbalanced regions, respectively, *fat* and *skinny* regions.

To understand the various notions of a canonical region, let us look at one specific canonical region R in Figure 4. Here we see the various sides of R, x^l, x^r, y^l, y^r, z^l, z^r. In particular, although not actually a true side of R, we still represent the side z^r. It is tangent to R and has zero length. From the figure, we see the various lengths of each side:

$$\begin{array}{lll} |x^l| = 2, & |y^l| = 5, & |z^l| = 1, \\ |x^r| = 3, & |y^r| = 4, & |z^r| = 0. \end{array}$$

Since we are using the L_∞ metric, the length of z^l is 1 rather than $\sqrt{2}$ as would be the case in the L_2 metric. We can also compute $\mathtt{diam}_i(R)$ for each of the three canonical directions as well as the aspect ratio of R.

- $\mathtt{diam}_x(R) = 5$,
- $\mathtt{diam}_y(R) = 3$,
- $\mathtt{diam}_z(R) = (2+5)/2 = 3.5$,
- $\mathtt{ar}(R) = \max(\mathtt{diam}_i(R))/\min(\mathtt{diam}_j(R)) = \mathtt{diam}_x(R)/\mathtt{diam}_y(R) = 2$.

3.1 Constructing the BAR tree

We now introduce the BAR tree data structure. Suppose we are given a point set $\mathcal{S}$ in the plane, $|\mathcal{S}| = n$, and an initially square region R containing $\mathcal{S}$. We construct a BAR tree T on $\mathcal{S}$ recursively dividing R into cells such that the following properties are guaranteed:

- Every cell in the tree is convex.
- Every cell in the tree has balanced aspect ratio.
- Every leaf cell contains at most a constant number of points of S.
- The tree has $O(n)$ nodes.
- The depth of the tree is $O(\log n)$.

The structure is straightforward and reminiscent of the original k-d tree. Recall that in a k-d tree, every node u in the tree represents a cell region u.`region` and an axis-parallel cut u.`cut` partitioning that region into two subregions, u.`left` and u.`right`. The leaves of the tree are cells with a constant number of points. In general, each cut divides the region into two roughly equal halves, and thus the tree has $O(\log n)$ depth and uses $O(n)$ space. However, if the vast majority of the points is concentrated close to any particular corner of the region, no constant number of axis-parallel cuts can effectively reduce the size of the point set and maintain good aspect ratio. This is a serious concern for many applications and for ours in particular. As a result, an extensive amount of research has been dedicated to improving and analyzing the performance of k-d trees and its derivatives, often concentrating on trying to maintain some form of balanced aspect ratio [5, 19, 29].

We now show how to construct a BAR tree T from a point set $\mathcal{S}$ using an aspect ratio parameter α and a balance parameter β. We prove that any α-balanced region can be divided by a sequence of one or two cuts into at most three subregions. We also guarantee that each subregion is α-balanced and the number of points in each of the three subregions is less than β times the number of points in the original region. We begin by defining the notions of a one-cut and a two-cut.

Definition 4 *Let R be an α-balanced canonical region containing n points. Let β be a given balance parameter. A one-cut is any canonical cut dividing R into two subregions R_1 and R_2 such that:*

1. *R_1 and R_2 are both α-balanced canonical regions.*
2. *R_1 and R_2 contain at most βn points.*

If there exists a one-cut for R, we say R is one-cuttable.

Definition 5 *Let R be an α-balanced canonical region containing n points. Let β be a given balance parameter. A two-cut is any canonical cut dividing R into two subregions R_1 and R_2 such that:*

```
create_BAR_tree($R, \alpha, \beta$)
    create node $u$
    $u$.region $\leftarrow R$
    if number of points in $R \leq c$,
        return $u$
    if an $(\alpha, \beta)$-balanced one-cut $s$, exists in $R$
        $u$.cut $\leftarrow s$
        $(R_1, R_2) \leftarrow s(R)$
    else let $s$ be an $(\alpha, \beta)$-balanced two-cut in $R$
        $u$.cut $\leftarrow s$
        $(R_1, R_2) \leftarrow s(R)$
    $u$.left $\leftarrow$ create_BAR_tree($R_1, \alpha, \beta$)
    $u$.right $\leftarrow$ create_BAR_tree($R_2, \alpha, \beta$)
    return $u$
```

Figure 5: Creating the BAR tree. The recursion stops when a cell has a constant number of points, $c \geq 1$.

1. *R_1 and R_2 are both α-balanced canonical regions.*
2. *R_2 contains at most βn points.*
3. *R_1 is one-cuttable.*

If there exists a two-cut for R, we say R is two-cuttable.

For an α-balanced region R which is two-cuttable, let s represent the two-cut dividing R into two regions R_1 and R_2, and let s' represent the one-cut dividing R_1. In other words, the sequence of two cuts, s and s', results in three α-balanced regions each containing at most βn points. To make it clear that α and β are parameters, we often refer to one-cuts (resp. two-cuts) of a region R as (α, β)-balanced one-cuts (resp. two-cuts).

Figure 5 shows the pseudo-code for the construction of a BAR tree. Here we use the notation $(R_1, R_2) \leftarrow s(R)$ as a shorthand for cutting the region R with a cut s resulting in subregions R_1 and R_2. We prove in the next section that every α-balanced region is either one-cuttable or two-cuttable for sufficiently large constant values of α and β. Since the algorithm only uses one-cuts and two-cuts, the regions produced are all α-balanced regions. The algorithm stops the recursion when a leaf cell has a constant number of points from S. Because at least every other cut used is a one-cut, the depth of the tree is $O(log_{1/\beta}n)$ and the size is $O(n)$. Therefore, the algorithm correctly creates a tree which satisfies the properties for a BAR tree.

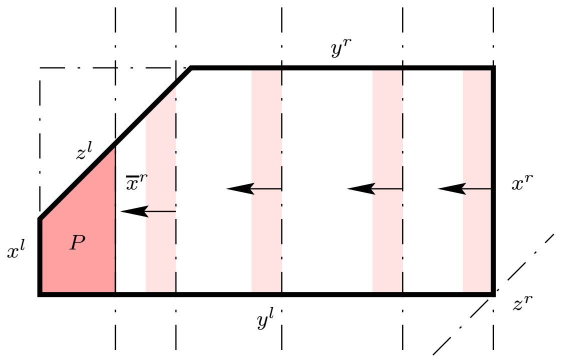

Figure 6: The shaded region P represents the region between x^l and a maximal cut of x^r for a region R.

3.2 Two-cut existence theorem

Since the correctness of the previous algorithm relies on the existence of a two-cut for a region, we prove that every region R is either one-cuttable or two-cuttable. Before we do this, we need to describe some basic terminology relating to cutting a region R into two subregions.

Definition 6 *Suppose we are given an α-balanced canonical region R and a canonical direction $\vec{v}_i$. Let i^l and i^r be the two (possibly zero length) sides of R normal to $\vec{v}_i$. Let $\overline{i}^l$ be the line containing i^l and let P be the region between i^r and $\overline{i}^l$ (at first P is the same as R). Sweep $\overline{i}^l$ towards i^r until either P is empty or just before P becomes unbalanced. We call this final region $R_{i,r} = P$ maximized in the direction from i^l. Similarly, we call $\overline{i}^l$ the maximal cut of i^l. $R_{i,l}$ is similarly defined.*

Definition 7 *For a region R with n points and a canonical direction $\vec{v}_i$, let $R_{i,l}$ (resp. $R_{i,r}$) represent the region maximized in the direction from i^r (resp. i^l), If $R_{i,l} \cap R_{i,r} = \emptyset$ define R_i to be the region $R_{i,l}$ or $R_{i,r}$ with the larger number of points. Otherwise if $R_{i,l} \cap R_{i,r} \neq \emptyset$, define R_i to be R.*

Since the change in aspect ratio during the sweep is continuous, the region $R_{i,r}$ has aspect ratio equal to α. Figure 6 illustrates a maximal cut of x^r for a canonical region R using the parameter $\alpha = 2$. The region $R_{i,r}$ maximized in the direction from x^r has aspect ratio $\texttt{ar}(R_{i,r}) = 2$. Figure 7 shows a few more examples of regions with their respective maximal cuts and associated subregions. The following lemma follows from a straightforward geometric argument.

Lemma 1 *Given regions R and $R_{i,r}$ and lines i^l and $\overline{i}^l$ as defined above, if $R_{i,r}$ is not empty and we continue sweeping in the same direction, the region between $\overline{i}^l$ and i^r will be unbalanced until it becomes empty.*

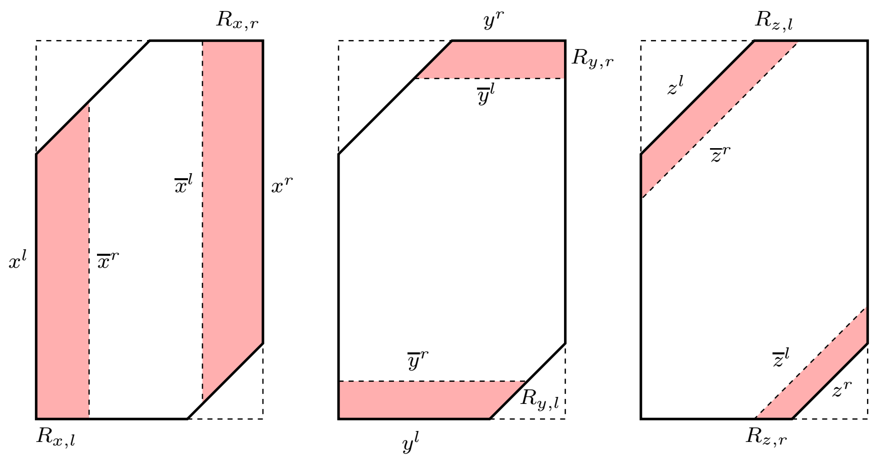

Figure 7: The labels on the sides of a general canonical region and the maximizing cuts from the respective directions.

Corollary 1 *For an α-balanced region R, if the region $R_{i,r}$ is maximized in the direction from i^l, then* $\min\{\texttt{diam}_x(R_{i,r}), \texttt{diam}_y(R_{i,r}), \texttt{diam}_z(R_{i,r})\} = \texttt{diam}_i(R_{i,r})$.

Corollary 2 *For an α-balanced region R and direction $\vec{v}_i$, if $R_{i,l} \cap R_{i,r} = \emptyset$, then any cut i^m with a normal $\vec{v}_i$ and lying between $\overline{i}^l$ and $\overline{i}^r$ produces two α-balanced subregions R_1 and R_2.*

Lemma 2 *Suppose we are given a region R with n points, a balance parameter $\beta \geq 1/2$ and two parallel lines c^l and c^r. Without loss of generality, let us orient these lines so that c^l lies to the left of c^r. Then one of the following must be true:*

- *The number of points from R to the left of c^l (i.e., away from c^r) is more than βn;*
- *The number of points from R to the right of c^r (i.e., away from c^l) is more than βn;*
- *There exists a line c' parallel and between c^l and c^r dividing R into two subregions R_1 and R_2 such that the number of points in either subregion is less than βn.*

Proof: Assume the first two conditions do not hold. Thus, we only need to prove that the last condition must hold. Let n_1 be the number of points to the left of c^l and let n_2 be the number of points to the left of c^r. We know then that $n_1 > \beta n \geq n/2$. Similarly, we know that $(n - n_2) > \beta n \geq n/2$. It follows then that $n_2 < n/2$. Sweep a line c' from c^l to c^r letting n_3 be the number of points to the left of c'. Since the sweep is continuous, n_3 varies from $n_1 > n/2$ to $n_2 < n/2$. In particular, there is a point where $n_3 = n/2$. This cut divides R into two subregions each with less than $n/2$ points. □

Corollary 3 *For an α-balanced region R with n points, a direction $\vec{v}_i$, and $\beta \geq 1/2$, either R is one-cuttable or R_i contains more than βn points.*

Proof: If the two subregions $R_{i,l}$ and $R_{i,r}$ intersect each other, then by definition $R_i = R$ and thus contains n points. If R is one-cuttable, then the statement is trivially true. Otherwise, we have two cuts $\vec{i}^r$ and $\vec{i}^l$ associated with $R_{i,l}$ and $R_{i,r}$ respectively. From Lemma 2, either $R_{i,l}$ or $R_{i,r}$ contains more than βn points or there exists a line c' parallel and between $\vec{i}^r$ and $\vec{i}^l$ dividing R into two subregions R_1 and R_2 such that the number of points in either subregion is less than βn. However, this implies that R is one-cuttable. □

The above corollary is quite useful in proving that certain regions are one-cuttable. For instance, let R be an α-balanced region such that, for some canonical direction $\vec{v}_i$, both $R_{i,l}$ and $R_{i,r}$ are empty. Since neither of these two subregions can contain any points, R must be one-cuttable. In fact, this notion can be extended to include multiple canonical directions.

Lemma 3 *Let R be an α-balanced region R with n points and $\beta \geq 2/3$. If $R_x \cap R_y \cap R_z = \emptyset$, then R is one-cuttable.*

Proof: This is a standard extension from set theory. For a set of points S, it is impossible to have three subsets of S each contain more than $2/3$ of S without their intersection containing at least one point. □

If we can prove that there exist regions such that no possible assignment for the R_i's allows for a non-empty intersection, then the region R is always one-cuttable. Do there exist regions which are guaranteed to be one-cuttable? We describe two such regions which we will use to argue that every α-balanced region is inevitably two-cuttable.

Definition 8 *For a given aspect ratio parameter α we define two special canonical regions with aspect ratio α as follows:*

- *Canonical isosceles trapezoidal (CIT) regions are trapezoids which have z^l and z^r as the two opposing parallel base sides, see Figure 8a.*
- *Canonical right-angle trapezoidal (CRT) regions are trapezoids which have their two opposing parallel base sides normal to either $\vec{v}_x$ or $\vec{v}_y$, see Figure 8b.*

Lemma 4 *For $\alpha > 4$ and $\beta \geq 2/3$, canonical isosceles trapezoidal (CIT) regions are one-cuttable.*

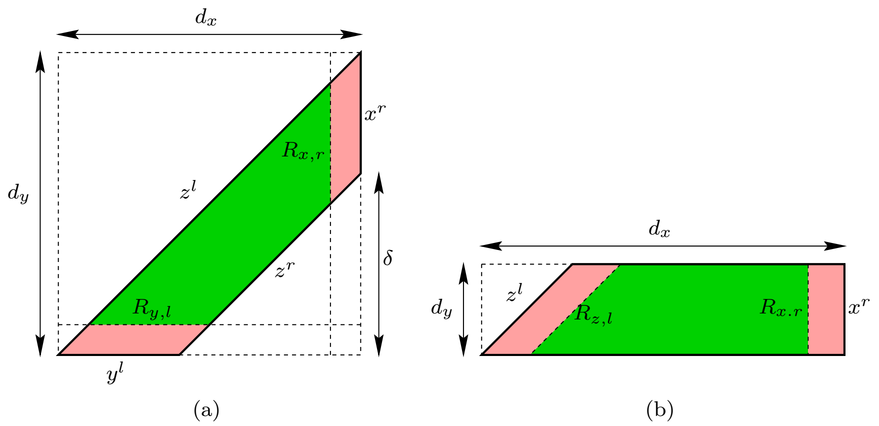

Figure 8: Examples of (a) CIT and (b) CRT regions.

Proof: Without loss of generality, we can analyze the region R in Figure 8a, since the other possible CIT regions are symmetrical. Let $d_i = \texttt{diam}_i(R)$ for $i \in \{x, y, z\}$. Define $\delta = |z^r| = d_x - |x^r|$. Since the trapezoid's two parallel sides are z^l and z^r, we know that $d_x = d_y$ and $|x^r| = |y^l|$. Recall that in the L_∞ metric, $d_z = (|x^l| + |y^l|)/2 = |y^l|/2$. Similarly, we get $d_z = |x^r|/2$. Since the region has aspect ratio α, we have $\texttt{ar}(R) = \alpha = d_x/d_z$. It follows that

$$\begin{aligned} d_x &= \alpha d_z \\ &= \alpha|x^r|/2 \\ &= \alpha(d_x - \delta)/2 \\ &= \alpha\delta/(\alpha - 2) \qquad (1) \end{aligned}$$

Let us examine the possible intersections of $R_x \cap R_y \cap R_z$. Since $R_{x,l}$ is empty, we know that $R_x = R_{x,r}$. Since by definition, $R_{x,r}$ is maximized from x^l, we know that $\texttt{diam}_x(R_x) \leq d_y/\alpha = d_x/\alpha$. From Equation 1 and from $\alpha > 4$, it follows that $\texttt{diam}_x(R_x) < \delta/2$. Similarly, we know that $R_y = R_{y,l}$ and $\texttt{diam}_y(R_y) < \delta/2$. This implies that $R_x \cap R_y = \emptyset$. From Lemma 3, R must be one-cuttable. □

Lemma 5 *For $\alpha > 4$ and $\beta \geq 1/2$, canonical right-angle trapezoidal (CRT) regions are one-cuttable.*

Proof: Without loss of generality, we can again analyze the region R in Figure 8b, since the other possible CRT regions are symmetrical. Let $d_i = \texttt{diam}_i(R)$ for $i \in \{x, y, z\}$. We know that $\max_{i\in\{x,y,z\}}(d_i) = d_x$ and $\min_{i\in\{x,y,z\}}(d_i) = d_y$ from the definition of the region. Therefore, we know that $\texttt{ar}(R) = \alpha = d_x/d_y$. Observing that $|y^r| = d_x - d_y$, we obtain:

$$d_y = d_x - |y^r|$$

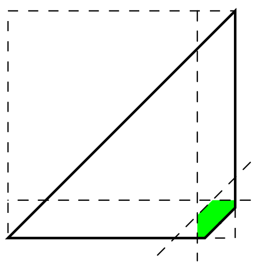

Figure 9: A region R which is not one-cuttable if the points are densely concentrated in the highlighted corner. Notice that no canonical cut can divide this region without creating a region that is too skinny.

$$= \alpha d_y - |y^r|$$
$$= |y^r|/(\alpha - 1) \qquad (2)$$

Let us examine the possible intersections of $R_x \cap R_y \cap R_z$. Since $R_{x,l}$ is empty, we know that $R_x = R_{x,r}$. Since by definition, $R_{x,r}$ is maximized from x^l, we know that $\mathtt{diam}_x(R_x) \leq d_y/\alpha$. From Equation 2 and from $\alpha > 4$, it follows that $\mathtt{diam}_x(R_x) < |y^r|/12$. Similarly, we can see that $R_z = R_{z,l}$ and $\mathtt{diam}_z(R_z) < |y^r|/6$. This implies that $R_x \cap R_z = \emptyset$. From Lemma 3 it follows that R must be one-cuttable. □

It is easy to construct examples where a region R is not one-cuttable for a given a point set, see Figure 9. However, the following theorem shows that by making a two-cut followed by a one-cut we can in fact divide an α-balanced region into at most three α-balanced subregions each containing less than a constant fraction of the points in R.

Theorem 1 (Two-Cut Existence Theorem) *Any α-balanced region R is either one-cuttable or two-cuttable for $\alpha \geq 6$ and $\beta \geq 2/3$.*

Proof: We can assume that R is not one-cuttable, and thus only prove that it must be two-cuttable. Again let $d_i = \mathtt{diam}_i(R)$ for $i \in \{x, y, z\}$. Without loss of generality, assume $d_y \geq d_x$. Consider the two parallel sides, z^l and z^r. We call a cut, z^i, $i \in l, r$, **small** if

$$|z^i| \leq \min(d_x, d_y)\frac{\alpha - 2}{\alpha} = d_x\frac{\alpha - 2}{\alpha},$$

and **large** otherwise. We now break the analysis into three cases based on the size of these two sides. Each case follows roughly the same argument. If a region is not one-cuttable, the three subregions R_x, R_y, and R_z must all intersect each other since $\beta \geq 2/3$. If one of these regions is one-cuttable, in particular either a CIT or CRT region, then R is two-cuttable. Therefore, we prove in each case that if all three subregions are not CIT or CRT regions, they cannot simultaneously intersect.

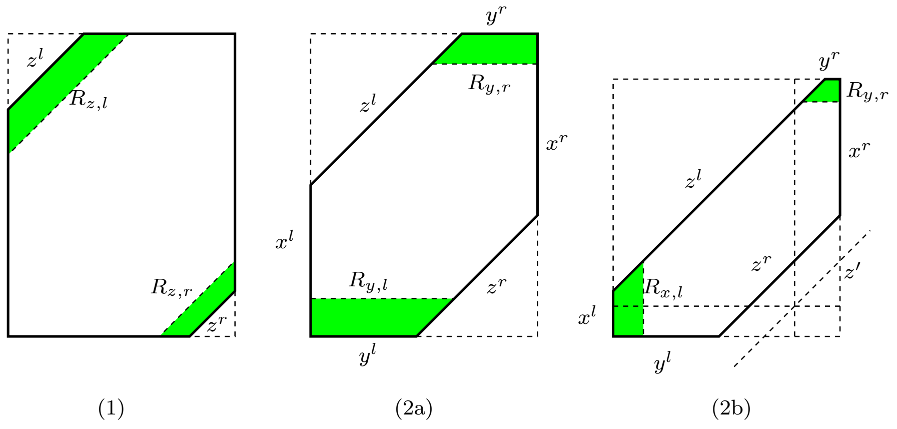

Figure 10: Case 1: both z^l, z^r are *small*. Case 2a: both sides are *large* and $|y^l| \leq |x^l|$, which guarantees that $R_{y,l}$ and $R_{y,r}$ are both CRT regions. Case 2b: both sides are *large* and $|y^l| > |x^l|$.

Case 1. (z^l and z^r are both small):
Let both z^l and z^r be small, see Figure 10.1. From Equation (1) and because z^l is small, we know that $\mathtt{diam}_x(R_{z,l}) = \alpha|z^l|/(\alpha - 2) \leq d_x$. The same holds for the region $\mathtt{diam}_x(R_{z,r})$. Thus these two CIT regions are disjoint. Since there was no one-cut, particularly in the z-direction, one of the two regions has more than βn points. By Lemma 4, both CIT regions are one-cuttable. Therefore, R has a two-cut, namely the one creating the CIT region with maximum points, R_z.

Case 2. (z^l and z^r are both large):
Let both z^l and z^r be large. Without loss of generality, let the larger of the two cuts be z^l. Notice that,

$$d_x(\alpha - 2)/\alpha < |z^r| \leq |z^l| \leq d_x.$$

Because $|z^l| \geq |z^r|$ and $d_x \leq d_y$, we know that $|y^r| \leq |x^r|$. Therefore, $R_{y,r}$ is a CRT region, and is one-cuttable.

If $|y^l| \leq |x^l|$, then $R_{y,l}$ is also a CRT region, see Figure 10.2a. From Lemma 5, R_y is always one-cuttable. Therefore, R is two-cuttable, the two-cut being either $\overline{y}^l$ or $\overline{y}^r$.

Otherwise, we have the situation in Figure 10.2b:

$$\begin{aligned} |x^l| &< |y^l| \\ &= d_x - |z^r| \\ &\leq d_x - d_x(\alpha - 2)/\alpha \\ &= d_x(1 - (\alpha - 2)/\alpha) \\ &= 2d_x/\alpha. \end{aligned} \quad (3)$$

We now have bounds on $|x^l|$, $|y^l|$, and $|y^r|$. Let us now bound $|x^r|$. Using Equation 3, we see that

$$\begin{aligned} d_y &\leq d_x + |x^l| \\ &\leq d_x + 2d_x/\alpha \\ &\leq d_x(1+2/\alpha). \end{aligned}$$

$$\begin{aligned} |x^r| &= d_y - |z^r| \\ &\leq d_x(1+2/\alpha) - d_x(1-2/\alpha) \\ &= 4d_x/\alpha \qquad (4) \end{aligned}$$

Using arguments similar to those used in proving Equation 2, we know that

$$\begin{aligned} \mathtt{diam}_x(R_{x,r}) &\leq |x^r|/(\alpha-1) \\ &\leq 4d_x/\alpha(\alpha-1), \text{ and} \end{aligned}$$

$$\begin{aligned} \mathtt{diam}_y(R_{y,l}) &\leq |y^l|/(\alpha-1) \\ &\leq 2d_x/\alpha(\alpha-1). \end{aligned}$$

Consider the intersection of $\overline{y}^r$ and $\overline{x}^l$ and the cut z' which passes through this point, see Figure 10.2b. If z' lies inside R, we can bound the size of the intersection of this cut with R by

$$\begin{aligned} |z'| &= (\mathtt{diam}_x(R_{x,r}) + \mathtt{diam}_y(R_{y,l})) \\ &\leq 6d_x/\alpha(\alpha-1) \\ &\leq d_x/5 \\ &< |z^r|. \end{aligned}$$

However, this implies that z' does not intersect R. Consequently, $R_{x,r} \cap R_{y,l} = \emptyset$, and either $R_x = R_{x,l}$ or $R_y = R_{y,r}$. Since either of these subregions is one-cuttable, R is two-cuttable.

Case 3. (only one of the two cuts is large):
Without loss of generality, let the larger of the two cuts be z^l. In other words, $|z^l| > d_x(\alpha-2)/\alpha$. Here we need to consider two subcases.

- **3i.** (long rectangle) If $d_y \geq d_x\frac{\alpha+1}{\alpha-2}$, we cannot necessarily cut the region using the direction $\vec{v}_x$. Using the same argument as in Case 2, we see that $R_{y,r}$ is a CRT region. Thus, if $R_y = R_{y,r}$, we are done. Similarly, using the argument for Case 1, we see that $R_{z,r}$ is a CIT region, see Figure 11a. Therefore, we can assume that $R_y = R_{y,l}$ and $R_z = R_{z,l}$ as in Figure 11b.

  From Equation 1, $\mathtt{diam}_y(R_{z,l}) \leq \alpha d_x/(\alpha-2)$. Similarly, from Equation 2, we know that $\mathtt{diam}_y(R_{y,l}) \leq d_x/\alpha$. Thus, combining the two yields,

$$\mathtt{diam}_y(R_{z,l}) + \mathtt{diam}_y(R_{y,l}) \leq d_x\frac{\alpha}{\alpha-2} + d_x/\alpha$$

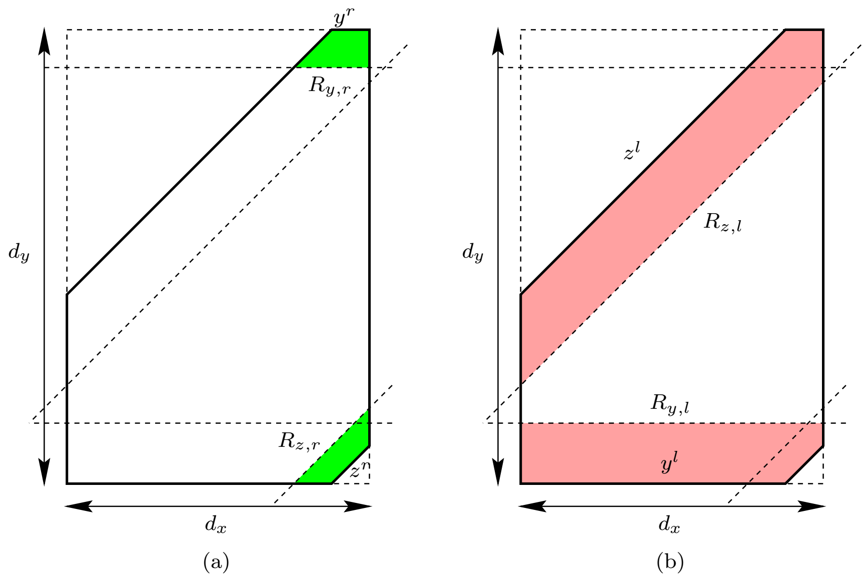

Figure 11: Case 3i, for a long rectangle. (a) Two one-cuttable subregions, $R_{y,r}$ and $R_{z,r}$. (b) Opposing not necessarily one-cuttable subregions, $R_{y,l}$ and $R_{z,l}$, but they cannot intersect.

$$\begin{aligned} &= d_x\left(\frac{\alpha}{\alpha-2}+\frac{1}{\alpha}\right) \\ &\le d_y\frac{\alpha-2}{\alpha+1}\left(\frac{\alpha}{\alpha-2}+\frac{1}{\alpha}\right) \\ &= d_y\frac{1}{\alpha+1}\left(\alpha+1-\frac{2}{\alpha}\right) \\ &< d_y. \end{aligned}$$

From this, we know that $R_{z,l}$ and $R_{y,l}$ cannot intersect. Therefore, either $R_z = R_{z,r}$ or $R_y = R_{y,r}$ and the region is two-cuttable.

- **3ii.** (squat rectangles) Now, we have $d_y < d_x\frac{\alpha+1}{\alpha-2}$. Since z^l is large, we know that $R_{y,r}$ is a CRT region. Since the rectangle is squat, we know that $R_{x,l}$ is also a CRT region, see Figure 12a. Since z^r is small, either $R_{z,l}$ is a CIT region or $R_{z,l} = R$. The latter case arises if maximizing from z^r and z^l produces regions which intersect each other. Notice, because of the dimensions of the region, this is not possible in either the $\vec{v}_x$ or $\vec{v}_y$ direction. Since $d_y \ge d_x$, $R_{y,l}$ cannot intersect $\cap R_{y,r}$. Notice also that, for $\alpha > 5$,

$$\begin{aligned} \mathtt{diam}_x(R_{x,l}) &\le d_y/\alpha \\ &< d_x\frac{\alpha+1}{\alpha(\alpha-2)} \end{aligned}$$

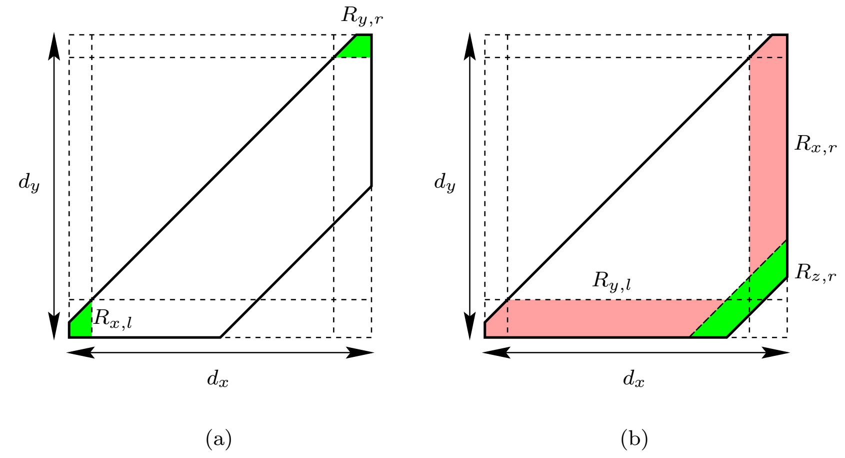

Figure 12: Case 3ii, for a short rectangle. (a) Two one-cuttable subregions, $R_{x,l}$ and $R_{y,r}$. (b) Opposing not necessarily one-cuttable subregions, $R_{x,r}$ and $R_{y,l}$. If they intersect, $R_z = R_{z,r}$ is a one-cuttable region.

$$< \quad d_x/2.$$

The same is true for $R_{x,r}$. So, $R_{x,l}$ cannot intersect $R_{x,r}$.

We only need to consider the case when $R_x = R_{x,r}$ and $R_y = R_{y,l}$. Since both regions contain more than βn points, they must intersect, see Figure 12b. It follows then that $|z^r| \leq 2d_x/\alpha$. We also know that $|z^l| \leq d_x$. Recalling that $\alpha \geq 6$, we can bound $\texttt{diam}_z(R)$, $\texttt{diam}_z(R_{z,r})$, and $\texttt{diam}_z(R_{z,l})$ by

$$\begin{aligned}
\texttt{diam}_z(R) &\geq d_x/2 - |z^r|/2 \\
&\geq d_x/2 - d_x/\alpha \\
&\geq d_x/2 - d_x/6 \\
&= \frac{d_x}{3} \\
\texttt{diam}_z(R_{z,l}) &\leq \frac{d_x}{\alpha} \\
&\leq \frac{d_x}{6} \\
\texttt{diam}_z(R_{z,r}) &\leq \frac{|z^r|}{\alpha - 2} \\
&\leq \frac{2d_x}{\alpha^2 - 2\alpha} \\
&\leq \frac{2d_x}{24}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{d_x}{12} \\
\mathtt{diam}_z(R_{z,r}) + \mathtt{diam}_z(R_{z,r}) &\le \frac{d_x}{6} + \frac{d_x}{12} \\
&= \frac{d_x}{4} \\
&< \frac{d_x}{3} \\
&\le \mathtt{diam}_z(R).
\end{aligned}
$$

This implies that $R_{z,l}$ does not intersect $R_{z,r}$ and similarly cannot intersect $R_{x,r} \cap R_{y,l}$. Therefore, we know that $R_z = R_{z,r}$. Since $R_{z,r}$ is a one-cuttable CIT region, we know that R must be two-cuttable.

This completes the proof of the two-cut existence theorem. □

Theorem 2 *Given a point set $\mathcal{S}$ in the plane, we can construct a BAR tree representing a decomposition of the plane into "fat" regions in $O(n \log n)$ time.*

Proof: To prove this, it suffices to note that a one-cut or a two-cut in any of the three canonical directions can be found in $O(n)$ time and that the depth of the tree is $O(\log n)$. □

4 Using a BAR tree for Cluster Based Drawing

Let $G = (V, E)$ be the graph that we want to draw. Once we obtain the embedding of G, using whatever algorithm is most appropriate for the graph, we associate with the graph the smallest bounding square, R, which we call G's *cluster region*. Using the embedding and its cluster region, we create the BAR tree T, as described above. Each node $u \in T$ maintains $u.\mathtt{region}$, $u.\mathtt{cluster}$, and $u.\mathtt{depth}$. Here $u.\mathtt{cluster}$ is the subgraph of G which is properly contained in $u.\mathtt{region}$. Recall that the depth of the tree T is $k = O(\log n)$. In our application of the tree structure to cluster-based graph drawing, we want every leaf to be at the same depth. Therefore, we propagate any leaf not at the maximum depth down the tree until the desired depth is reached. This is merely conceptual and does not require any additional storage space or change in the tree structure.

Using the tree T, we create the clustered graph C, which consists of k layers. Each layer is an embedded subgraph of G along with the regions and clusters obtained from T. The layers are connected with vertical edges which are simply the edges in T. The other inputs to LGD are the aspect ratio parameter α and the balance parameter, β. Here, α determines the maximal aspect ratio of a cluster region in C, and β determines the cluster balance, the ratio of a cluster's size to its parent's. For a summary of the operations, see Figure 13.

Lemma 6 *A call to $\mathtt{LGD}(G, \alpha, \beta)$ for $\alpha = 6$, $\beta = 2/3$ results in $2/3$-balanced clustering with aspect ratio less than or equal to 6 and cluster depth $O(\log n)$.*

```
LGD($G, \alpha, \beta$)
    embed($G$)
    $T \leftarrow$ create_BAR_tree($G, \alpha, \beta$)
    $C \leftarrow$ create_clustered_graph($T, G$)
    display($C$)
```

Figure 13: Main algorithm. The inputs to the algorithm are graph G along with the aspect ratio parameter α and the balance parameter β. Graph G is embedded in the plane, after which the BAR tree T is created. Finally, the clustered graph C is created and displayed.

Proof: By construction, the clusters are β-balanced and the cluster depth is equivalent to the depth of T. Thus, for $\alpha \geq 6$ and $\beta \geq 2/3$ the depth is $O(\log_{1/\beta} n)$. □

Theorem 3 *For $\alpha \geq 6$, $\beta \geq 2/3$, algorithm* LGD *creates a 2/3-balanced clustered graph C in $O(n \log n + m + D_0(G))$ time.*

Proof: The proof follows directly from the construction of the algorithm and previous statements about the running time of each component. □

Once we obtain the clustered graph C, we can display it as a 3-dimensional multi-layer graph representing each cluster by either the the convex hull of its vertices or by its associated region in the BAR tree. Along with the clustered graph C we can display a particular cluster with more details. Thus we provide the global structure using the clustered graph and the local detail using the individual clusters.

4.1 Planar Graphs

When the graph G is planar, we are able to show a few special properties of our clustered drawings.

Theorem 4 *If G is planar, for $\alpha \geq 6$, $\beta \geq 2/3$, algorithm* LGD *creates a 2/3-balanced clustered graph C in $O(n \log n)$ time. Moreover, C is embedded with straight lines and no crossings on the $n \times n \times k$ grid, where $k = O(\log n)$.*

Proof: We begin with a planar grid embedding with straight-line edges [6, 12, 28] and then the original layer, G_k, is planar. Since each successive layer is a proper subgraph of the previous layer, it too must be planar and drawn without edge crossings. □

In Figure 14 we can see a clustered graph $C = (G, T)$ in which the clusters are represented by the partitions of the plane obtained from the BAR tree. Note that in this case there is no need to select a representative vertex for a cluster.

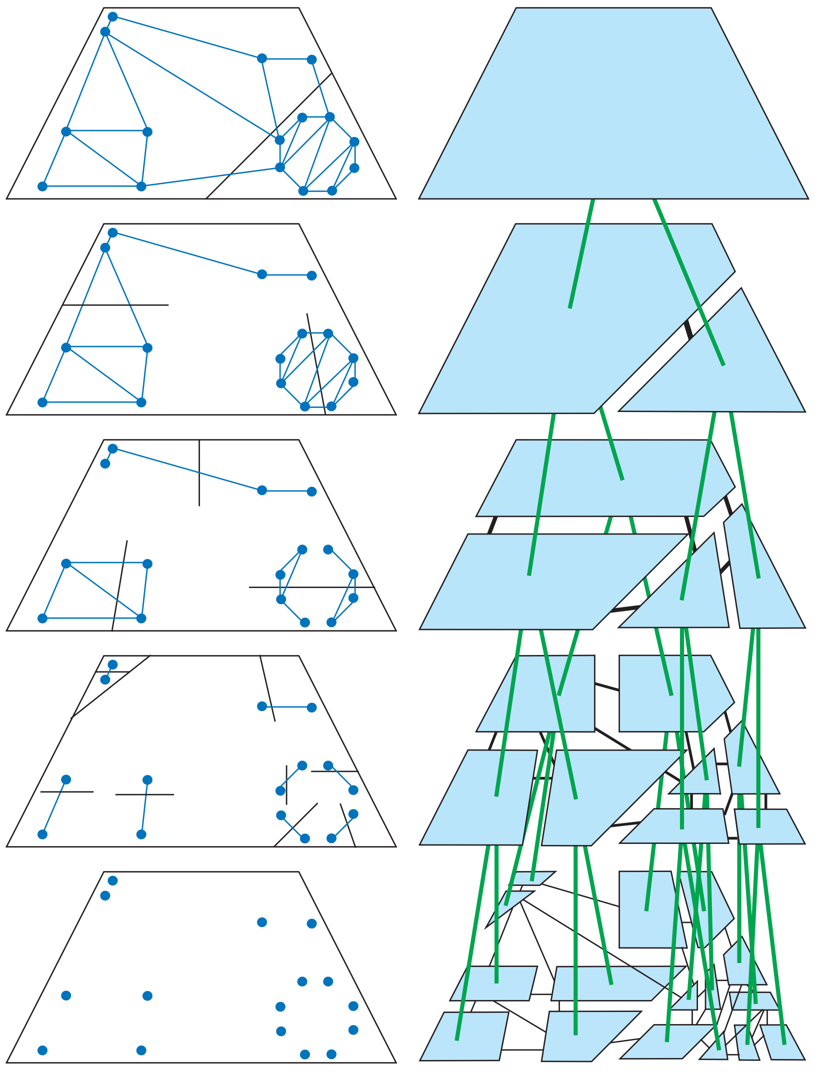

Figure 14: A clustered graph $C = (G, T)$. The clustering of G on the right is obtained from the BAR tree cuts on the left. Each cluster is represented by the region defined by the BAR tree cuts. Note the edge-region crossings at the last two levels.

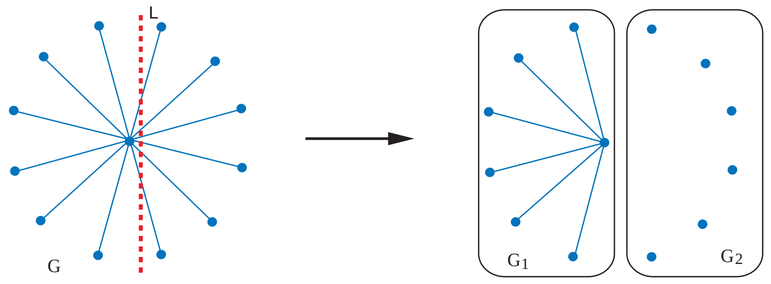

Figure 15: Graph G with an inherently large cut. Any cut L which maintains a β-balance between the clusters, where $1/2 \leq \beta < 1$, cuts $O(n)$ edges.

For such drawings it is possible to have an edge cross a region that it does not belong to. Moreover, it is possible to have an edge cross the convex hull of a cluster that it does not belong to. If we represent a cluster by the convex hulls of its connected components, however, there will be no such crossings. Thus, if we could guarantee that each cluster is connected or has a small number of connected components, the display of the graph can be improved even further. Alternatively, we can redefine the clusters at each level to be the connected components of vertices inside each cluster region of the BAR tree. With this definition of clusters we could then use the algorithm of Eades and Feng [10] to produce a new clustered embedding of the planar graph so as to have no edge or region crossings.

4.2 Extensions

Throughout this paper we do not discuss the cut sizes produced by our algorithm, that is the number of edges intersected by a cut line in the BAR tree. In some applications it is important that the number of such edges cut be as small as possible. There exist graphs, however, that do not allow for "nice" cuts of small size. Consider the *star* graph G on Figure 15. Any cut, which maintains a β-balance between the two subgraphs it produces, intersects $O(n)$ edges. If the balance parameter is $\beta = 1/2$, the cut contains $\lfloor \frac{n}{2} \rfloor$ edges. As this example shows, we cannot hope to guarantee cut sizes better than $O(n)$. Still, if the given graph has a small cut then we would like to find a small cut as well.

Minimizing the cut size violates two of our five criteria, namely, speed and convexity. First of all, looking for the best β-balanced cut is a computationally expensive operation, and while it can be done in polynomial time, it is not hard to see that it cannot be done in linear time. In addition, the best β-balanced cut may not preserve the convex cluster drawing property that `LGD` maintains. As shown in Figure 16, this may result in new edge crossings in our clustered graph.

Our algorithm does not guarantee that it will find the optimum β-balanced

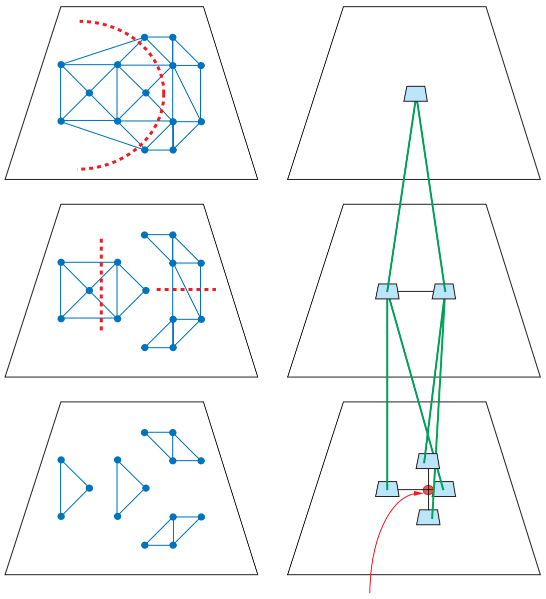

Figure 16: An example of a graph in which each cluster is represented by a single node. Note that the non-straight line cut produces a crossing in the multi-level graph.

cut but we can modify the BAR tree construction so that we find locally optimal cuts. Here are some of the possible criteria that we can use in choosing among the potential cuts: minimize cut size, minimize connected components resulting from a given cut, minimize aspect ratio, maximize β-balance.

These criteria can also be combined in various ways to produce desired optimization functions. In finding such optimal cuts, it is important to note that a one-cut, if available, might not always be a better choice over a potential two-cut. Yet again, a two-cut that minimizes the cut size may have no subsequent one-cut that does not cut many more edges. Thus, it may be reasonable to go two levels in evaluating possible scores instead of choosing greedily.

5 Conclusion and Open Problems

In this paper we present a straightforward and efficient algorithm for displaying large graphs. The `LGD` algorithm optimizes cluster balance, cluster depth, aspect ratio and convexity. Our algorithm does not rely on any specific graph properties, although various properties can aid in performance, and produces the clustered graph in a very efficient $O(n \log n + m + D_0(G))$ time.

The embedding of the cluster graph is determined in the very first step of our algorithm. Unfortunately, it is possible that the initial embedding is not the best one (for example, in terms of the size of the cuts produced by our

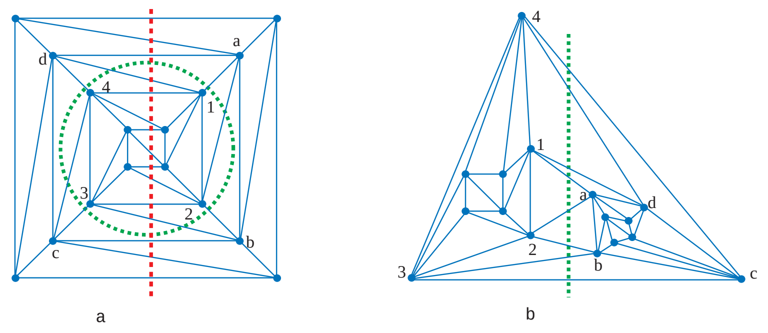

Figure 17: The graph in part (a) has no β-balanced line cut of size better than $O(n)$ but it does have a cycle cut (the dotted circle) of size $O(1)$. We can transform the graph in (a) to the graph in (b) by taking one of the faces crossed by the cycle as the outer face. Note that in (b) the cycle cut has become a line and its size is $O(1)$.

algorithm). In fact, as shown on Figure 17, G may have a minimum β-balanced cut of size $O(n)$ or $O(1)$, depending on the embedding. While it is still true that some graphs may always have cuts of size $O(n)$ (for example, the star graph, Figure 15), we would like to minimize the cut whenever we can. It is an open question whether it is possible to determine the optimal embedding, one that yields the minimum β-balanced cuts.

Another open question is related to the separator theorems of Lipton and Tarjan [21] and Miller [22]. Is it possible given a 2-connected planar graph G to always produce $O(\sqrt{dn})$ β-balanced cuts, where d is its maximum degree, and n is the number of vertices? If so, can we find an embedding for the resulting clustered graph which preserves efficiency, cluster balance, cluster depth, convexity, and guarantees good aspect ratio and straight-line drawings without crossings?

Acknowledgements

We would like to thank Rao Kosaraju and David Mount for their helpful comments regarding the balanced aspect ratio tree.

References

[1] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[2] S. Arya and D. M. Mount. Approximate range searching. In *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, pages 172–181, 1995.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[4] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.

[5] S. P. Dandamudi and P. G. Sorenson. An empirical performance comparison of some variations of the k-d tree and BD tree. *International Journal of Computer and Information Sciences*, 14(3):135–159, June 1985.

[6] H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fary embeddings of planar graphs. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 426–433, 1988.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.

[8] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ, 1999.

[9] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees: Combining the advantages of *k-d* trees and octrees. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 300–309, 1999.

[10] P. Eades and Q.-W. Feng. Multilevel visualization of clustered graphs. In *Proceedings of the 4th Symposium on Graph Drawing (GD '96)*, pages 101–112, 1996.

[11] P. Eades, Q.-W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Proceedings of the 4th Symposium on Graph Drawing (GD '96)*, pages 113–128, 1996.

[12] I. Fáry. On straight lines representation of planar graphs. *Acta Scientiarum Mathematicarum*, 11:229–233, 1948.

[13] Q.-W. Feng, R. F. Cohen, and P. Eades. How to draw a planar clustered graph. In *Proceedings of the 1st Annual International Conference on Computing and Combinatorics (COCOON '95)*, pages 21–31, 1995.

[14] Q.-W. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In *Third Annual European Symposium on Algorithms (ESA '95)*, pages 213–226, 1995.

[15] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[16] G. W. Furnas. Generalized fisheye views. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23, 1986.

[17] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[18] K. Kaugars, J. Reinfelds, and A. Brazma. A simple algorithm for drawing large graphs on small screens. In *Graph Drawing (GD '94)*, pages 278–281, 1995.

[19] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(2):23–29, Apr. 1978.

[20] R. J. Lipton, S. C. North, and J. S. Sandberg. A method for drawing graphs. In *Proceedings of the 1st Annual ACM Symposium on Computational Geometry*, pages 153–160, 1985.

[21] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9:615–627, 1980.

[22] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986.

[23] F. J. Newbery. Edge concentration: A method for clustering directed graphs. In *Proceedings of the 2nd International Workshop on Software Configuration Management*, pages 76–85, 1989.

[24] S. C. North. Drawing ranked digraphs with recursive clusters. In *Graph Drawing, ALCOM International Workshop PARIS on Graph Drawing and Topological Graph Algorithms (GD '93)*, Sept. 1993.

[25] R. Sablowski and A. Frick. Automatic graph clustering. *Proc. of 4th Symposium on Graph Drawing (GD '96)*, LNCS 1190:395–400, 1996.

[26] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[27] M. Sarkar and M. H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.

[28] W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 138–148, 1990.

[29] Y. V. Silva-Filho. Average case analysis of region search in balanced k-d trees. *Information Processing Letters*, 8(5):219–223, June 1979.

[30] S. K. Stein. Convex maps. *Proceedings of the American Mathematical Society*, 2(3):464–466, 1951.

[31] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compou nd digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.

[32] W. T. Tutte. How to draw a graph. *Proceedings London Mathematical Society*, 13(52):743–768, 1963.

[33] K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.