ALIASE: Application of Least Inertia Axis for Skeleton Extraction in Handwritten Character Images.

Abdul-Rahman O. Ibraheem

Computing and Intelligent Systems Research Group, Dept. of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Osun State, Nigeria.

Abstract
For the case of character images, we present a first response to the question of whether the axis of least inertia can lead to a symmetry-seeking skeleton representation for shapes. We developed an algorithm that iteratively partitions an image into sub-regions until the content of each sub-region is sufficiently (as determined by two thresholds) approximated by its axis of least inertia. Experiments showed the algorithm produces useful skeletons comprised of line strokes, when appropriate thresholds are hand-chosen. Specifically, experiments suggested thresholds increase with increasing character width. Therefore, towards automatic thresholding, we propose a future study on modeling the relationship between thresholds and widths. However, an advantage of our approach is that it unifies skeletonization and stroke extraction; unlike stroke extraction via standard routes, our approach is free of errors from preliminary thinning. For completeness, we studied the run time of our algorithm rigorously. Although rigor shows an $O(N^2)$ run time for our algorithm, experiments revealed a sort of desirable "linear performance."

1 Introduction

Imagine a sub-region of the image of a line drawing, such as the English letter "F." If the "line" within a sub-region of the letter is reasonably straight, then it should be possible to approximate it by its axis of least inertia (ALI). Based on this intuition, we developed an algorithm, which we refer to as ALIASE (Application of Least Inertia Axis for Skeleton Extraction), and which iteratively partitions the image of a line drawing until the content of each sub-partition is reasonably straight, and can thus be approximated by its ALI. The expected effect is that the algorithm converts a binary image into a set of line segments which approximates the shape of the image. For an early feel, we show a sample output in Fig. 1, and reserve others for the section on "Experiments."

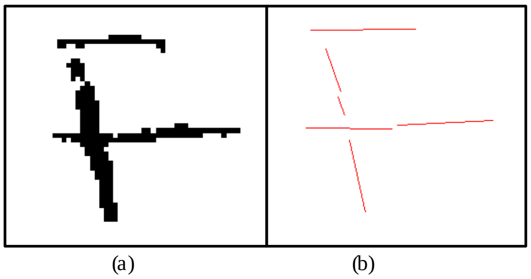


Fig. 1. Letter "F" being Converted to a Set of Strokes.
(a) A binary image; (b) the strokes produced by our algorithm for the binary image in (a)..

Looking at Fig. 1, it is obvious that the "F" formed by the output line segments resemble the original binary image very closely. However, the line segments disconnected. Traditionally, connectivity is one of the important criteria for good skeletonization/thinning techniques. For graphics applications, we concur that connectivity is a lofty aesthetic goal worth pursuing assiduously. However, for recognition applications, which is our concern herein, we opine that connectivity needs not show up directly in the skeleton's image; after all, the skeleton will still be segmented and converted into a graph, for recognition purposes. Thus, what matters most is whether the underlying connectivity information can be successfully transferred to the final graph recognition structure. As it turns out, it can be transferred. Specifically, in the final graph representation, the nodes corresponding to any two output line segments should be linked by an edge if and only if the binary sub-image corresponding to the first node contains a foreground pixel that is a neighbor to a foreground pixel in the binary sub-image corresponding to the second node. On the flip side though, we must highlight a strength of our approach: it performs thinning and stroke extraction in a single module, unlike traditional approaches (e.g. [1], [2]) that first thin the image, and then subsequently extract strokes from the thinned image, using say the Ramer – Douglas-Peucker algorithm [3]. This means that our approach is free from errors committed at the preliminary thinning stage.

2 Related work

Our work herein falls under the heading of structural shape representation, and, in particular, skeleton representations. To begin with, we point out that the skeleton representation paradigm leads to useful applications; see the works in [4], [5], [6], and [7] for examples. A key goal of skeletonization methods has been to implement the medial axis transform (MAT) of Blum [8]. The MAT of an image is the locus of the centers of bi-tangential circles inscribable in the image [9]. As in [10] and [11], the distance transform can be used to implement the MAT. To obtain the distance transform, one first computes a distance map, which contains the distance of each point in the image from the nearest boundary point. One then locates the desired skeleton along the local extreme points of the distance transform. Recently, Latecki et. al. [12] have used the distance map to obtain shape skeleton in a different way. They first compute a "pseudo-representation" of the distance transform field. They then diffuse the gradient of the result isotropically, and compute a skeleton strength map from this. Finally, they obtain the skeleton as the local maxima of the strength map.

Further, a significant contribution, the well known shock graph, is due to Steve W. Zucker and collaborators [13]. The shock graph approach tracks singularities that occur as the closed boundary of a shape evolves under two types of motion: a constant hyperbolic conservation-law motion, acting through the shape's interior, and a local curvature dependent parabolic conservation law motion that tends to smooth the boundary anisotropically. More recently, Macrini et. al. [14] have improved on the shock graph, via what they call bone graphs. In particular, bone graphs are aimed at remedying the problem of over segmentation, which sometimes plagues shock graphs. What might informally pass for a "discrete version" of shock graphs is in the approach of [15], who computed skeletons by exploiting the concept that the critical points on the boundary of a shape correspond to skeletal end-points, under discrete curve evolution. They use particle filters to link critical points up, thus forming the required skeleton. Further in the earlier cited work of Choi et. al. [10], based on skeleton width considerations, a set of criteria are derived which a skeleton point's 3×3 neighbourhood must satisfy. A pixel is dismissed as not being a skeleton point if no pixel in its 3×3 neighbourhood satisfy the criteria. One problem with skeleton representations has to do with spurious branches. In very recent work, Shen et. al. [4] contributed towards solving this problem. They proposed the bending potential ratio as a measure of skeleton branch significance. The bending ratio purportedly captures both local and global curvature information of shape boundary segment, and so is expected to be a reliable measure of skeleton branch significance.

Probably beginning with [16], a body of research exists which connects Voronoi diagrams, as well as Delaunay triangulations with skeletonization. In this arena, we should mention the work of [17] who followed the Voronoi approach: an approach which computes skeletons as a subset of the Voronoi medial axis, which in turn is the intersection of the shape and the Voronoi polygons obtained using a subset of shape boundary points as generating points. A main contribution of [17] was the introduction, as a measure of skeleton branch significance, of the anchor distance, which, for an edge of the Voronoi medial axis, is the shortest path between corresponding generating points along the shape's boundary. Later, Morrison and Zou [18] used Delaunay triangulations, in place of Voronoi diagrams, for obtaining skeletons. As one of their key contributions, they proposed a novel technique for choosing boundary "generating points." Very recently, Liu et. al. [19] described the generalized Voronoi skeleton (GVS). A major difference between the GVS and "traditional Voronoi skeletons" is that it uses curve segments as generating points rather than single pixels. Further, they proposed shape area associated with (in the "shape reconstruction sense") skeleton branch divided by total shape area as one of two measures of skeleton branch significance.

Yet, a class of skeletonization methods iteratively peel boundary pixels from the shape, subject to shape-remnant connectivity, until hopefully a one-pixel width skeleton results. A comprehensive survey of this class of methods is in [20]. In this context, a representative approach is that of Rutovitz [21]. He used crossing numbers, defined over a 3×3 neighbourhood of pixels as the number of white-to-black or black-to-white transitions, to decide whether a pixel be deleted or not. Later, Hilditch [22] introduced a definition of crossing numbers as approximately the number of white-to black transitions encountered in a counter-clockwise walk around the 3×3 pixel neighbourhood. A more recent work within this category is that of Ahmed and Ward [23]. They proposed a rule-based algorithm, employing twenty rules, defined over a 3×3 pixel neighbourhood, to determine whether pixels can be deleted or not. For two-pixel wide regions, they examine 5×5 neighbourhoods. Rockett [24] improved upon the above algorithm, designing a two-stage strategy, whose first stage uses the algorithm of [23] to obtain an intermediate result, which is an ideal skeleton, except that it may contain two-pixel wide lines. The second stage of the strategy constructs an undirected graph over 3×3 pixel neighbourhoods; the centre pixel is deleted if it does not break connectivity of the graph.

Our approach differs from any of the above referenced techniques. In typical structural character recognition applications, the character is first thinned using any of the above thinning methods. Strokes are then extracted from the thinned character, for example via the Ramer-Douglas-Pecucker algorithm. However, in our approach, thinning and stroke extraction are combined into a single stage, so that our approach is free from errors committed during preliminary thinning.

3 Illustrating ALIASE

Although we depict the quintessence of ALIASE in the tree shown in Fig 2, we first give a general outline of its workings as follows. ALIASE begins action by first using connected component labeling to determine the number of objects in the image space. All objects found are pushed unto a stack. Then, we pop out the object at the top of the stack. The ALI of the object is computed, along with the mean absolute deviation of all its pixels from the just computed axis. If the mean deviation falls below a pre-defined threshold, τ, then we are done with that object; we need not divide it any further. We simply output its properties: its centroid, ALI, and bounding box, which are then used to draw an output line segment. However, if this mean deviation is greater than τ, then the image is partitioned into two (possibly unequal) halves, either vertically or horizontally. We then apply connected component labeling to find the objects in each half, pushing all found objects onto the stack. At this point, a while loop checks if the stack is empty or not. If it is, the procedure terminates. If it is not, the next object on top of the stack is popped out, and the previously explained procedures are applied to it.

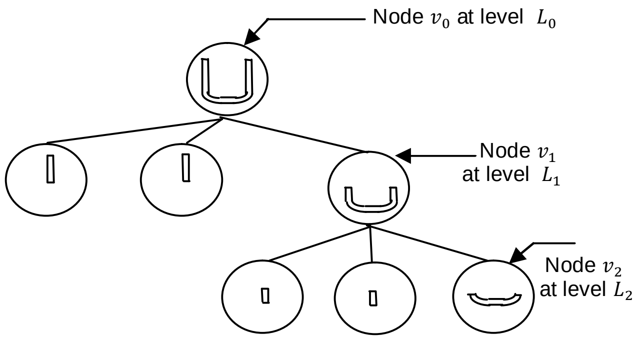

Node v_0 at level L_0
Node v_1 at level L_1
Node v_2 at level L_2

Fig. 2. An Example ALIASE Partition Tree

In Fig. 2, we show how the letter "U" might be processed by ALIASE. The "U" is first divided horizontally into an upper region containing two elements; and a lower region containing one element. The two elements in the upper region are both sufficiently straight. So, they appear as leaf nodes in the tree. The single element in the lower region requires further division. It is divided horizontally into an upper region and a lower region. This time, all three elements produced by the splitting require no further division, and so appear as leaf nodes.

We call trees of the type in Fig. 2 ALIASE Partition Trees (APTs). Our formal definition of an APT is a tree "generated" when our algorithm is invoked with an $m \times n$ ($m \geq 2$ and $n \geq 2$) binary image, I, and a $\tau \geq 0$. We say that the tree in Fig. 2 has tree levels: L_0, L_1 and L_2, with the height of a tree equal to the maximum index associated with any level in it. We take the length of a path through a tree to be the number of edges in the path. Lastly, we refer to an object contained in the leaf node of an APT as a primitive.

We see from the above discussion that ALIASE comprises five main building blocks: computation of axes of least inertia, which subsumes computation of centroids; computation of mean deviations; image space partitioning; connected component labeling; and computation of bounding boxes.

4 Describing the components of ALIASE

This section is devoted to a detailed discussion of the first three main building blocks of ALIASE. As for the fourth, we use the method in [25] ; the fifth is a well known technique [26].

.

4.1 Axis of Least Inertia (ALI)

The ALI of a shape is a line which gives the direction along which the shape is most elongated [26]. It is a line which passes through the object's centroid, and minimizes the sum of the squares of perpendicular distances from itself and each point in the object [26]. If it is denoted by α, then it can be shown that:

$$2\alpha = \tan^{-1}(2\mu_{rc}/(\mu_{rr} - \mu_{cc})) \quad (1)$$

where μ_{cc} μ_{rr} and μ_{rc} are respectively the second order column moments, the second order row moments, and the second order mixed moments of the object about its centroid, $(\bar{r}, \bar{c})$. In symbols, for an object R, we have:

$$\mu_{cc} = (1/A) \sum_R (c - \bar{c})(c - \bar{c}) \quad (2)$$

$$\mu_{rr} = (1/A) \sum_R (r - \bar{r})(r - \bar{r}) \quad (3)$$

$$\mu_{rc} = (1/A) \sum_R (c - \bar{c})(c - \bar{c}) \quad (4)$$

4.2. Mean Deviation

Our algorithm uses mean absolute deviation to distinguish between "reasonably straight" objects and not reasonably straight objects. We denote the mean absolute deviation of an object by $\bar{d}$, and define it according to:

$$\bar{d} = (1/n) \sum_{i=1}^{n} d_i \quad (5)$$

In the object in question, d_i represents the absolute deviation of the i-th pixel from the ALI; and n is the number of pixels in that object. To be most specific, for each pixel, p_i, d_i is measured from the center of p_i. It should be easy to see that, in the worst case, for an $m \times n$ image, I, with $N = m \times n$, Equation (5) requires $O(N)$ time.

We draw attention to a special type of primitives, which we call primitive cells. A primitive cell is an object which exactly fills either a $1 \times n$ image space, or a $m \times 1$ image space. A special property of primitive cells is that they have mean absolute deviation of zero.

4.3. Image Space Partitioning.

Our image space partitioning algorithm is motivated by one basic consideration, illustrated in Fig. 3. In the figure, the thick line shows what we consider a "line of good partitioning," while the dashed line represents what we consider a "line of bad partitioning." From the figure, we see that each time the line representing a good partitioning enters an object, R, through some foreground pixel having row (or column) co-ordinates a, it quickly exits it via another foreground pixel having row (or column) co-ordinates b, with the quantity $|b - a|$ being "small." On the other hand, for lines of bad partitioning, there is at least one instance wherein the line enters the object via a foreground pixel having row (or column) co-ordinate a' and exits via another pixel with row (or column) co-ordinate b', with the quantity $|b' - a'|$ being "large." Physically, good partitioning tends to preserve the line segments being sought by ALIASE, while bad partitioning tends to destroy them. Via experimentation, it is possible to arrive at a threshold, p_t, such that whenever $|b - a| \leq p_t$, the partitioning in question is considered good; else, it is deemed bad.

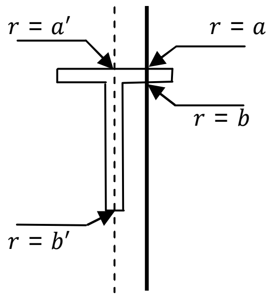


Fig. 3. Good versus Bad Lines of Partitioning. The thick line is a line of good partitioning, but the thin line is a line of bad partitioning.

The aim is divide the image (or sub-image), I, into two sub-images via the line of good partitioning closest either to the middle row or the middle column of I. Without loss of generality, for the case wherein the number of columns, n, is greater than or equal to the number of rows, m, of I, then the order in which our algorithm searches for a line of good partitioning is specified by the ordered set, S_L,:

$S_L = \{r = r_m, c = c_m, r = r_m + 1, c = c_m + 1, r = r_m - 1, c = c_m - 1, r = r_m + 2, c = c_m + 2, r = r_m - 2, c = c_m - 2, \cdots, r = 1, \cdots, c = 1\}$; where r_m and c_m respectively denote the (r, c) raster co-ordinates of the middle row and middle column of I. If no line of good partitioning is found in S_L, the image is arbitrarily divided along r_m.

5 Maximum height of partition trees and maximum run time of ALIASE

In this section, we conduct a formal analysis of the run time of ALIASE. We follow the analysis of merge sort algorithm in [27]. The idea is to sum the count of operations at each level of the tree "generated" by the algorithm (for our purpose, ALIASE partition trees, or APT) over the maximum height of the tree. Towards this goal, we first need to place a bound on the height of APT. We then count operations at each level of the tree, and then sum up this count over the height of the tree. We claim that, for any $m \times n$ image, I, (with $m \geq 2$ and $n \geq 2$) the height of any APT rooted at I cannot exceed $m + n - 3$. Towards proving this claim, we need some formalism:

Definition 1. Let T be a tree whose nodes are rectangular sub-images. We refer to T as a partition tree if:

1). The root node of T holds an $m \times n$ image with $m \geq 2$ and $n \geq 2$.

2). Each non-leaf node, v, in T has children $v_1, v_2, \cdots, v_l$ satisfying: $v_1 \cup v_2 \cup \cdots \cup v_l \subset v$ and $v_1 \cap v_2 \cap \cdots \cap v_l = \emptyset$.

3). In T, any node, v, holding an $r \times c$ sub-image cannot bear children, if either $r = 1$ or $c = 1$.

Let us observe the relationship between the definition of a partition tree, given above, and that of an APT, given earlier in Section 3. In particular, let us appreciate how the third condition above stipulates that primitive cells (a type of primitives) cannot be further partitioned. This is in perfect consonance with what happens within APTs, wherein primitives, which subsume primitive cells, cannot be further partitioned. The crux of our juxtaposition should be the realization that: all APTs are instances of partition trees, but not all partition trees are APTs. A strong benefit of this is that properties, such as upper or lower bounds, that apply to partition trees will also apply to APTs as well.

Definition 2. Suppose $P = v_0, v_1, \cdots, v_l$ is a path of length l in a partition tree, T. We call P a root path in T, if v_0 is the root node of T.

Definition 3. Given a path $P = v_0, v_1 \cdots v_l$ in a partition tree, T. We say that P is prudent if, for each node $v_i (i \geq 1)$ in P, the sub-image at v_i is smaller than that at v_i's parent node, v_{i-1}, by exactly one row or column.

We are ready to state the first lemma of this section:

Lemma 1. Let T be a partition tree rooted at an image I, and let l^* be the maximum possible length of any root path in T. If a given root path, P, in T has length l^*, then P is prudent

Proof We go by contradiction. Let $P = v_0, v_1, \cdots, v_{i-1}, v_i, \cdots, v_{l-1}, v_l$ be a root path of length l in some partition tree T. Suppose $l = l^*$, but that P is not prudent. For each node v_i ($i \geq 1$), in P, the sub-image at v_i must be smaller than that at v_i's parent node, v_{i-1}, by at least one row or column. Otherwise, no partitioning occurred at v_{i-1}; so, how did v_i come about ? Now, by the supposition that P is not prudent, it means there is at least one pair of consecutive nodes v_{i-1}, v_i in P such that v_i is smaller than v_{i-1} by either more than one row or more than one column. Without loss of generality, assume, for example, that v_i is smaller than v_{i-1} by $r + 1$ rows ($r \geq 1$). This allows for the possible existence of another path P' which can be built from P, by simply introducing extra r nodes between v_{i-1} and v_i, such that each of these extra nodes differs from its parent node by exactly one row. Labeling these r nodes as $v^1, v^2, \cdots, v^r$, the path between v_{i-1} and v_i becomes $v_{i-1}, v^1, v^2, \cdots, v^r, v_i$, so that P' can be written $P' = v_0, v_1, \cdots, v_{i-1}, v^1, v^2, \cdots, v^r, v_i, \cdots, v_{l-1}, v_l$. This makes it clear that the length of P' is $l + r = l^* + r$, which is greater than l^*; a contradiction of the supposition that l^* is the maximum possible length of any root path in T.

We proceed with:

Lemma 2. The maximum possible length, l^*, of any root path in any partition tree, T, rooted at an $m \times n$ image, I, is $m + n - 3$. In other words: $l^* \leq m + n - 3$.

Proof. Notice Lemma 2 requires us to show $l^* < m + n - 2$. Now, suppose $P = v_0, v_1, \cdots, v_l$ is a root path with length l^*. By Lemma 1, P is prudent: each edge in P corresponds to either a loss of exactly one row, or a loss of exactly one column. But these rows and columns being lost come from a finite store of m rows and n columns contained in the image, I, at the root node, v_0, of T. This allows us to write: $l^* \leq (m-1) + (n-1) = m + n - 2$; that is, we can lose at most $(m-1)$ rows and at most $(n-1)$ columns from I, because the sub-image at the last node, v_l, in P must, at least, subsume a pixel. It should be clear that, if we can show that $l^* \neq m + n - 2$, then we are done. Towards this, suppose, for a contradiction, that $l^* = (m-1) + (n-1)$, and then introduce a formalism: a sequence $\vec{R}$ of horizontal and vertical razors, such that an horizontal razor, denoted $\vec{h}$, removes a single row from an image, while a vertical razor, denoted $\vec{v}$, removes a single column from an image. In essence, each $\vec{h}$ corresponds to an edge in path P; likewise each $\vec{v}$. With this, it means to lose $m-1$ rows and $n-1$ columns from an $m \times n$ image, I, we need a set $\vec{R}$ containing $m-1$ $\vec{h}$'s as well as $n-1$ $\vec{v}$'s. Now, observe that a primitive cell will occur in P either once the $(m-1)$-th $\vec{h}$ is encountered in $\vec{R}$, or once the $(n-1)$-th $\vec{v}$ is encountered in $\vec{R}$. But according to the third condition in our formal definition of partition trees, P can have only one primitive-cell node, and this primitive-cell node will be the last node of P. Therefore, the razor which causes a primitive cell to occur must be in the last position of $\vec{R}$. So, if both the $(m-1)$-th $\vec{h}$ and the $(n-1)$-th $\vec{v}$ must co-exist in $\vec{R}$, they must both occupy the last position in $\vec{R}$, simultaneously: a sheer contradiction.

With the aid of Lemma 2, we arrive at a key result:

Theorem 1. The maximum possible height, h^*, of any partition tree, T, rooted at an $m \times n$ image, I, is $m + n - 3$. In other words: $h^* \leq m + n - 3$.

Proof. Consider a specific partition tree, T, rooted at an $m \times n$ image, I. Following the definition given near the end of Section 3, the height, h_T, of T is equal to the maximum index, i_T, associated with any level of T. Looking at Fig. 2 in that section, i_T is also the length, l_T, of the "longest" root path in T. We may thus write, $h_T = i_T = l_T$. Therefore, across all possible partition trees that can be generated on I, the maximum possible height, h^*, of any of such trees must equal the maximum possible length l^* of any of them. Consequently, by virtue of Lemma 2, we have: $h^* = l^* \leq m + n - 3$..

Remark 1. *Since APTs are examples of partition trees, Theorem* 1 *applies as well to APTs* .

We can now show that our algorithm runs in $O(N^2)$ time. As earlier mentioned, the idea is to sum the count of operations at each level of an APT over the maximum possible height of the APT. From Theorem 1, we already know that, for an m by n image, the maximum possible height of any partition tree generated by our algorithm is $m + n - 3$. Therefore, what remains is to track the operations at each level of our APTs. Towards this, we note that, if the image at the root of an APT contains N pixels, then no matter how the partitioning proceeds, there will be a total of N pixels at any level of the APT. Now, at any node of an APT, there is maximum of five kinds of major operations to be performed: computation of connected component labels; computation of bounding boxes; computation of axes of least inertia, which subsumes computation of centroids; computation of mean absolute deviations; and computation of lines of good partitioning. Each of these take a maximum of $O(P)$ time for a sub-image of P pixels. Since there is a total of exactly N pixels at any level of an APT, we conclude that a total time of $O(5N) = O(N)$ is spent at each level of the APT. Hence, the total amount of time across all levels is $O\big(N(m + n - 3)\big)$. Now, via induction, it is shown in the appendix that, for all $m \in \mathbb{Z}, n \in \mathbb{Z}$, such that $m \geq 2$ and $n \geq 2$, we have $m + n - 3 \leq mn = N$. Thus the total time taken by our method is $O\big(N(m + n - 3)\big) = O(N^2)$. We shall soon confirm this experimentally.

6 Experiments

We exhibit results of ALIASE for characters of the English alphabet; explore the variation of threshold values, τ and p_t with image scale, and hence character width; answer the question of which image form, and threshold values, cause ALIASE to run for the longest time; and plot worst-case running times of ALIASE.

6.1 Line Segments Computed by ALIASE

We tested our algorithm on 40 × 40 handwritten uppercase English letters. We show sample results in Figure 4. Except for the last row, all other outputs in the figure have been produced using thresholds τ = 1.6 and p_t = 6. The key thing to notice about the outputs (except those on the last row) is that ALIASE is capable of producing line segments which **resemble** the original image. As was pointed out in Section 1, the line segments output by ALIASE are disconnected, but this is not a source of worry in recognition applications, since connectivity information can still be extracted directly from the underlying binary image (as explained in Section 1). A key source of worry however is that a single pair of threshold values does not work for all 40 × 40 binary images. For example the threshold pair, τ = 1.6 and p_t = 6, does not work well for the 40 × 40 binary image of "Y" on the last row of Figure 4. Rather, we found that τ = 1.9 and p_t = 7 works well for it. Specifically, when we tried τ = 1.6 and p_t = 6 on that "Y", we got the output immediately to the right of the original binary image of the "Y." Notice that that output contains undesirable zigzag lines (due to **over segmentation**) in the upper right area. A careful scrutiny of the original binary image shows that its upper right area is markedly fatter than the other areas; observe the two bulges, one to the left and one to the right, in that upper right area. This suggests that the reason why the zigzag lines showed up in the upper right area of the output is that that area corresponds to a fat area in the original image. To remove the zigzags, we simply **increased** the threshold values from τ = 1.6 and p_t = 6 to τ = 1.9 and p_t = 7. This resulted in the second output on the last row of Figure 4. Notice especially that the zigzag lines have now disappeared. This hints at the hypothesis that threshold values increase with character width, a topic which we explore in greater details in the next sub-section.

6.2 Variation of τ and p_t with Image Size and Character Width

Suppose ALIASE works well with thresholds τ_1 and $p_{t,1}$ on an image, I_1. When I_1 is **enlarged** to yield I_2, there would be a need to **increase** the thresholds to new values τ_2 and $p_{t,2}$. The reason for increasing τ_1 is that any given primitive line in I_2 will be thicker, and consequently have higher mean absolute deviation, than its counterpart in I_1. Therefore in I_2, the mean absolute deviation corresponding to the primitive line may not fall below τ_1. Hence, in the image I_2, the "given primitive line" might be further partitioned by ALIASE, resulting in a situation wherein a primitive is unnecessarily further divided. This causes **over segmentation** and causes ALIASE to output zigzag lines. The rationale for the needed increment in $p_{t,1}$ is that, since the strokes in I_2 will be thicker than their counterparts in image I_1, it would be generally harder to find lines of good partitioning in I_2 using $p_{t,1}$.

Conversely, when I_1 is **reduced** in size to obtain a new image I_0, there would be a need to **decrease** the thresholds τ_1 and $p_{t,1}$ to new values τ_0 and $p_{t,0}$. The reason for the needed reduction in τ_1 is that any given primitive line in I_0 will be thinner, and thus have lower mean absolute deviation, than its counterpart in the original image I_1. As such, while primitives alone yielded values of mean absolute deviations falling below τ_1 in the image I_1, it would be possible for non-primitives, along with primitives, to yield values of mean absolute deviations falling below the value τ_1 in the case of image I_0. In I_0, this results in a situation in which non-primitives are taken as primitives by ALIASE. This can cause **obliteration**, a situation in which significant portions of the original image are not accounted for by the output line segments.

Fig. 5a shows a 40 × 40 binary image which is converted by ALIASE, using τ = 1.6 and p_t = 6, to the line segments in Fig. 5b. Next, we enlarged the binary image by a factor of two, producing thereby an 80 × 80 image. We then passed this 80 × 80 binary image to ALIASE, using τ = 1.6 and p_t = 6. We show results for this in Fig. 5c. Looking at the upper right region of the "Z" in Fig. 5c, we see quite a lot of zigzags. To address this issue, we raised the thresholds to τ = 2.2 and p_t = 7, producing the better line segments (free of zigzag lines) in Fig. 5d. Further, we scaled the original 40 × 40 binary image by half, to get a 20 × 20 image. We then passed the resulting 20 × 20 image to ALIASE using τ = 1.6 and p_t = 6. We show results of this trial in Fig. 5e. Looking closely at Fig. 5e, one sees that the line segment produced for the diagonal stroke of the "Z" is too short: the diagonal stroke suffers **obliteration**. To resolve this issue, we decreased the thresholds to τ = 0.6 and p_t = 5 . These new values produced the line segments depicted in Fig. 5f. It is obvious that, with these lower thresholds, more of the diagonal stroke of the "Z" is now being accounted for.

Clearly, the just described experiments suggest that appropriate threshold values increase (decrease) with increasing (decreasing) image size. Now, as image size increases (decreases), character width also increases (decreases). This sheds light on what we saw in the last row of Figure 4. The "Y" in that last row suffers over segmentation in an area that has large width, compared to the rest of the Y's body. This leads us to postulate that threshold values should depend on local widths. We therefore propose a future study aimed at modeling this dependence in a supervised learning framework.

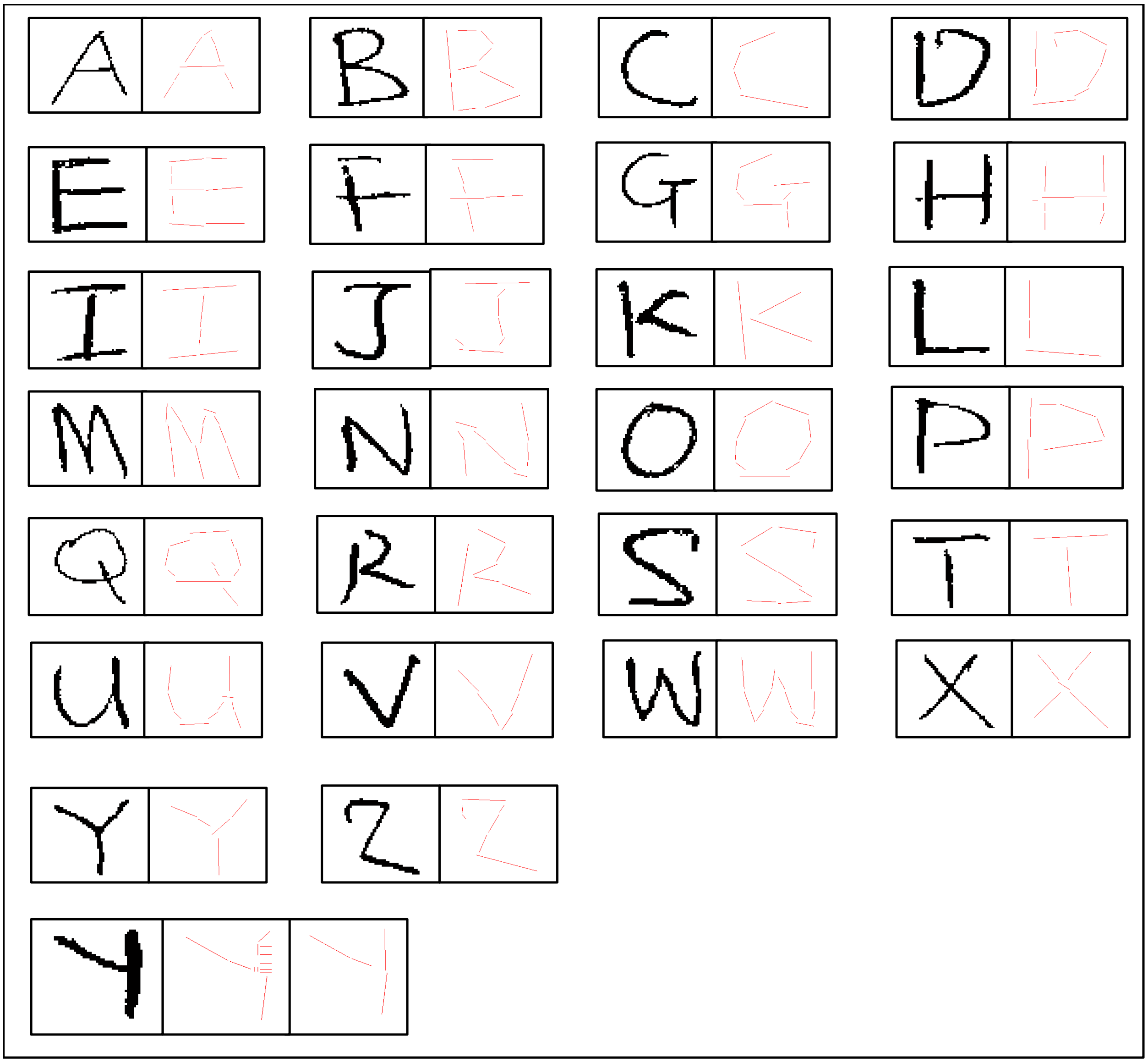


Fig. 4. Sample outputs produced by ALIASE. Except on the last row, all other outputs have been produced using threshold values $\tau = 1.6$ and $p_t = 6$. On the last row, the first output, which contains zigzags in the upper right area, was also produced using $\tau = 1.6$ and $p_t = 6$. The second output, which is devoid of zigzags, on the last row was produced with $\tau = 1.9$ and $p_t = 7$.

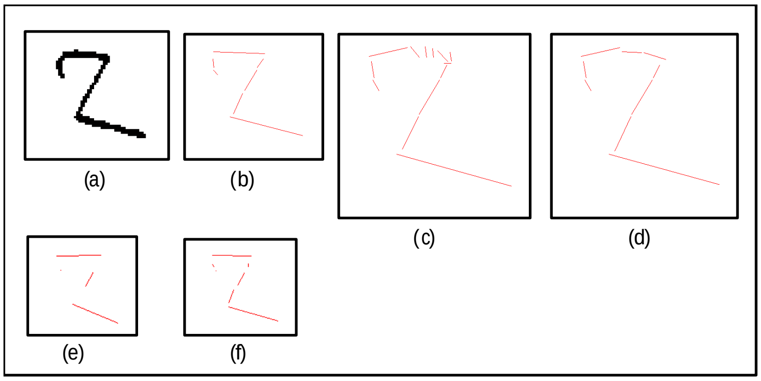


Fig. 5. Effects of image scale on appropriate threshold values.

6.3 *Choice of Image Form and Threshold Values for the Worst Case Running Times of ALIASE***.**

Given an m, which $m \times m$ binary image yields the worst case run time of our algorithm? We posit that it is the $m \times m$ binary image all of whose pixels are foreground pixels. This guess is informed by the idea that this "completely filled" image should be the "hardest" for our algorithm to break down into primitives. In addition, we need to answer the question of which values of τ and p_t give the worst case run time of our algorithm, on a completely filled binary image. The first choice is easy: τ = 0 corresponds to the worst case run time of ALIASE, because, for a given image, it is a choice that "maximizes" image partitioning. The choice of p_t is subtlier. We want a p_t^* that maximizes the height of the image's APT. We guess that p_t^* = 2 will, and support this claim experimentally (See Fig. 6).

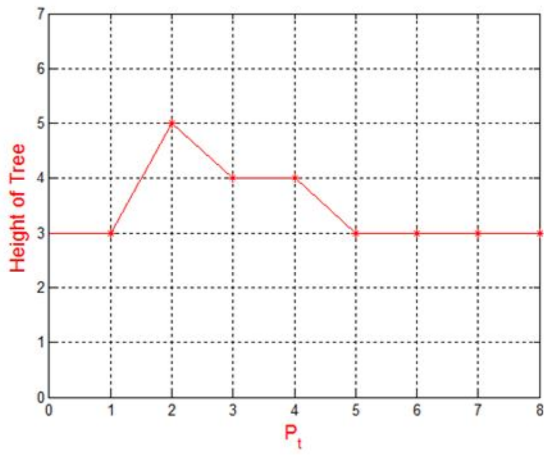
6.4 Worst Case Run Time of ALIASE.

Using τ = 0 and p_t = 2, we set up an experiment in which we pass completely filled $m \times m$ binary images to our algorithm, starting with m = 5 through m = 100, in steps of five. We set N = m^2, and plot a graph of run time versus N (the asterisked plot in Figure 7).

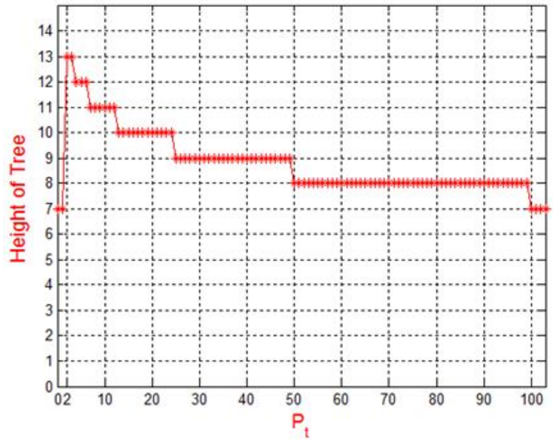
The non-asterisked quadratic graph in Figure 7 is a plot of $f(N)$ = $10^{-6}N^2$. A careful study of Figure 7 reveals that, for all $N \geq 30^2$ (i.e. the "upper" sixteen asterisks in the figure), we have $T(N) \leq f(N)$ = $10^{-6}N^2$. This is a perfect confirmation that the worst case running time of ALIASE is $O(N^2)$. The non-asterisked straight line in the figure is a plot of $g(N)$ = $10^{-3}N$. A scrutiny of the figure reveals that, for all $15^2 \leq N \leq 100^2$ (i.e. the "upper" eighteen asterisks in the figure), we have $T(N) \leq g(N)$ = $10^{-3}N$. This says that for $m \times m$ images with $15 \leq m \leq 100$, the worst case running time of ALIASE is bounded above by a straight line whose slope is as small as 10^{-3} seconds/pixel. This is a **very good** property, because many images we are interested in satisfy $15^2 \leq N \leq 100^2$, where N is number of pixels. To belabor the point, it means that ALIASE is exhibiting a sort of linear performance on the set of images we are most interested in practically.

7 Conclusion and future work

In this work, we presented an algorithm, called ALIASE, for converting binary images of line drawings into a skeleton of line segments. The algorithm works by iteratively partitioning an image until the object in each sub-partition can be sufficiently (as dictated by a pair of thresholds) approximated by the object's axis of least inertia. We captured the iterative partitioning process set up by ALIASE via our so-called APTs, and proved an upper bound on the height of the APTs. Although this allowed us to establish a formal quadratic bound on the algorithm, experiments indicated the algorithm exhibits a sort of desirable linear performance on the class of images that are of practical importance. An advantage of ALIASE is that, unlike traditional approaches that first thin the image



(a)



(b)

Fig. 6. Plots of APT heights versus p_t for: (a) a 5 × 5 Completely Filled Binary Image (CFBI); and (b), a 100 × 100 CFBI. Each plot peaks at p_t = 2.

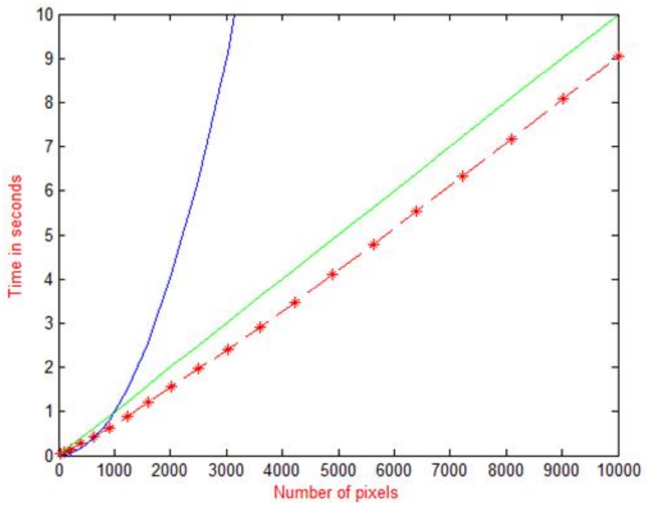


Fig. 7. Plot (asterisked) of worst case run times of ALIASE with (non-asterisked) quadratic and straight line graphs super-imposed.

before extracting strokes from it, the ALIASE approach unifies stroke extraction and thinning into a single module. This means ALIASE is free from errors committed during preliminary thinning. On the flip side, a present drawback of ALIASE is that it is tied to a pair of thresholds that must be hand-chosen for now. Fortunately though, experiments indicate a correlation between these thresholds and character width. We therefore propose a future study aimed at modeling this relationship.

References

[1] C. Liu, I. Kim, J.H. Kim, Model-based stroke extraction and matching for handwritten chinese character recognition, Pattern Recognition 34 (2001) 2339 – 2352.

[2] P.S. Wang and A. Gupta, An improved structural approach for automated recognition of handprinted characters, Int. J. Pattern Recognition and Artificial Intell. 5 (1991) 97–121.

[3] D.H. Douglas, T.K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Canadian Cartographer 10(2) (1973) 112-122.

[4] W. Shen, X. Bai, R. Hu, H. Wang, L. J. Latecki, Skeleton growing and pruning with bending potential ratio, Pattern Recognition 44 (2011) 196–209.

[5] D. Macrini, S. Dickinson, D. Fleet, K. Siddiqi, Object categorization using bone graphs, Comput. Vision and Image Understanding 115 (2011), 1187-1206.

[6] J. Xie, P. Heng, M. Shah, Shape matching and modeling using skeletal context, Pattern Recognition 41 (5) (2008) 1756–1767

[7] A. Farina, Z.M. Kovacs-Vajna, A. Leone, Fingerprint minutiae extraction from skeletonized binary images, Pattern Recognition 32 (1999) 877- 889.

[8] H. Blum, A transformation for extracting new descriptors of shape, in: Proceedings of the Models for the Perception of Speech and Visual Form, Cambridge, MA, 1967, pp. 362–380.

[9] H. Blum, Biological shape and visual science: Part I, J. Theoretical Biology 38 (1973) 205-287.

[10] W.P. Choi, K..M. Lam, W.C. Siu, Extraction of the Euclidean skeleton based on a connectivity criterion, Pattern Recognition 36 (2003) 721 – 729.

[11] Y. Ge, J. M. Fitzpatrick, On the generation of skeletons from discrete Euclidean distance maps, IEEE Trans. Pattern Anal. Mach. Intell. 18(11) (1996) 1055-1066.

[12] L.J. Latecki, Q. Li, X. Bai, W. Liu, Skeletonization using SSM of the distance transform, in: Proceedings of the IEEE Int. Conf. on Image Processing, 2007, pp. 349-352.

[13] B.B. Kimia, A.R. Tannenbaum, and S.W. Zucker, Shape, shocks and deformations I: the components of 2d shape and the reaction-diffusion space, Int. J. Comput. Vision 15(3) (1995) 189-224.

[14] D. Macrini, S. Dickinson, D. Fleet, K. Siddiqi, Bone graphs: medial shape parsing and abstraction, Comput. Vision and Image Understanding 115 (2011) 1044–1061.

[15] Y. Tang, X. Bai, X. Yang, L. Lin, S. Liu, L.J. Latecki, Skeletonization with particle filters, Int. J. Pattern Recognition and Artificial Intell. 24(4) (2010) 1- 16.

[16] U. Montanari, A method for obtaining skeletons using a quasi-Euclidean distance, J. Assoc. Comput. Machinery 15(4) (1968) 600–624.

[17] R.L. Ogniewicz, O. Kubler, Hierarchic Voronoi skeletons, Pattern Recognition 28 (3) (1995) 343–359.

[18] P. Morrison, J.J. Zou, Skeletonization based on error reduction, Pattern Recognition 39(6) (2006) 1099-1109.

[19] H. Liu, Z. Wu, D. F. Hsu, B.S. Peterson, D. Xu, On the generation of skeletons using generalized voronoi diagrams, Pattern Recognition Letters 33 (2012) 2113-2119.

[20] L. Lam, S. Lee, C.Y. Suen, Thinning methodologies: a comprehensive survey, IEEE Trans. Pattern Anal. Machine Intell. 14(9) (1992) 869-885.

[21] D. Rutovitz, Pattern recognition, J. Roy. Stat. Soc. Series A (129) (1966) 504-530.

[22] C.J. Hilditch, Linear skeletons from square Cupboards,.in: B. Meltzer, D. Michie (Eds.), Machine Intelligence. Elsevier, New York, 1969, vol. 4, pp. 403-420.

[23] M. Ahmed and R. Ward, A rotation invariant rule-based thinning algorithm for character recognition, IEEE Trans. Pattern Anal. Machine Intell. 24 (12) (2002) 1672-1678.

[24] P.I. Rockett, An improved rotation-invariant thinning algorithm, IEEE Trans. Pattern Anal. Mach. Intell. 24 (10) (2005) 1671- 1674.

[25] R.M. Haralick and L.G. Shapiro, Computer and Robot Vision, vol. I, Addison-Wesley, 1992.

[26] L. Shapiro, and G. Stockman, Computer Vision, Prentice Hall, New Jersey, 2001.

[27] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms. MIT Press, 2001.

Appendix: A statement and its proof.

Statement: *For all $m \in \mathbb{Z}, n \in \mathbb{Z}$, with $m \geq 2$ and $n \geq 2$, we have $m + n - 3 \leq mn$.*

Proof :
To begin with, it should be obvious that we are done, if we are able to show that: for all $m \in \mathbb{Z}, n \in \mathbb{Z}$, with $m \geq 2$ and $n \geq 2$, we have $m + n \leq mn$. We shall proceed by induction. We take our base case to be for $n = 2$, $m \geq 2$. This base case holds because the hypothesis that $m \geq 2$ implies $m + m \geq 2 + m \Rightarrow 2m \geq 2 + m \Rightarrow m + 2 \leq 2m$. For the induction step, we must show that, if $m + n \leq mn$, then $m + (n + 1) \leq m(n + 1)$. Now, $m + n \leq mn \Rightarrow m + n + 1 \leq mn + 1 \leq mn + m = m(n + 1)$, where the last inequality holds by the hypothesis that $m \geq 2$. This fulfills our goal.