Saarland University
Faculty of Mathematics and Computer Science

Bachelor's Thesis

Compressing Binary Images with the Medial Axis Transform

Marie Josefine Philippine Mühlhaus

Supervisor:	Prof. Dr. Joachim Weickert
Advisor:	Dr. Pascal Peter
Reviewers:	Prof. Dr. Joachim Weickert
	Dr. Pascal Peter
Submitted:	27th February 2020

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 27th February 2020

Marie Josefine Philippine Mühlhaus

Abstract

We develop a near-lossless binary image codec, called *MAT compression*, that exploits the sparse object representation given by skeletonization of the image. More explicitly, we use the medial axis transform (MAT) that enables a nearly perfect object reconstruction from its skeleton and the corresponding medial axis function (MAF) in a discrete setting. The codec consists of several preprocessing steps that shall increase the efficiency of the compression codec, and the compression step itself. To decrease the complexity of the skeleton structure, we apply a threshold-based pruning method to all skeleton branches. Afterwards, the remaining branches are smoothed and chain coded, yielding a compact object description. To thin the MAF, we select data points such that a piecewise linear approximation of the remaining points provides a reconstruction error which falls below a tunable tolerance threshold. The sparse skeleton and its thinned MAF are then transferred in an encoding scheme that is compressed with LPAQ—a strong compression standard developed by Matt Mahoney.

Finally, we experimentally evaluate the performance of MAT compression based on three criteria that characterize a good compression standard. Therefore, test images from the common MPEG7 CE Shape-1 database are used. The approach is compared to existing image compression standards such as JBIG, JPEG, JPEG2000, and modified Huffman encoding. This enables us to draw conclusions about the suitability and potential of the MAT for binary image compression.

Acknowledgements

First of all, I would like to express my gratitude to my advisor Dr. Pascal Peter who developed the idea for the topic of this thesis. I appreciate his advice and feedback and I am very thankful for the fast support when I encountered problems.

Furthermore, I owe thanks to Prof. Dr. Joachim Weickert who inspired my interest in the area of image processing and who offered a variety of exciting thesis topics to me.

My thanks also go to Ferdinand Jost for taking the time to discuss some ideas for my thesis.

Special thanks also go to Gregory and Simon for proofreading and giving precious LaTeXtips. In particular, I am very grateful for the thesis template Gregory provided me with.

Last but not least, I would like to thank my family and Lukas who gave me warm encouragement and support during my studies.

Contents

1 Introduction

The amount of data transmitted via the Internet increases rapidly. The reasons for this development are, for instance, the various possibilities to stream videos on mobile devices such as tablets, mobile phones, and TVs. These devices also enable us to take and send photos or videos easily at any time. However, since memory on hardware is limited, the need for efficient video and image compression methods has increased.

Although many of the stored images may be colored, binary images, i. e., images consisting of white and black pixels only, are still relevant today. Binary images are attractive for storage as only one bit per pixel is sufficient for representing its value whereas at least one byte is necessary for grayscale and color images. Especially I/O devices such as fax machines and scanners still offer the possibility to use binary images. This type of image is often encountered in the area of image processing as well. There are, for example, so-called binary masks used to describe regions of interest, e. g., for image segmentation or inpainting. Another application is a method called *dithering* that transforms a gray valued image into a binary one. Thereby, different gray values are imitated by arrangements of black and white pixels with different densities in order to reduce the color depth.

Large binary image databases are utilized, for instance, for the training of shape recognizing artificial neural networks or for performance evaluation of shape descriptors. One such widespread database is MPEG7 CE Shape-1 [Lon92] to which we also refer in this thesis. So-called skeletonization algorithms offer a natural sparse representation of those images. These algorithms have been used since the late 1960s, i. e., first by Blum [Blu67], in order to transform images to a more compact version resembling a skeletal structure. The resulting skeletons can, for instance, be helpful for fingerprint or character recognition. If we store additional information making it possible to reconstruct the original object from the skeleton, they are even

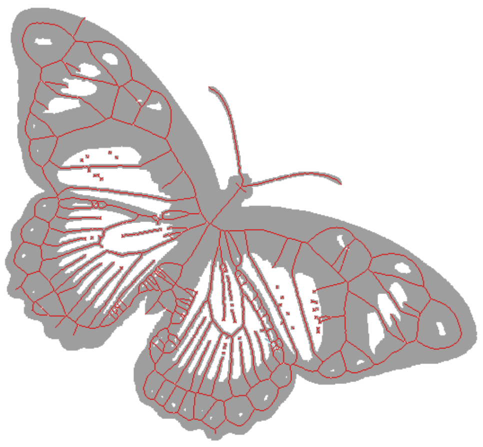


Figure 1.1: This gray figure shows an image from the MPEG7 CE Shape-1 database. Its skeleton computed with the MAT is depicted in red.

a good starting position for image compression. Figure 1.1 gives an impression of how a skeleton computed by the medial axis transform might look like.

1.1 Goals

In this thesis, we develop a near-lossless binary image codec based on the medial axis transform. We name it *MAT compression*. As the codec is near-lossless, it is not expected to keep all fine-scaled details of an object but it should be able to maintain the basic object shape and as many details as desired. In order to exploit the compact object representation that the skeletonization algorithm already gives us, we further thin the representation and propose a method for an efficient description of the remaining skeleton. Furthermore, we approximate the corresponding skeleton function with sufficient exactness. The remaining data must be transformed into an appropriate textual representation that is finally compressed with a strong compression standard called LPAQ. To evaluate the suitability of the MAT for binary image compression, we conduct a series of tests. Their evaluation is based on three quality criteria, i. e., the compression ratio, the exactness of reconstruction, and the runtime.

1.2 Related Work

In this section, we consider related work. The first three paragraphs give an overview of some general encoding strategies that are also used in MAT compression, namely skeleton pruning, function approximation, and chain coding. The last paragraph gives an overview of existing methods of binary image compression based on skeletonization algorithms similar to the MAT.

Skeleton Pruning. Pruning or rather deleting certain features of a skeleton is an important part during skeletal thinning in order to overcome noise sensitivity problems resulting in spurious skeleton branches. Spurious skeleton branches might lead to an inefficient compression. Since the distinction between lines arising from boundary perturbations of the object and lines representing desired fine-scaled details is not possible, there are many different approaches how to decide which parts of the skeleton can be removed without affecting the reconstruction quality too much. Bruckstein et al. [SB98] proposed two pruning methods that keep the homotopy of the skeleton, i. e., the number of connected components and holes inside the skeleton does not change. One of the methods prunes the skeleton based on a significance measure depending on the pruned area. The other approach is based on boundary smoothing. Latecki et al. [She+11] proposed to prune and regrow a skeleton branch depending on local and global shape information, i. e., make the decision dependent on the context of the boundary segment that is considered to be pruned. This approach preserves the connectivity of the original skeleton as well. Similar to Latecki et al., Heidrich and Tam [TH02] introduced a robust feature-preserving medial axis noise removal consisting of two steps. First, a threshold-based pruning process is applied. In order to recover lost details, features are reconstructed without reintroducing noise in a second phase. A quite different approach was introduced by Hamarneh and Ward [WH10]. They developed a groupwise skeletonization framework that requires a group of shapes and computes a significance measure for each. The results are combined into a single significance measure for each branch of a shape.

Piecewise Linear Function Approximation. One part of MAT compression is the sparse representation and approximation of the corresponding medial axis function. A piecewise linear approximation of a two-dimensional digital curve or function offers a good possibility to achieve this goal. In general, such approximation algorithms are divided into two categories as mentioned by Kolesnikov et al. [KF03].

- min-ϵ problem: The goal of a min-ϵ problem is the approximation of a function with a given number N of linear line segments while minimizing the error between the approximated and the original function.
- min-# problem: Given an approximation error tolerance ϵ, the number of line segments needed to approximate the function without exceeding ϵ is minimized.

If we are given a set of very different functions that should all be approximated with a fixed number of segments N, the approximation error would vary significantly within this set. If we fix the approximation error instead, a stable reconstruction quality is guaranteed. Thus, only the second kind of problem is relevant in this work and we will only give an overview of related work solving the min-# problem. One of the early solutions of the min-# problem was proposed by Dunham in 1986 [Dun86]. He developed an adapted recursive subdivision algorithm to partition the curve until the approximation error falls below ϵ. Another approach was given by Hamann et al. [HC94] that selects significant curve points based on a local curvature measure. As for most approaches, such as those mentioned above, yielding optimal results is computationally expensive. Therefore, Kolesnikov and Fränti [KF02] proposed a compromise between performance and runtime in terms of a fast near-optimal approximation. It consists of two steps. In the first step, a non-optimal reference solution is computed dependent on an error tolerance ϵ returning the necessary number of line segments. The approximation is refined in a second step giving a reasonable quality that depends on the precomputed number of segments. In other words, Kolesnikov et al. transformed the min-ϵ problem into a min-# problem by estimating the number of needed segments with a sub-optimal approach. They used an algorithm from Schröder et al. [SL99] that transfers the problem into a problem of finding the shortest path within a graph.

Chain Coding. Another way of encoding functions and binary contours are chain codes. Chain codes are a popular choice when a compact shape representation is needed, for example, during pattern recognition, shape analysis, or binary image compression. Thus, they are also interesting for describing skeleton lines. Chain codes encode the direction in which a shape boundary propagates pixel by pixel. In the following, an overview of different chain codes and the incorporation of chain codes in compression is given. A study from Bribiesca et al. [SBR07] reveals that compression standards using chain codes can outperform existing standards such as JBIG if the images are sufficiently small. This motivates to use chain coding in MAT compression.

The basics of chain coding were set by Freeman in 1974 [Fre74]. He introduced an 8-directional chain code (FCCE) that assigns fixed values between zero and seven to eight absolute directions of propagation. A bit later, Nunes et al. [NPM97] introduced a chain-difference coding scheme that encodes the difference between two consecutive codes followed by Bribiesca [Bri99] who invented a chain code based on grid vertices named vertex chain code (VCC). Instead of encoding the propagation direction, the number of touching object pixels is stored for each boundary vertex. Thus, only three symbols are needed as each vertex of the boundary can have at most three neighboring pixels. Otherwise, it would not be located on the boundary. New coding strategies based on VCC were introduced afterwards by Liu et al. [Liu+07]. Recently, a pattern-based chain code (PCC) was proposed by Raj et al. [RA18] that reduces redundant patterns by replacing them with new symbols.

Sánchez-Cruz et al. [SR05] proposed a compression method based on another chain code also consisting of three symbols which is called Three OrThogonal Chain Code (3OT). The next code is chosen depending on the two previously encoded links. After describing the object's contour with 3OT, the sequence is compressed with Huffman encoding [Huf06]. To increase the compression ratio, Zahir et al. [ZD07] introduced a compression algorithm that presmoothes the object's edges to decrease the change of propagation directions of the boundary. Another strong lossless chain code compression was given by Lukač and Žalik [ZL14] who applied Move-to-Front and an adapted runlength encoding (RLE) to the chain code representation. One year later, they developed a universal chain code compression method [ZML15] that utilizes different modes based on adapted versions of LZ77 coding and RLE. Both of their methods perform similarly strong since they provide different modes that can deal with long symbol runs as well as with repeating symbol patterns. Both methods outperform existing chain code compression methods which makes them interesting for shape and skeleton compression.

Binary Image Compression by Skeletonization. One of the early studies about skeleton coding was made by Maragos and Schafer in 1986 [MS86]. They used morphological operators, i. e., erosions and openings, for skeletonization. The resulting skeleton can be reconstructed by performing dilations. The structuring element applied in those operations is not restricted to circles, but also squares or rhombi can be used which enables to adapt well to different object shapes. Additionally, the authors proposed a method for thinning the skeleton by computing globally and locally minimal skeletons. They argued that these two methods improve a straightforward, time-consuming approach that sequentially removes points from the skeleton such that the remaining skeleton still guarantees a perfect reconstruction. In the following, we will call this approach *sparsification approach*. Three different modes for the encoding of the remaining skeleton were proposed:

- Block-Huffman encoding: Divide the image into rectangular blocks and encode each with Huffman encoding.
- Runlength-Huffman encoding: First, compute the runlength of consecutive black and white pixels. Afterwards, encode them with Huffman encoding. Different Huffman codes are used for black and white runs.
- Elias encoding: Vectorize the image and describe it with Elias encoding [Eli75].

The function that stores the information needed for reconstruction is encoded separately by Huffman encoding. The codec by Maragos et al. outperforms runlength encoding for graphical images whereas its performance is inferior for the compression of textual images. In 1991, Brandt et al. [BJA91] criticized this approach as its performance was only reported on a small set of nonstandard test images. Thus, they invented another codec to encode scanned documents with the help of a medial axis function that gives unconnected skeletons with a line width of one or two pixels. We call this skeleton *thick*. In order to thin out the skeleton lines to a width of one, they encoded them with chain codes that are tailored to thick lines, i. e., special chain codes for lines of width two named *double-pointed chain codes*. The chain coding serves as an intermediate step to identify thick lines. Those lines are then thinned by collapsing all points that were encoded by double-pointed chains. Brandt et al. proposed to continue thinning with piecewise linear and quadratic interpolation of the skeleton lines. Thereby, the lines are recursively subdivided until the resulting line segments can be approximated with a sufficiently low error. As the skeleton is unconnected, many single points need to be encoded separately, producing a storing overhead. The authors found that their approach can compete with runlength encoded text data and is superior if graphical images are compressed but they admit that there is still considerable room for improvement.

1.3 Outline of the Thesis

In Chapter 2, we give essential definitions and background information about the MAT and existing compression standards that will be used throughout the whole thesis.

Chapter 3 establishes the MAT compression codec. Details about its implementation are given in Chapter 4. The methods from the previous chapter are translated into pseudocode.

The efficiency, reconstruction quality, and runtime are evaluated and discussed in Chapter 5.

Finally, Chapter 6 concludes the work with a summary based on the results obtained in the previous chapter and an outlook on future work.

The appendix contains details about the use of the programs as well as some selected tables and images providing additional information about reconstruction quality, runtime, and compression ratio that were used for the evaluation in Chapter 5.

2 Background Information

In this chapter, we introduce important methods and terms that will hold throughout the thesis if not mentioned otherwise.

2.1 Definitions

Some general definitions that are tailored towards a discrete setting are presented in the following.

Binary Image $u(p)$. We represent discrete binary images as functions $u: \Omega \to \{0, 1\}$ with Ω: $\{0, ..., dim_x - 1\} \times \{0, ..., dim_y - 1\}$ where dim_x and dim_y are the image dimensions in x- and y-direction. Thus, the function maps all points $(x, y) \in \Omega$ either to zero or to one which stands for black or white, respectively.

Object O. The object $O \subseteq \Omega$ contains all black points from the original image, i. e.,

$$O := \{p \mid u(p) = 0\}.$$

8-Neighborhood $\mathcal{N}_{8_S}(p)$. Here, the 8-neighborhood $\mathcal{N}_{8_S}(p)$ of a point $p \in S$ is the set of all points $s \in S$ that are direct horizontal, vertical, or diagonal neighbors of p. Thus, $\mathcal{N}_{8_S}(p)$ is an adaptation of the common 8-neighborhood $\mathcal{N}_8(p)$ that contains all 8-adjacent points without any set restrictions. $\mathcal{N}_{8_O}(p)$ denotes, for instance, the neighbored object points of a point $p \in O$.

4-Neighborhood $\mathcal{N}_{4_S}(p)$. Analogously, the 4-neighborhood $\mathcal{N}_{4_S}(p)$ of a point p is the set of all points $p \in S$ that are direct horizontal or vertical neighbors of p.

Object Boundary δO. The object boundary $\delta O \subseteq O$ consists of those object points $p \in O$ that define the object contour, i. e., that are 8-adjacent to at least one point that is not part of the object O:

$$\delta O := \{p \in O \mid \mathcal{N}_{8_O}(p) \setminus O \neq \emptyset\}$$

Open Ball $B_\epsilon(p)$. An open ball $B_\epsilon(p_1)$ around a point $p_1 \in \Omega$ with the radius ϵ covers a disk-shaped area excluding the boundary points and can be written as

$$B_\epsilon(p_1) := \{p_2 \in \Omega \mid |p_2 - p_1| < \epsilon\}.$$

Distance Map $d(p)$. The distance map $d(p)$ in its simplest form is a function $d\colon \Omega \to \mathbb{R},\ p_1 \mapsto \min\limits_{p_2 \in \delta O} |p_2 - p_1|$. It maps each point in the image domain Ω to the minimal distance towards a point on the object boundary δO.

Medial Axis Function $f(p)$. The medial axis function (MAF) $f\colon \Sigma \subseteq \Omega \to \mathbb{R}$, $p_1 \mapsto \min\limits_{p_2 \in \delta O} |p_2 - p_1|$ is defined as a function that maps each point of the skeleton Σ to the minimal distance towards a point in the object boundary δO. Thus, $f(p)$ is the restriction of $d(p)$ to Σ. The meaning of Σ is explained below.

Skeleton Σ. A skeleton $\Sigma \subseteq O$ is a set of points in the interior of an object O. The set of points is selected as

$$\Sigma := \left\{p_1 \in O \;\middle|\; \forall p_2 \in O : B_{f(p_1)}(p_1) \nsubseteq B_{f(p_2)}(p_2)\right\}.$$

It consists of all central points p_1 of maximal inscribed discs described by $B_{f(p_1)}(p_1)$ and is thin, i. e., $\delta\Sigma = \Sigma$. A thin skeleton especially means that all lines of the skeleton have a width of one pixel. A more detailed explanation is given in Section 2.2.

Skeleton Reconstruction $R(\Sigma)$. The object O can be reconstructed from the skeleton Σ with the help of the MAF f by drawing black, filled discs with radius $f(p)$ around all $p \in \Sigma$, i. e.,

$$O = R(\Sigma) := \bigcup_{p \in \Sigma} B_{f(p)}(p).$$

Endpoints $Endpoints(\Sigma)$. A skeleton endpoint p of the skeleton Σ is a point that has either zero or one skeletal points in its 8-neighborhood or has exactly two 8-adjacent neighbors $n_1,\ n_2$ and those two points are in each other's horizontal or vertical neighborhood $\mathcal{N}_{4_\Sigma}$. Therefore, we can define the set of all endpoints in dependency of the skeleton Σ as

$$\begin{aligned} \mathit{Endpoints}(\Sigma) := \{p \in \Sigma | |\mathcal{N}_{8_\Sigma}| = 0 \vee |\mathcal{N}_{8_\Sigma}| = 1 \\ \vee (\mathcal{N}_{8_\Sigma} = \{n_1, n_2\} \wedge |\mathcal{N}_{8_\Sigma}| = 2 \wedge n_1 \in \mathcal{N}_{4_\Sigma}(n_2))\}. \end{aligned}$$

Branching Points *Branch*(Σ)**.** A branching point p of the skeleton Σ is a point such that it is neither an endpoint nor has it less than three skeletal points in its 8-neighborhood. Thus, the set of all branching points is defined as

$$\mathit{Branch}(\Sigma) := \left\{p \in \Sigma \;\middle|\; p \notin \mathit{Endpoints}(\Sigma) \wedge |N_{8_\Sigma}(p)| > 2\right\}.$$

Simple Points *Simple*(Σ)**.** We refer to a point as a simple skeletal point if it is neither an endpoint nor a branching point, i. e., the set *Simple*(Σ) of simple points is given as

$$\mathit{Simple}(\Sigma) := \Sigma \setminus \big(\mathit{Branch}(\Sigma) \cup \mathit{Endpoints}(\Sigma)\big).$$

Skeleton Lines *Line*(Σ)**.** We define a line $l \subseteq \Sigma$ of a skeleton as a set containing a connected sequence of skeletal points starting in a branching or endpoint $p_0 \in \mathit{Branch}(\Sigma) \cup \mathit{Simple}(\Sigma)$ and ending in another branching or endpoint $p_{n-1} \in \mathit{Branch}(\Sigma) \cup \mathit{Simple}(\Sigma)$. For all points p_i in between of p_0 and p_{n-1}, $p_i \in \mathit{Simple}(\Sigma)$ has to hold. The set of lines *Line*(Σ) consists of all lines $l \subseteq \Sigma$ such that

$$\Sigma = \bigcup_{l \in \mathit{Line}(\Sigma)} l.$$

Near-Lossless Compression. A lossless compression standard finds a sparse representation of some input data, for instance, by deleting redundant information, such that the data size is reduced. Furthermore, the original input data can be reconstructed perfectly. In contrast to this, a near-lossless compression standard neglects the requirement of perfect reconstruction. Thus, losing some details of the input data is allowed.

2.2 Medial Axis Transform

The so-called medial axis transform (MAT) is a skeletonization algorithm that yields a simple and intuitive representation of an object. It was developed by Blum in 1967 as he saw the need of a shape processing mechanism [Blu67]. This representation should consist of a set of unique and connected shape attributes making a perfect object reconstruction possible in a continuous setting. There are many analogies

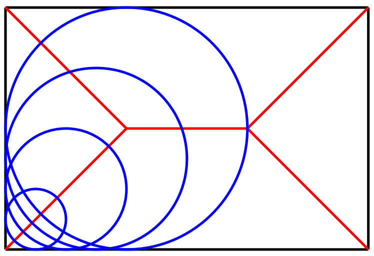


Figure 2.1: A visualization of a rectangular object (black lines) and its corresponding skeleton depicted in red. Some circles with radii given by the MAF that are used for reconstruction are shown as a way of example.

giving an intuition about the principle of MAT. Amongst them, the maybe most well-known one is the so-called *Grass-Fire Analogy*.

Grass-Fire Analogy. Given a rectangular image I depicting an object O, let the whole area I be covered with grass. A grass-fire starts burning on the object boundary δO and propagates uniformly in all directions. On its way, it leaves burned grass spots. Thus, as each grass spot can only burn once, two or more fire fronts may collide eventually and cancel each other out. In this case, we mark their collision points. After all waves have canceled out, the marked points inside the object O form the skeleton. This procedure is called skeletonization. Additionally, if we map the waves' traveling distances to their collision points, it is possible to reconstruct the original object perfectly. The MAF is built by this mapping of skeleton points to distances. The reconstruction is described in Subsection 2.2.2. An example of the MAT can be seen in Figure 2.1.

2.2.1 Skeletonization

There are many different methods of skeletonization algorithms. In this work, we focus exclusively on five methods described by Peter [Pet10]. All of them are further advancements of the so-called Hamilton-Jacobi skeletonization described by Siddiqi et al. [Sid+02]. The resulting skeletons are homotopy preserving, i. e., the amount of holes and connected components equals those of the original object. Furthermore, the skeletons are thin, meaning that all lines have a width of one pixel. An intuition of the working principles of the five methods is given below.

Flux-Ordered Thinning (FOT). FOT consists of three steps. First, the distance map $d : \Omega \to \mathbb{R}$ is computed for each point $p \in \Omega$, i. e., p is mapped to the distance towards the nearest point on the object boundary. Next, the gradient vector field ∇d of the distance map is determined with $\nabla(\frac{\partial}{\partial x}, \frac{\partial}{\partial y})^T$. Points with large negative values in *div* ∇d represent the sinks of this vector field. We interpret them as skeleton points. The resulting flux map is then used for the thinning process, i. e., we remove all skeleton points with a flux map value smaller than a user-defined threshold τ.

FOD. FOD in its spirit is identical to FOT. The only difference consists in the algorithm that is used for the distance map computation.

Flux-Ordered Adaptive Thinning (FOA). FOA is an extension of FOT but adapts the threshold to the given shape, i. e., performs an educated guess for the right choice of τ during a second computation step called Secondary MAT Detection (SMD). As SMD will not be needed in the following, we do not give a detailed explanation here. In general, FOA gives the most complex skeletons amongst all five methods but tends towards having the smallest reconstruction error.

Maximal Disc Thinning (MDT). MDT is an extension of the RT scheme [RT05] that thins points according to their values in the computed distance map. In contrast to the previous methods, a flux map computation is not necessary. The computed skeleton is pruned in order to minimize the homotopy-induced occurrence of skeleton branches. In other words, as the method is expected to preserve homotopy, long branches connect points that would be isolated in the skeleton otherwise. Those branches are reduced by pruning.

Flux-Ordered Maximal Disc Thinning (FMDT). Similarly to FOA, FMDT also uses SMD, but this time, SMD fully substitutes the need of the user-defined parameter τ, i. e., no parameter must be given beforehand.

2.2.2 Object Reconstruction

Given a skeleton Σ and its MAF $f : \Sigma \longrightarrow \mathbb{R}$, we can reconstruct the original object by drawing a filled black circle with the radius $f(p)$ centered at point p around all $p \in \Sigma$. Unfortunately, circles cannot be drawn perfectly on a discrete grid. Thus, discretization artifacts can occur, meaning that the exactness of reconstruction is not perfect.

2.3 PAQ

PAQ is a series of very powerful lossless compression standards developed by Matt Mahoney [Mah13] and is one of the best existing entropy coding algorithms. It can be interpreted as an evolution of Prediction by Partial Matching (PPM) [Bla+12]. We introduce PPM in the next paragraph. In contrast to PPM, PAQ drops the requirement to use a fixed length context and fixed context type. Thereby, contexts are build by a set of previously encoded symbols. Indeed, there are several different types of contexts that can be taken into account as only the efficiency of the context function matters. A context of a fixed length n of previously seen symbols as in the predictor PPM, for example, is called *n-gram* or *n-th order* context. Beside consecutive sequences of symbols, sparse contexts with gaps are possible as well, e. g., using every second symbol of a word only. Since PAQ uses many different predictors that possibly yield many different predictions, these predictions are combined such that the resulting prediction can be encoded with arithmetic coding. Arithmetic coding is explained in the next to last paragraph. The combination of predictions into a single prediction is called context mixing. An insight into different ways of context mixing is given in the last paragraph of this section.

LPAQ1 is a light version of PAQ that is faster and yields a good compression. Hence, we use it for MAT compression.

Prediction by Partial Matching (PPM). PPM predicts the next symbol s that we want to encode with the help of a context c consisting of the k previously encoded symbols and is therefore also called k-th order PPM. Thereby, we compute the relative occurrence of s that appears immediately after the context c within the already encoded data, i. e., the conditional probability of s is adapted to the used context. Hence, PPM can be interpreted as a combination of higher order and adaptive encoding. If s does not occur in the recent context of length k, we decrease its length by one. If the symbol s was never seen in any context of length zero to k, we encode it in a context of -1-th order . In such a context, each symbol within the encoded word has equal probability.

Arithmetic Coding. Arithmetic coding maps a source word w to a single code word c. Therefore, it assigns w to an interval I and chooses a number inside this interval for c. To determine I, each symbol is assigned a subinterval of the start interval $[0, 1)$ according to its probability. For example, given a word w consisting of three symbols A, B, and C with probabilities 0.2, 0.3, and 0.5, respectively, we assign A to the subinterval $[0, 0.2)$, B to $[0.2, 0.5)$, and C to $[0.5, 1)$. When encoding the next symbol s of w, we pick the new interval I as the current subinterval corresponding to s and repeat the procedure until w is encoded.

Context Mixing. In early versions of PAQ, context mixing consists of weighted averaging, i. e., each context is assigned a fixed weight w. Then, all weights indicating the occurrence of a certain symbol s are averaged and used to determine the estimated probability.
In later versions, those weights are not fixed. Instead, they are adapted during

encoding using a gradient descent scheme for the coding costs. Following the cost gradient, we minimize the cost function meaning that the symbol weights are updated such that the loss during encoding is kept minimal.
In the most recent versions of PAQ, neural networks estimate the weights. Thereby, the returned probabilities of the models are used as input of the neural networks. Its output is the prediction probability of the next symbol.

2.4 Other Compression Standards

In the following, we present an overview of compression algorithms that are used for comparative experiments in Chapter 5 and sketch the ideas behind them.

JPEG. JPEG was created in 1992 by the Joint Photographic Experts Group Committee as a lossy compression standard [Wal92]. It is a so-called transform coder as it exploits global image representations during compression and eases a differentiation between high frequency and low frequency details. Furthermore, high frequency coefficients are stored with lower accuracy as they are not important for human perception. The compression consists of six steps:

1. Color space conversion: Given a color image represented by three separate color channels Red, Green, and Blue, namely the RGB color space, the channels are transformed into a color space named YCC that yields less channel correlation. YCC consists of a luma and two chroma channels. The luma channel encodes the brightness of the image. The two chroma channels represent the image colors in blue-yellow and in red-green direction.
2. Channel subsampling: JPEG subsamples the two chroma channels by averaging their values within 2×2 blocks. This step is lossy although the human eye barely recognizes any differences because of its small sensitivity to slight color changes.
3. Discrete Cosine Transformation (DCT): We apply DCT to 8×8 blocks of all color channels separately. This decomposes each channel in its frequencies with the cosine function as its basis. The frequency coefficient computation is lossless.
4. Coefficient quantization: The coefficients obtained by the DCT are quantized, i. e., coefficients representing high-frequency details are stored with less accuracy than low-frequency details as they are less important.
5. Coefficient reordering: The quantized coefficients are reordered from low to high frequencies.
6. Encoding: Finally, an entropy encoder encodes the reordered coefficients.

A more detailed insight into JPEG can be found in the JPEG compression standard paper [Wal92].

JPEG-2000. JPEG-2000 is a successor of the transform coder JPEG. Its steps resemble those of JPEG:

1. Color space transformation.
2. Tiling: Decompose the image into non-overlapping rectangular tiles.
3. Discrete Wavelet Transform (DWT): Represent each tile by wavelets that are localized in space and frequency. The result is an image that is decomposed into subbands of similar frequency.
4. Quantization: The resulting coefficients of all tile subbands are quantized, i. e., reduced in their precision. Afterwards, they are collected into so-called code blocks.
5. Encoding and ordering: JPEG-2000 decomposes the coefficients of the code blocks into bit planes according to their binary representation, i. e., each bit plane contains bits with equal significance. These bit planes are then entropy encoded.

More details can be found in the JPEG-2000 compression standard by Skodras et al. [SCE01].

JBIG. The Joint Bi-level Image Expert Group (JBIG) is a lossless compression standard for binary images [Kyr99]. It achieves a high compression by predicting the next pixel based on adaptive two-dimensional contexts and arithmetic coding. It offers several variations of progressive modes during which a coarse image resolution is computed and builds up the image by transmitting resolution layers giving more detailed image information. To create coarser resolutions, JBIG uses a lowpass filter. Unfortunately, this filter can lead to wrongly classified pixels. Thus, a table handling exceptions is stored such that, e. g., preservation of edges and correct encoding of periodic patterns is guaranteed.

Runlength Encoding (RLE). Given a binary image consisting of values zero and one, we count the absolute occurrence of a symbol until the symbol value changes. Thereby, the sequence is encoded by storing all such symbol runs by their number of occurrence followed by its symbol value sequentially. For example, RLE transforms the input 11101100 into the sequence 3 1 1 0 2 1 2 0.

Huffman Encoding. Huffman encoding is an entropy coder that was introduced by Huffman [Huf06]. First, for all symbols s_i, the relative occurrence p_i is computed.

The goal of Huffman encoding is to build a code tree that encodes all symbols of a word. Therefore, we interpret the symbols as its leaves. The tree is constructed recursively by merging those two nodes $s_i,\ s_j$ with the smallest probabilities into a new node s_{new} with p_{new} as the sum of the corresponding probabilities $p_i\ +\ p_j$ until there is only one node remaining. For all nodes, we assign zero to the branch of the left children and one to the right branch. Finally, we can encode a symbol s_i by walking the path from the root to its leaf node and concatenating the values that we assigned to the branches.

Modified Huffman Encoding. During modified Huffman encoding, the binary image is first described by runlength encoding (RLE). The resulting stream of numbers and symbols is then compressed with Huffman encoding.

3 MAT Compression

In this chapter, we propose a new approach for binary image compression that we call *MAT compression*. We divide the algorithm into a preprocessing and a compression part. The preprocessing aims to increase the compactness of the skeleton representation such that it can be compressed efficiently in the next step. We approach this goal as follows:

1. Skeleton Preprocessing: We increase the efficiency of compression by
   - Pruning: Subsection 3.1.1 introduces a method for data reduction by removal of selected skeleton lines.
   - Smoothing: We discuss the handling of discretization artifacts of skeleton thinning in Subsection 3.1.2.
2. Compression: The actual compression task is divided into
   - MAF approximation: In Subsection 3.2.2, we formulate MAF approximation in terms of a min-# problem.
   - Chain coding: A chain code representation of the skeleton is developed in Subsection 3.2.2. The representation together with the approximation of the MAF is transferred into an encoding scheme.

3.1 Preprocessing

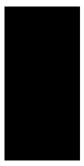
The preprocessing steps deal with discretization problems of the MAT in order to increase the compression efficiency.

3.1.1 Skeleton Pruning

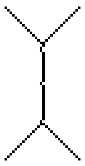
Unfortunately, the MAT is sensitive to boundary perturbations. Refer to Figure 3.1 for an example of MAT's noise sensitivity. For a simple geometric form such as a rectangle, the MAT is relatively simple. After adding perturbations on the object boundary, new unintuitive branches arise. They result in a more complex skeleton although they represent only small parts of the object. Thus, given a noisy image, the skeleton will contain many branches while only a subset amongst them represents the actual object. Storing those unnecessary branches is inefficient. Additionally, the decomposition of the skeleton into lines as defined in Section 2.1 can lead to many small fragments, i. e., to an object oversegmentation. Often, these small fragments are not important for the reconstruction quality and hence yield an inefficient compression. Like the branches arising from boundary perturbations, they should be removed before compression. Then again, the distinction between those branches and branches that come from desired boundary details is quite difficult. As we have seen in Section 1.2, there are many different ideas for handling this problem. In this work, we will approach this problem with a simpler attempt that can be put in a scale-space context.

Therefore, our goal is to assign each skeleton line $l \in Line(\Sigma)$ an importance value $\mathcal{I}(l)$. $\mathcal{I}(l)$ gives the number of pixels that the skeleton line l contributes to the reconstructed object $R(\Sigma)$. The differentiation between the reconstructed object and the original object is justified by the fact that the object cannot be reconstructed perfectly due to discretization errors that occur when approximating the reconstruction circles on a discrete image grid. Thus, it would be unfair to measure the importance of a skeleton line with respect to an object quality that can never be achieved in a discrete setting. Assigning the pixel contribution or the importance value, respectively, to a line, we implicitly determine the amount of information that is carried by the line, meaning we divided the skeleton lines into different scale-spaces. The lower $\mathcal{I}(l)$ is, the less important and the more detailed information is carried by the line. Hence, we might remove all lines l whose importance values $\mathcal{I}(l)$ are falling below a user-defined threshold p. The new skeleton Σ_{pruned} is defined as

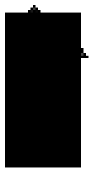
$$\Sigma_{pruned} = \bigcup_{\substack{l \in L(\Sigma) \\ \mathcal{I}(l) > p}} l.$$



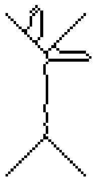
(a) simple rectangle



(b) skeleton of the simple rectangle



(c) rectangle with boundary perturbations



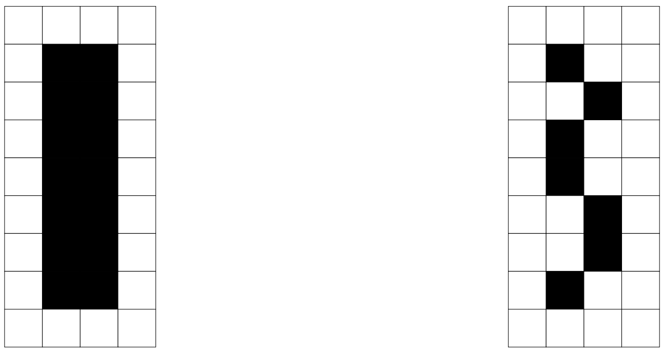
(d) skeleton of the perturbed rectangle

Figure 3.1: The impact of noise on the MAT is demonstrated for synthetically designed images without (top left) and with (bottom left) boundary perturbations. The corresponding skeletons are computed with FOD.

The threshold p enables us to determine the reconstruction's level of details. This means as well, that we delete branches arising due to small boundary perturbations. At the same time, we also neglect details with the same size as the noise. Consequently, the user must be able to estimate the size of noise appropriately. The smaller the desired level of details and noise are, the higher can the parameter for skeleton pruning be chosen and the higher becomes the compression ratio.

3.1.2 Line Smoothing

In this section, we deal with another discretization problem that occurs during skeleton thinning. Given a medial line that would be located in between of two grid points, it is discretized by using the pixels adjacent to the optimal line. This produces a line of width two, such as depicted in Figure 3.2a. When thinning the skeleton, it is possible that points left and right from the optimal line are removed, resulting in a shaky line with pixels deviating by at most ϵ pixels from the optimum. Such a line could look like depicted in Figure 3.2b with $\epsilon = 1$. ϵ can be tuned individually. In the compression step of Subsection 3.2.2, we will encode skeleton lines with chain codes. However, for further compression, encoding shaky contours is not efficient as the steady change in direction impends long chain symbol runs. Thus, smoothing the lines of Σ_{pruned} would not alter the image quality remarkably



(a) thick line of width two **(b)** thinned line of width one

Figure 3.2: If the optimal medial line would be located between the grid, it needs to be approximated by a discrete line with pixels adjacent to the optimum, i. e., by pixels left and right of this line as in Figure 3.2a. Figure 3.2b shows a possible outcome after thinning.

but increase the efficiency of chain coding. Therefore, we need to identify shaky lines and smooth them to reduce discretization artifacts. There are two possibilities of smoothing:

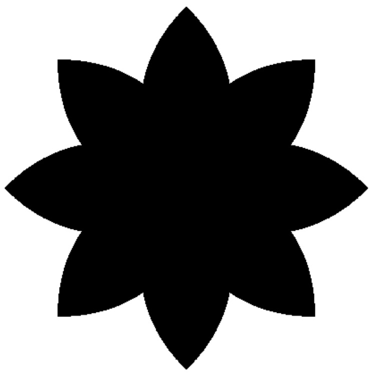
- Smoothing horizontally: If the line propagates from left to right, i. e., in x-direction, the x-coordinate increases by one for each subsequent pixel. If some of the pixels' y-coordinates are localized above or below the optimal line, the horizontal line might be called *shaky*. When smoothing such a line, all its y-coordinates have to be set to the same near-optimal y-coordinate.
- Smoothing vertically: Analogously, a shaky vertical line means pixels deviating in x-direction from an optimal line that propagates in y-direction. Therefore, we smooth the line by changing its x-coordinates.

After smoothing, a fully preprocessed skeleton $\Sigma_{smoothed}$ is given by

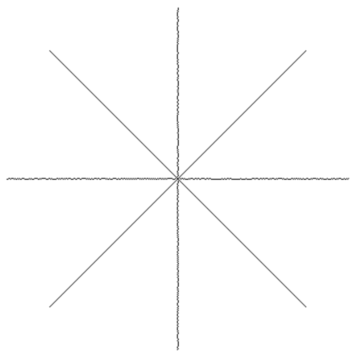
$$\Sigma_{smoothed} = \bigcup_{l \in Line(\Sigma_{pruned})} S(l)$$

where $S(l)$ is the smoothing function smoothing a line $l \in L(\Sigma_{pruned})$.

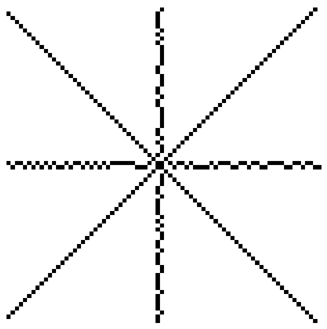
An example demonstrating the effects of the methods from Subsection 3.1.1 and Subsection 3.1.2 is shown in Figure 3.3. Figure 3.3c depicts a detail of the skeleton that illustrates the discretization artifacts of thinning. A close-up of the pruned and smoothed skeleton can be seen in Figure 3.3d. Clearly, some centered skeleton points are removed during pruning. The remaining lines were successfully smoothed in x- and y-direction.



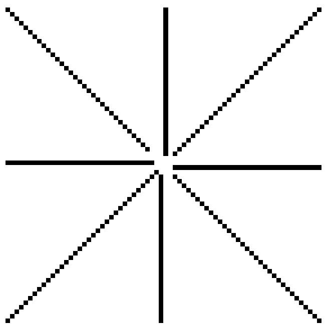
(a) device2-14



(b) skeleton thinned with FOD



(c) close-up of the thinned skeleton



(d) close-up of the preprocessed skeleton

Figure 3.3: The first image is taken from the MPEG7 database. The next two images depict the skeleton and its close-up. The discretization problems of the thinning process are visible in the third image. The effect of the preprocessing is shown in the bottom right. The depicted skeleton has been pruned with a pruning threshold of five.

3.2 Compression

For compression, we first need to consider the following aspects separately:

- Skeleton $\Sigma_{smoothed}$: $\Sigma_{smoothed}$ contains all points that are part of the preprocessed skeleton, i. e., is the graphical skeleton representation.
- MAF: The medial axis function $f(p)$ maps a radius to each skeletal point $p \in \Sigma_{smoothed}$. It is needed to reconstruct the object from the skeleton $\Sigma_{smoothed}$.

Thus, both of them need to be encoded. We start with a method for compressing the MAF and incorporate its result in the next step covered in Subsection 3.2.2.

3.2.1 MAF Compression

We divide the MAF in $|Line(\Sigma_{smoothed})|$ different functions, i. e., one MAF for each line $l \in Line(\Sigma_{smoothed})$. We thin the function by identifying points such that after removing radii in between, these radii can be reconstructed by piecewise linear approximation between the picked points. As discussed in Section 1.2, we approach the MAF approximation as a min-# problem. Kolesnikov and Fränti [KF02] provide a fast and near-optimal solution for this problem. Their algorithm offers two modes. In its strong mode, the algorithm finds an approximation such that the approximation error e fulfills $e \leq a$ where a is a user-defined threshold. The weak form approaches the error e such that $e \approx a$. Kolesnikov et al. state that the algorithm in its weak form proposes a fast and sufficiently reasonable result for practical applications. It consists of two steps:

1. Finding a reference solution.
2. Optimizing this solution.

In the following, assume we are given the tolerance parameter a and a line $l \in Line(\Sigma_{smoothed})$ consisting of n points $p_1, ..., p_n$.

Finding a Reference Solution

Kolesnikov et al. propose to use an algorithm developed by Schröder et al. [SL99] to derive a reliable reference solution. Schröder's method transfers the approximation of a function consisting of n points to a shortest path problem on a directed acyclic graph. In particular, the authors define a function that computes the least number of line segments, i. e., the shortest path connecting the line's first and last point. Thereby, the maximal local distortion between the approximated values in between those points and the original values is not allowed to exceed the error tolerance parameter a. For the local distortion, they use an L_∞-norme.

Optimizing the Solution

Kolesnikov et al. base the optimization of the reference solution on their previous work [KF03] where they invented a reduced-search dynamic programming approach of finding an optimal approximation to reduce runtime. They define a discrete state space Ω such that each point $(i, j) \in \Omega$ represents the subproblem of approximating the radii between the line's starting point p_1 and p_i with j line segments where $0 \leq j \leq M$ and $1 \leq i \leq n$. M is the minimal number of segments that is needed to approximate the function between the points p_1 and p_n with sufficient exactness. Thus, interpret a path through the state space from $(1, 0)$ to (n, M) as a function consisting of $M + 1$ points that starts in p_1 and ends at p_n. The remaining $M - 1$

points are picked from the set $\{p_2, ..., p_{n-1}\}$. Considering this, the reference solution from the previous step can be expressed in terms of a path through Ω. The authors decrease the search space by inventing a corridor within Ω bounded from the left and from the right. The corridor width can be chosen freely. Within this corridor, dynamic programming is performed that minimizes the error costs of the optimal approximation path. The authors propose an additive error metric L_2 that yields a quality improvement in contrast to the proposed L_∞-norm of step one. The L_2-norm is not used in the first step as the additive character is not suitable for the measurement of local distortion.

Although the threshold a can be chosen freely, Kolesnikov et al. propose to choose a within [0.5, 2]. The closer it is to 0.5, the more suitable it is for a detailed approximation as more points need to be chosen. For all points $p \in \Sigma_{smoothed}$ that are not picked for approximation, we reset its radius in the MAF.

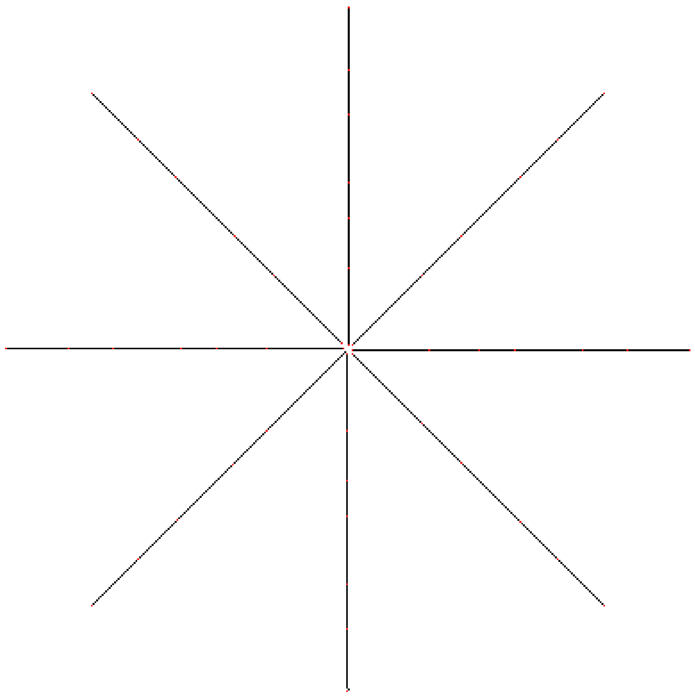
Refer to Figure 3.4 for an example result of function point selection.

3.2.2 Skeleton Compression

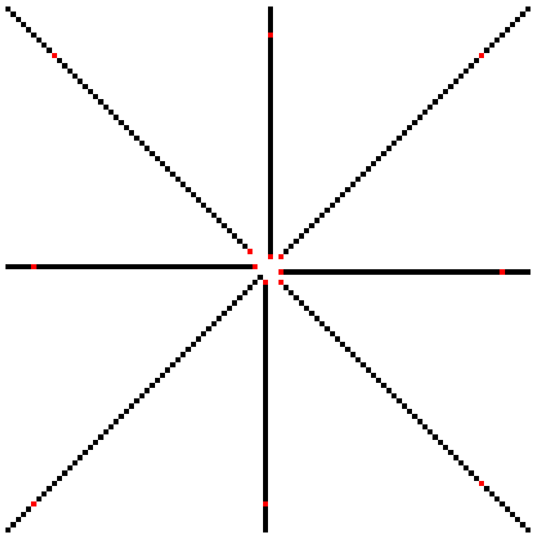
Given that we can identify the points whose radii are not removed from the MAF, the next step is the lossless description of the skeleton. For this, we use a relative 8-directional chain code (R8CC) that efficiently encodes the skeleton line by line. Its encoding is described in Figure 3.5a. A relative chain code encodes a line by first describing the initial absolute propagation direction between the first two pixels. Afterwards, the remaining pixels are chain coded by their relative perspective with respect to the preceding directional chain value. This means that the encoding perspective is related to the moving perspective of the line. We encode the initial absolute direction of a line according to Figure 3.5b.

We encode, for instance, the line in Figure 3.6a from left to right as NE FR F FR L and store it as 1 1 0 1 4 where the first number describes the absolute direction of propagation NE between the first two pixels and the following four numbers encode the relative directions FR, F, FR, and L of the subsequent four pixels. Refer to Figure 3.6b to see an illustration of the chain code.

Clearly, we need to incorporate the coordinates of the first pixel into the sequence of numbers such that we can locate the line within a grid. Additionally, we may not forget about the radii stored in the MAF. After approximating the radii as described in Subsection 3.2.1, the MAF contains only a subset of all skeleton points with a radius larger than zero. In particular, the radii of the first and last point of a line must always be kept. Assume that no other radii are kept beside these two. We

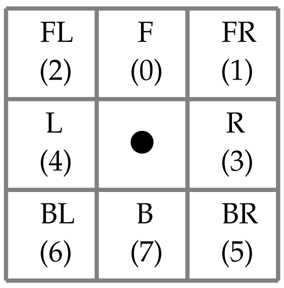


(a) processed skeleton with marked interpolation points

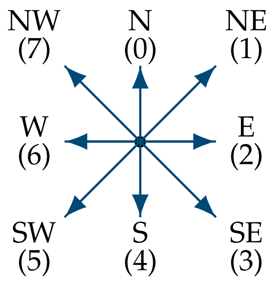


(b) close-up of the processed skeleton

Figure 3.4: Figure 3.4a shows the preprocessed skeleton of Figure 3.3. The red pixels mark the pixels picked for interpolation, i. e., the points whose radii are kept during encoding. The radii of the remaining black points are removed from the MAF. Figure 3.4b shows a more detailed view of the central part of the skeleton. The approximation error threshold a was set to one.

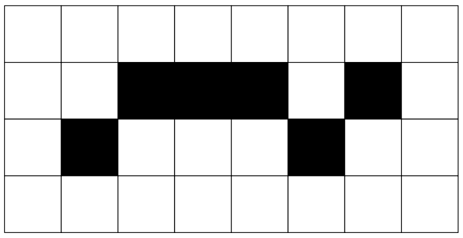


(a) relative 8-directional chain code (R8CC)

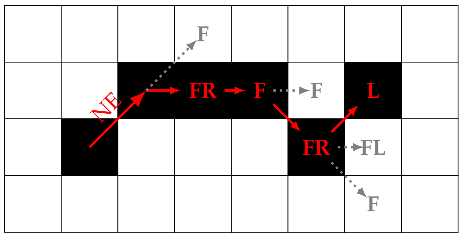


(b) 8-directional absolute directions

Figure 3.5: Figure 3.5a shows the encoding of the relative 8-directional chain code. *F* stands for *Front*, *B* for *Back*, L for *Left*, and so on. Figure 3.5b is a representation of the encoding of the absolute 8-directional chain code. *N* stands for *North*, *S* for *South* etc. The numbers in brackets are the encodings used in the implementation.



(a) simple line



(b) line with annotated orientations

Figure 3.6: Figure 3.6a is an example for a thinned line that can be encoded with R8CC. The initial absolute and some relative orientations are plotted in Figure 3.6b. The red arrows represent the propagation direction of the line. The corresponding red codes at the arrow tips stand for the recent relative directions that are used for chain coding. The dotted gray lines and its descriptions depict relative directions that are not needed for the chain description. They were added to ease comprehension

encode the image dimensions, the lines' initial coordinates, and the radius of its start point via

```
<x-dimension> <y-dimension>
<first x-coordinate> <first y-coordinate> <first radius>
<chain code>+
```

The plus sign (+) indicates that one or more subsequent chain code values are stored.

We add the radius of the endpoint at the end of the chain code sequence. Therefore, we signalize the end of the sequence with an End-Of-Sequence symbol EOS. By this, we can even encode all lines $l \in Line(\Sigma_{smoothed})$ sequentially. As a side effect, a unique decomposition into lines is possible during decoding.

The encoding scheme is expressed as:

```
<x-dimension> <y-dimension>
(<first x-coordinate> < first y-coordinate> <first radius>
 <chain code>+ EOS <last radius>)*
```

The star (*) indicates zero or more occurrences of the expression within the preceding round brackets.

Next, we drop the assumption that only radii corresponding to a start and endpoint of a line are kept for approximation. Thus, we need to encode the MAF for points in between that have a radius larger than zero. Thereby, we introduce the symbol EOCC signalizing End-of-Chain-Coding that helps us to separate the encoding of the MAF from the skeleton encoding. This results in the encoding scheme:

```
<x-dimension> <y-dimension>
(<first x-coordinate> <first y-coordinate> <first radius>
 <chain code>+ EOS <last radius>)*           /* skeleton lines */
EOCC
(<x-coordinate> <y-coordinate> <radius>)*    /* remaining MAF */
```

For the sake of efficiency and unambiguity, lines consisting only of a single point are encoded after the EOCC symbol as well.

As a way of example, let us consider again the line from Figure 3.6 as simple skeleton. Assume that the radius of the first point p_1 is given by the MAF as $f(p_1) = 9$, the radius of the last point p_6 by $f(p_6) = 5$, and the radii of p_3 by $f(p_3) = 8$, respectively. All other radii are zero. The origin $(0, 0)$ of the grid is located in the top left. Following the encoding scheme, the skeleton is represented as follows:

```
8 4                         /* x- and y-dimension */
1 2 9 1 1 0 1 4 EOS 5       /* line encoding */
EOCC
3 1 8                       /* p3 = (3,1), f(p3) = 8 */
```

3.2.3 PAQ Compression

After preprocessing and transforming the skeleton into the encoding scheme described in Subsection 3.2.2, the data is finally compressed with LPAQ1 - a very strong and efficient version of PAQ developed by Matt Mahoney [Mah13]. PAQ is introduced in Section 2.3.

3.3 Decoding

The decoding is done in five steps:

1. Decompress the LPAQ encoded file.
2. Extract the skeleton linewise from the file until the EOCC symbol appears. Add the decoded points and radii to the skeleton line and the MAF, respectively.
3. Expand the MAF with the coordinates and radii after the EOCC symbol.
4. For all lines, linearly approximate the missing radii r_k of the points p_k lying between two points p_i and p_j as
$$r_k = r_i + \frac{r_j - r_i}{j - i} \cdot (j - k)$$
where r_i and r_j are the radii corresponding to p_i and p_j.
5. Reconstruct the object as described in Subsection 2.2.2.

3.4 Properties

In this section, properties of MAT compression are discussed.

Exactness of Reconstruction. Due to discretization errors while thinning the skeleton and discrete circle approximation, skeleton reconstruction is not exact. Therefore, we need a quality measurement for the evaluation of skeleton reconstruction in Chapter 5. The reconstruction error determines the percentage of false positive and false negative pixels between the reconstructed and the original image. The false positives and false negatives are called *erroneous* pixels. Hence, we measure the relative error e as
$$e := \frac{err}{dim_x \cdot dim_y}$$
where dim_x and dim_y are the x- and y-dimensions of the image and err is defined according to Peter [Pet10] as
$$err := \left|\left\{p \;\middle|\; \left(p \in R(\Sigma) \wedge p \notin O\right) \vee \left(p \notin R(\Sigma) \wedge p \in O\right)\right\}\right|.$$

Invariances. As the skeleton is invariant under translations [Pet10], compression quality is invariant under translations as well. In contrast to the translation invariance, MAT compression is not robust under rotational changes. The reason is that the skeleton itself is not invariant under rotations as discretization artifacts can occur depending on the rotation angle of the object, resulting in varying reconstruction quality [Pet10].

Homotopy. Homotopy that was preserved during thinning according to Peter [Pet10; PB13] is destroyed during skeleton pruning as we only keep those parts that are relevant for reconstruction. For compression purposes, homotopy preservation is not necessary as long as it does not influence the exactness of reconstruction more than desired. Additionally, the homotopy of the reconstructed object is not guaranteed after reconstruction as well. This is shown in Section 1.2.

3.5 Comparative Sparsification Approach

There are multiple ways how to thin and compress a skeleton. The approach developed above is one way of compression. Another one is expressed in terms of sparsification similar to the sparsification approach we encountered in Paragraph 1.2. Sparsification can be interpreted as a thinning process during which points are sequentially selected according to a specific property and removed. For our purposes, we want to achieve the most exact reconstruction quality. To this end, assume we want to delete n skeleton points. To get the best possible reconstruction quality with respect to the removal of n points, we need to delete those pixels that have the least contribution to the reconstruction quality. This may give us a set of unconnected points and lines. The higher the sparsification ratio is, the more will the skeleton consist of single points. The sparsified skeleton and its remaining MAF are transformed into the encoding scheme from Subsection 3.2.2 and finally compressed with LPAQ.

This approach could serve as a kind of ground truth with respect to the reconstruction quality as it removes exactly those points that have the least influence.

4 Implementation

All code of this thesis is written in C99. The algorithms of this chapter are written in pseudcode that abstracts from the details of the programming language C.

4.1 Prerequisites

According to Section 2.1, we represent the skeleton Σ as a set of unordered skeleton points. The points $p \in \Sigma$ are tuples (x, y). The MAF f is a partial function $f\colon \Sigma \subseteq \Omega \to \mathbb{N}$, $p_i \mapsto r_i$ for $i \in \{0, \ldots, |\Sigma| - 1]\}$ that maps a point p_i to its corresponding radius r_i. In the following pseudocodes, the MAF f is handed over as an argument to keep the number of arguments small. Nevertheless, f is known within the algorithms and can be modified. Furthermore, each line $l \in \mathit{Line}(\Sigma)$ consisting of n points $\{p_0, \ldots, p_{n-1}\}$ has a partial function $\mathit{line}_l\colon \mathbb{N} \to \Sigma$, $i \mapsto p_i$ for $i \in \{0, \ldots, |\Sigma| - 1\}$ that defines the ordering of the point sequence. For instance, the second point of a line l can be accessed as $\mathit{line}_l(1)$. $|l| = n$ gives the length of the line. The sets $\mathit{Endpoints}(\Sigma)$, $\mathit{Branch}(\Sigma)$, $\mathit{Simple}(\Sigma)$, $\mathcal{N}_{8_S}(p)$, and $\mathcal{N}_{4_S}(p)$ from Section 2.1 are given as well. We can access data as demonstrated below.

```
p$_i$ ←$line_l$(i)              /* point p$_i$ of line l */
// or
(x,y) ← $line_l$(i)

r$_i$ ←$f$(p$_i$)             /* radius corresponding to the point p$_i$ */
x ← p$_i$.x               /* x-coordinate */
y ← p$_i$.y               /* y-coordinate */
neighbors ← $\mathcal{N}_{8_\Sigma}$(p$_i$)   /* set of neighboring skeleton points */
|neighbors|             /* the number of neighbors of p$_i$ */
```

4.1.1 Reconstruction

Skeleton reconstruction is an essential part of MAT compression and decompression. The pseudocode for reconstruction is shown in Algorithm 4.1. According to the definition in Subsection 2.2.2, we draw filled black circles with radius $f(p_i)$ around each point p_i.

Algorithm 4.1 Reconstruct a given skeleton Σ with the MAF f and return the resulting image.

```
procedure reconstruct(Σ, xDim, yDim) {
    img[xDim][yDim]          /* initialize a white image */
    for point ∈ Σ do
      for j ∈ {-f(point)+1,...,f(point)-1} do
        for k ∈ {-radius+1,...,radius-1} do
          /* color each pixel inside the disc black */
          if $j^2 + k^2 \leq f^2$(point) then
            img[j][k] ← 0
          end if
        end for
      end for
    end for
    return img
}
```

4.1.2 Skeleton Segmentation

As the line representation is useful for preprocessing and compression tasks, we define a method that decomposes Σ into its lines.

Recall that a line is defined as a unique connected sequence of points from a point p_0 to a point p_{n-1}, such that $p_0,\ p_{n-1} \in \mathit{Branch}(\Sigma) \cup \mathit{Endpoints}(\Sigma)$ and $p_i \in \mathit{Simple}(\Sigma)$ must hold for all points p_i in between. Thus, the line segmentation algorithm iterates through all skeleton points $p_0 \in \mathit{Branch}(\Sigma) \cup \mathit{Endpoints}(\Sigma)$ and explores their outgoing branches until the next point $p_{n-1} \in \mathit{Branch}(\Sigma) \cup \mathit{Endpoints}(\Sigma)$ is found. This process is described in Algorithm 4.2.

In case of a closed polygon, i. e., if we do not have any end or branching points, we just select one of the contour points and treat it as an endpoint of our new line segment. We proceed adding points to the line until we encounter this endpoint again. Thus, the procedure is similar to the above described behavior. For the sake of simplicity, we do not include this special case in Algorithm 4.2.

Algorithm 4.2 Pseudocode for the line segmentation algorithm. Given the skeleton Σ, it returns the corresponding decomposition into skeleton lines.

```
procedure segmentation(Σ) {
    lineCollection ← ∅ /* set of skeletal lines */
    visited ← ∅        /* set of visited points */

    /* run through all branches and endpoints */
    for p ∈ $Branch(\Sigma)$ ∪ $Endpoints(\Sigma)$ do
      visited ← visited ∪ {p}
      /* iterate through all outgoing branches of $p$ */
      for n ∈ $\mathcal{N}_{8_\Sigma}$(p) do
        l ← ∅       /* line */
        x ← n       /* temporaly saving recent point */
        /* add points to $l$ until a branch- or endpoint is found */
        while x ∉ $Branch(\Sigma)$ ∪ $Endpoints(\Sigma)$ do
          l ← l ∪ {(|l|,x)}
          visited ← visited ∪ {x}
          /* x can only have one unvisited neighbor */
          {succ} ← $\mathcal{N}_{8_\Sigma}$(x) \ visited
          x ← succ
        end while

        /* add the endpoint */
        l ← l ∪ {(|l|, x)}
        /* add the constructed line to the set of lines */
        lineCollection ← lineCollection ∪ {l}
      end for
    end for

    return lineCollection
}
```

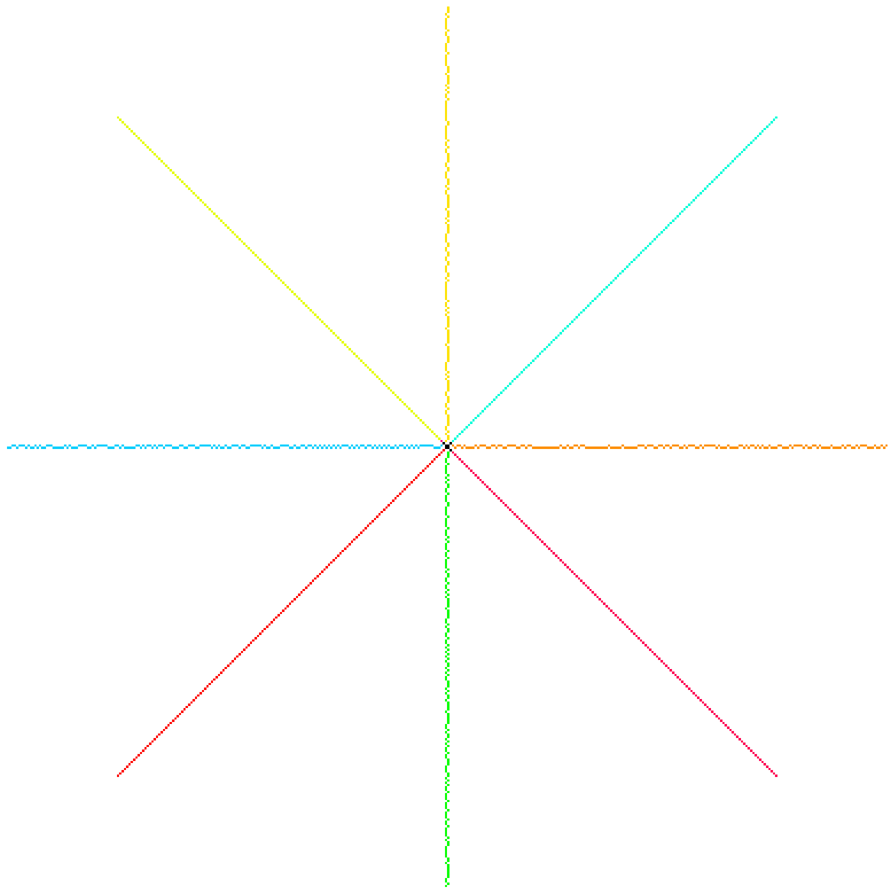


Figure 4.1: The figure shows the thinned skeleton of Figure 3.3a. The decomposition into lines is visualized by colors. The branching points remain black for more clarity, although they might be contained in some lines.

Refer to Figure 4.1 for an example result of the line segmentation algorithm. All pixels of the same color are assigned to the same line.

4.2 Skeleton Pruning

To keep the pruning pseudocode from Algorithm 4.3 simple, we use

$$\mathcal{I}(l) := \left|\left\{p \;\middle|\; \big(p \in R(\Sigma) \land p \notin R(\Sigma \setminus l)\big) \lor \big(p \notin R(\Sigma) \land p \in R(\Sigma \setminus l)\big)\right\}\right| \qquad \forall l \in \mathit{Line}(\Sigma)$$

as the importance measurement. $\mathcal{I}(l)$ gives the amount of pixels that the skeleton line $l \in \mathit{Line}(\Sigma)$ contributes to the reconstructed object $R(\Sigma)$.

Algorithm 4.3 Pseudocode of the pruning algorithm. Given a skeleton Σ, it removes skeleton lines if they fall below a given treshold p and returns the pruned skeleton.

```
procedure pruning(Σ, p) {
    $\Sigma_{pruned}$ ← Σ
    lines ← segmentation(Σ)        /* set of lines */

    for l ∈ lines do
      if $I(l)$ ≤ p then
        /* remove l if the importance measure falls below $p$ */
        for point ∈ l do
          $\Sigma_{pruned}$ ← $\Sigma_{pruned}$ \ {point}
        end for
      end if
    end for

    return $\Sigma_{pruned}$
}
```

4.3 Branch Smoothing

As described in Subsection 3.1.2, we want to smooth lines horizontally and vertically. To this end, we need to identify candidates for smoothing. In the following, we develop a method for horizontal smoothing. The derivation for smoothing vertically is analog.

Assume we want to check for line $l \in \mathit{Line}(\Sigma)$ if it is a candidate for smoothing in x-direction, i. e., if it propagates clearly straight from left to right or vice versa without any sharp curve or loop such that an x-coordinate would occur twice.

$$\text{horizontal line} \Leftrightarrow \big(\max_{x\in\mathbb{N}}\{x \mid (x,y) \in l\} - \min_{x\in\mathbb{N}}\{x \mid (x,y) \in l\} + 1\big) = |l|$$

Next, we need to find a y-coordinate $yOpt$ that is located as close as possible to the optimal line. Therefore, we do a majority vote on y-coordinates, meaning we count the absolute occurrence for each y-coordinate. If there are several y-coordinates having the same highest number of occurrences, we chose the median:

$$yOpt := median(\operatorname*{arg\,max}_{y\in\mathbb{N}} |\{(x,y) \in l\}|)$$

where

$$\operatorname*{arg\,max}_{x\in S\subseteq\mathbb{N}} f(x) := \big\{x \;\big|\; x \in S \wedge \forall y \in S\colon f(y) \leq f(x)\big\}$$

and

$$median(\{s_1,\ldots,s_n\}) := s_{\lfloor \frac{n}{2} \rfloor} \qquad \text{with } s_1 < s_2 < \ldots < s_n.$$

To decide if a line can be indeed smoothed in x-direction, we have to check if it is a horizontal line and if its pixels' y-coordinates do not deviate more than ϵ pixels from the near-optimal y-coordinate *yOpt*. ϵ is the allowed deviation from the optimal line that can be tuned individually. The aforementioned condition is expressed as:

$$\begin{aligned} cond_h(l) := &\big(\max_{x\in\mathbb{N}}\{x \mid (x,y)\in l\} - \min_{x\in\mathbb{N}}\{x \mid (x,y)\in l\} + 1\big) = |l| \\ &\wedge \big(\max_{y\in\mathbb{N}}\{y \mid (x,y)\in l\} - yOpt\big) \leq \epsilon \\ &\wedge \big(yOpt - \min_{y\in\mathbb{N}}\{y \mid (x,y)\in l\}\big) \leq \epsilon \end{aligned}$$

If $cond_h(l)$ is fulfilled, we can smooth the line l horizontally by setting the y-coordinates of each point $p \in l$ to *yOpt*. $smooth_h(l)$ represents the horizontally smoothed line:

$$smooth_h(l) := \{(x, yOpt) \mid (x,\_) \in l\}$$

Analogously, we define the smoothing in vertical direction. The vertical line is described as:

$$\text{vertical line} \Leftrightarrow \big\{\max_{y\in\mathbb{N}}\{y \mid (x,y)\in l\} - \min_{y\in\mathbb{N}}\{y \mid (x,y)\in l\} + 1\big\} = |l|$$

We compute the optimal x-coordinate with a majority voting:

$$xOpt := median\big(\arg\max_{x\in\mathbb{N}} |\{(x,y)\in l\}|\big)$$

And the condition whether a line can be smoothed in vertical direction is:

$$\begin{aligned} cond_v(l) := &\big(\max_{y\in\mathbb{N}}\{y \mid (x,y)\in l\} - \min_{y\in\mathbb{N}}\{y \mid (x,y)\in l\} + 1\big) = |l| \\ &\wedge \big(\max_{x\in\mathbb{N}}\{x \mid (x,y)\in l\} - xOpt\big) \leq \epsilon \\ &\wedge \big(xOpt - \min_{x\in\mathbb{N}}\{x \mid (x,y)\in l\}\big) \leq \epsilon \end{aligned}$$

This yields the vertical smoothing function in y-direction:

$$smooth_v(l) := \{(xOpt, y) \mid (\_, y) \in l\}$$

Combining these functions, a line $l \in Line(\Sigma_{pruned})$ is preprocessed via:

$$S(l) = \begin{cases} smooth_h(l), & \text{if } cond_h(l) \\ smooth_v(l), & \text{if } cond_v(l) \\ l, & \text{otherwise.} \end{cases}$$

The function $S(l)$, $smooth_h(l)$, $smooth_v(l)$, $cond_h(l)$, and $cond_v(l)$ are translated to pseudocode in Algorithm 4.4. The functions *median*, *max*, *min*, *arg max*, and *arg min*

Algorithm 4.4 Pseudocode for the smoothing algorithm. Given the skeleton Σ, it implements the function $S(l)$ and returns the smoothed skeleton.

```
procedure smoothing($\Sigma$, $\epsilon$) {
    smoothH, smoothV ← true   /* smoothing flags */
    lines ← segmentation($\Sigma$)   /* set of lines */
    /* smooth each line separately */
    for l ∈ lines do
      /* lines with two or less points shall not be smoothed */
      if |l| ≤ 2 then continue end if
      /* if the difference between the coordinates is too large,
         smoothing is not possible */
      (minX,maxX) ← ( $\min_{0 \le i \le |l|-1}$ {$line_l$(i).x}, $\max_{0 \le i \le |l|-1}$ {$line_l$(i).x})
      (minY,maxY) ← ( $\min_{0 \le i \le |l|-1}$ {$line_l$(i).y}, $\max_{0 \le i \le |l|-1}$ {$line_l$(i).y})
      if maxX - minX > 2 * $\epsilon$ ∧ maxY - minY > 2 * $\epsilon$ then
        continue
      end if
      smoothH ← (maxX - minX + 1) = |l|
      smoothV ← (maxY - minY + 1) = |l|

      /* smoothing not possible */
      if (smoothH ∧ smoothV) ∨ (¬smoothH ∧ ¬smoothV) then
        continue
      end if

      /* smooth horizontally */
      if smoothH then
        yOpt ← $median(\arg\max_{y \in \mathbb{N}} |\{(x,y) \in \mathbb{N}\}|)$
        /* check if y-coordinates and yOpt differ by less than $\epsilon$ */
        for p ∈ l do
          if |p.y - yOpt| > $\epsilon$ then continue with next line end if
        end for

        for p ∈ l do
          p.y ← yOpt         /* $smooth_h$(l) */
        end for
      else
        /* smooth vertically */
        xOpt ← $median(\arg\max_{x \in \mathbb{N}} |\{(x,y) \in \mathbb{N}\}|)$
        /* check if x-coordinates and xOpt differ by less than $\epsilon$ */
        for p ∈ l do
          if |p.x - xOpt| > $\epsilon$ then continue with next line end if
        end for

        for p ∈ l do
          p.x ← xOpt         /* $smooth_v$(l) */
        end for
      end if
    end for
    /* return the smoothed skeleton */
    return $\Sigma$
}
```

are used as usual. In the worst case, the perfect medial skeleton line would be located in between the points of the grid, resulting in a discrete skeleton line of width two surrounding the optimal line. Thus, a natural choice for the allowed maximal derivation from the optimal line is $\epsilon = 1$. In the implementation of MAT compression, ϵ is set to one.

A drawback of Algorithm 4.4 with $\epsilon = 1$ is illustrated in Figure 4.2. The object in Figure 4.2a is synthetically designed such that it is identical to its thinned skeleton. The effect of line smoothing is depicted in Figure 4.2c. We observe that the resulting skeleton is not thin anymore as an additional point below the original branching point is added. The reason is that all three lines that are visualized in Figure 4.2b have the black marked branching point in common. After doing a majority vote for the blue and yellow line, we see that the optimal y-coordinate *yOpt* is located one pixel below the branching point. Thus, the y-coordinates of those points differing from the optimal line are set to *yOpt*, i. e., for the yellow and blue lines the position of its branching point is changed. In contrast to the branching points of these yellow and blue lines, the position of the branching point with respect to the pink line remains unchanged. The side effect is that new branching points arise when decomposing the smoothed skeleton again, as we see in Figure 4.2d. The harm that is done by this is not large as we must only store one additional point. At the same time, the pink and red lines are shortened by one.

4.4 MAF Approximation

In this section, we describe how Kolesnikov's two-step fast near-optimal approximation algorithm is implemented [KF02].

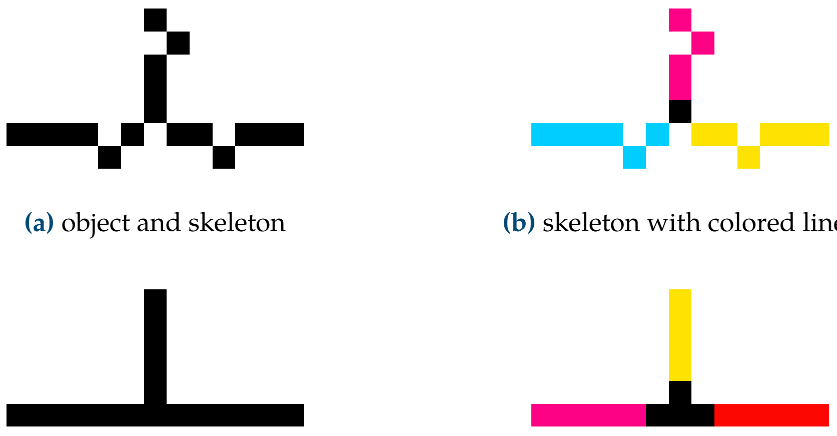
Finding the Reference Solution. Schröder et al. [SL99] define a recursive function $R\colon l \longrightarrow \mathbb{N}$ that computes the least number of segments needed to connect p_1 and any other point p_j of the line, i. e.,

$$R(j) := \max\big\{0, \min_{0<i<j}\{R(i) + 1 \mid d(i,j) \leq a\}\big\}$$

such that the maximal local distortion $d(i,j)$ between the approximated radius and the actual one does not exceed the user-defined error tolerance parameter a. Thus, we define $d(i,j)$ as an L_∞-norm

$$d(i,j) := \max_{i \leq k \leq j} \left| \left(f(p_i) + \frac{f(p_j) - f(p_i)}{j-i} \cdot (k-i) \right) - f(p_k) \right|.$$

$B(n)$ gives the number M of needed segments for approximating the line $l = \{(0, f(p_0)), \ldots, (n-1, f(p_{n-1}))\}$. To this end, we store the index of the



(a) object and skeleton

(b) skeleton with colored lines

(c) smoothed skeleton

(d) smoothed skeleton with colored lines

Figure 4.2: Figure 4.2a is a synthetically designed object. Its skeleton is identical to the object itself. Figure 4.2b shows the skeleton decomposed into lines. Each line is colored differently. The branching point in the middle remains black for clarity. The result of line smoothing is depicted in Figure 4.2c. Analogously to Figure 4.2b, the lines of the smoothed skeleton are colored in Figure 4.2d. The color choice has no meaning as the algorithm colors the lines automatically.

predecessor vertex p_i of a point p_j giving the minimal number of segments in a function $B : \mathbb{N} \to \mathbb{N}$ with

$$B(j) := \arg\min_{0 \le i < j} \{R(i) + 1 \mid d(i, j) \le a\}.$$

$R(i)$ and $B(i)$ are computed in Algorithm 4.5 which we name A1 in the following. The loop conditions are adapted as Schröder et al. approximate a curve with $n + 1$ points starting at index zero whereas we are given a curve with n points starting at one. Additionally, in line eleven, an error in the paper was corrected: Instead of $i = j - i$, the index i must start at $i = j - 1$.

Optimizing the Reference Solution. According to Subsubsection 3.2.1, we interpret a path from $(1, 0)$ to (n, M) through the search space Ω as approximation of a curve starting in p_1 and ending at p_n consisting of M line segments. Therefore, the reference solution from the previous step can be expressed in terms of a reference path $G = \{(G(m), m) \mid 0 \le m \le M\}$ as defined by Schröder et al. We can extract G from B computed in A1 by recursive backtracking:

$$G(i) = B(k_i) \quad \text{with} \quad k_M = n,\ k_{i-1} = B(k_m).$$

Algorithm 4.5 Algorithm A1 from Kolesnikov et al. approximating the MAF for a line l with an approximation error threshold a. It returns the functions R and B.

```
procedure a1(l, a) {
    /* initialize arrays */
    R(0),R(1),B(0) ← 0
    B(1) ← 1
    /* Implement R(j) and B(j) as defined above. The loop
       conditions are changed as Schroeder et al. approximate
       a curve with $n+1$ indexed from 0 to $n$ whereas
       we approximate a curve indexed from 0 to $n-1$. */
    for j ∈ {2,...,|l|} do
      R(j) ← ∞
      for i ∈ {j-1,...,1}
        /* Stop searching if the error gets too large. Decrease
           error arguments by one due to index offset w.r.t.
           original paper. */
        if d(i-1, j-1) > 2 * a then
          break
        end if
        if d(i-1, j-1) ≤ treshold ∧ R(i) + 1 < R(j) then
          /* set number of segments */
          R(j) ← R(i) + 1
          /* set parent vertex of j */
          B(j) ← i
        end if
      end for
    end for
    return (R,B)
}
```

The authors make the invention of the bounded corridor in the search space Ω concrete by indroducing the left bounding function

$$L(m) := \begin{cases} m+1, & \text{if } m \in \{0, ..., B_1\} \\ \max\{m+1, G(m-B_1)\}, & \text{otherwise} \end{cases}$$

and the right bounding function

$$R(m) := \begin{cases} \min\{n, G(m+B_2-1)\}, & \text{if } m \in \{0, ..., M-B_2\} \\ n, & \text{otherwise} \end{cases}$$

with $B_1 := \lfloor \frac{W}{2} \rfloor$, $B_2 := W - B_1$, and n as the length the line. W is the corridor width that can be chosen freely. The authors propose to choose $W \in \{6, 7, 8\}$. Thus, we fix $W = 6$ and $B_1 = B_2 = 3$. The offsets of the lower boundary relative to the boundary bottom are given by the function

$$\Delta m(i) := \begin{cases} 0, & \text{if } i \in \{1, ...R(0)\} \\ \max\{0, m-B_1\} & \text{if } i \in \{R(m-1)+1, ..., R(m)\} \text{ for } m \in \{1, ..., M\}. \end{cases}$$

Within this bounded corridor, dynamic programming is performed in order to minimize the error cost function $E(i, m) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ for the path from $(1, 0)$ to (n, M). In contrast to the algorithm A1, Kolesnikov et al. propose another additive error metric L_2 for the error measurement $e^2(i, j)$ when approximating a discrete curve with M segments:

$$e^2(i, j) := \sum_{k=i+1}^{j-1} \left| \Big(\big(f(k) - \frac{f(j) - f(i)}{j-i} * k - f(i)\big)^2 \Big) / \Big(1 + (\frac{f(j) - f(i)}{j-i})^2\Big) \right|$$

Algorithm 4.6 implements the described procedure.

4.5 Skeleton Compression

The skeleton compression in Algorithm 4.7 consists of the transformation of the skeleton lines into the encoding scheme from Subsection 3.2.2 followed by compression with LPAQ. To keep the pseudocode short and readable, we assume that the following methods are given:

- `encodeLpaq(<File>)`: Compress the given file with LPAQ and return the resulting file.
- `write(<Data>)`: Write the given data in a text file.
- `relativeDir(<Absolute>,<Absolute>)`: Given two subsequent absolute directions encoded as in Figure 3.5b, the method returns the relative direction in which the view is turned according to Figure 3.5a.

Algorithm 4.6 Algorithm A2 from Kolesnikov et al. optimizing the MAF approximation for a line l. It returns the function H giving the optimal solution for an approximation with M segments. The function B was computed in A1.

```
procedure a2(l, M, B) {
    /* initialize the error cost funtion */
    E(1,0) ← 0
    for n ∈ {2,...,|l|} do
      E(n,0) ← ∞
    end for

    /* extract the reference path as defined above */
    G(0) ← 1
    G(M),$k_m$ ← |l|
    for m ∈ {M-1,...,1} do
      G(m) ← B($k_m$)
      $k_m$ ← B($k_m$)
    end for

    /* perform dynamic programming */
    for m ∈ {1,...,M} do
      for n ∈ {$L$(m),...,$R$(m)} do
        $C_{min}$ ← ∞
        for j ∈ {$L$(m-1),...,$R$(m-1)} do
        /* Decrease the arguments of $e^2$ by one as we use
           indices from 0 to $n-1$ whereas Kolesnikov uses
           indices from 1 to $n$. */
          C ← E(j, (m-1) % 2) + $e^2$(j-1, n-1)
          if C < $C_{min}$ then
            $C_{min}$ ← C
            $j_{min}$ ← j
          end if
        end for
        E(n,m % 2) ← $C_{min}$
        A(n, m - Δm(n)) ← $j_{min}$
      end for
    end for

    /* backtrack the cheapest path */
    H(M) ← N
    for m ∈ {M,...,1} do
      H(m-1) ← A(H(m), m - Δm(H(m)))
    end for
    return H
}
```

- `absoluteDir(<Point>,<Point>)`: Given two subsequent points, the absolute propagation direction is returned.

4.6 MAT Compression

In this section, we combine the algorithms defined above to obtain the complete MAT compression algorithm. We assume we are given an image *img* and thin it with a thinning function *thin*(*img*) representing FOD, FOT, FOA, FMDT, or MDT. Afterwards, we prune and smooth the skeleton. We then pick the points for approximation $\mathcal{I}(l) \subseteq l$ for all $l \in \mathit{Line}(\Sigma_{smoothed})$ according to Kolesnikov's fast near-optimal algorithm. For all points p that are not picked for approximation, the MAF resets its radius, i. e.,

$$f(p) = 0 \quad \text{iff} \quad p \in \Sigma \setminus \Big(\bigcup_{l \in \mathit{Line}(\Sigma)} \mathcal{I}(l) \Big).$$

Finally, the remaining skeleton and the MAF f are compressed as described in Section 4.5. The pseudocode is depicted in Algorithm 4.8.

4.7 Decoding

For decoding, we need to decompress the file that was compressed with LPAQ. Next, we backtransform the data from the chain code representation described in Subsection 3.2.2 into the skeleton Σ. Hence, we need to decode the chain codes into points of the skeleton. In order to keep the pseudocode in Algorithm 4.9 clear and readable, the pseudocode of the chain decoding is shortened. Therefore, we use the following methods:

- `decodeLpaq(<File>)`: Decode the given file with LPAQ and return the decoded version.
- `read()`: Return the next word from the linked text file.
- `decodeAbs(<Absolute>,<Relative>)`: Given the last decoded absolute direction *abs* and a new relative direction *rel*, the method returns the new absolute direction , i. e., the absolute direction after turning from *abs* in *dir* direction.
- `decode(<Point>,<Absolute>)`: Given the last computed point and the recent absolute direction, the method returns the new x- and y-coordinates when moving one pixel in the given direction starting from the given point.

Algorithm 4.7 Implementation of the encoding scheme from Subsection 3.2.2. Given a set of skeleton lines Σ, return the LPAQ encoded file.

```
procedure chaincode(lines) {
    /* encoded is an initially empy text */
    encoded ← write(xDim, yDim)  /* write information in file */

    /* encode linewise */
    for l ∈ lines ∧ |l| > 1 do
      /* encode the initial absolute direction */
      prevAbs ← absoluteDir($line_l$(0), $line_l$(1))
      encoded ← write(prevAbs)

      /* add relative directions */
      for i ∈ {2,...,|l|-1} do
        newAbs ← absoluteDir($line_l$(i-1), $line_l$(i))
        newRel ← relativeDir(prevAbs, newAbs)
        encoded ← write(newRel)
        prevAbs ← newAbs
      end for
      encoded ← write(EOS)
      /* save last radius */
      encoded ← write(f($line_l$(|l|-1)))
    end for

    encoded ← write(EOCC)

    /* encode points having a radius larger than zero */
    for l ∈ lines ∧ |l| > 1 do
      for i ∈ {1,...,|l|-2} do
        if f($line_l$(i)) > 0 then
          encoded ← write($line_l$(i).x, $line_l$(i).y, f($line_l$(i)))
        end if
      end for
    end for

    /* encode the lines consisting of a single point */
    for l ∈ lines ∧ |l| ≤ 1 do
      encoded ← write($line_l$(1).x, $line_l$(1).y, f($line_l$(1)))
    end for

    return encodeLpaq(encoded)
}
```

Algorithm 4.8 According to Section 4.6, the compression algorithm returns the compressed file given the input image *img*, the pruning threshold *p*, and the approximation threshold *a*.

```
procedure compress(img, p, a) {
    /* preprocess the skeleton */
    $\Sigma$ ← thin(img)
    $\Sigma_{pruned}$ ← pruning($\Sigma$, p)
    $\Sigma_{smoothed}$ ← smoothing($\Sigma_{pruned}$, 1)

    /* decompose the skeleton into lines */
    lines ← segmentation($\Sigma_{smoothed}$) /* set of lines */

    /* find the points for interpolation */
    for l ∈ lines do
      /* R and B are the functions defined for A1 */
      (R,B) ← a1(l, a)
      M ← R(|l|)
      /* H is the optimal reference path from A2. */
      H ← a2(l ,M, B)
      /* Decrease the index given by H(i) by one as we use
         indices from zero to $n-1$ whereas Kolesnikov et al.
         use points with indices starting at one. */
      interpolants ← {$line_l$(H(i)-1) | i ∈ {1,...,M}}
      /* reset the MAF for points not chosen for approximation */
      for point ∈ l do
        if point ∉ interpolants then
          f(point) ← ∅
        end if
      end for
    end for

    /* encode the remaining skeleton and the MAF */
    encoded ← chaincode($\Sigma_{smoothed}$)
    return encoded
}
```

Algorithm 4.9 Decode a given encoded file *encoded* and return the reconstructed image according to the steps mentioned in Section 3.3.

```
procedure compress(encoded) {
    f, Σ ← ∅  /* MAF and skeleton initialization */
    /* Step 1: decode the LPAQ encoded file */
    decoded ← decodeLpaq(encoded)
    /* start reading from file */
    (xDim,yDim) ← (read(), read())
    while not reached EOF do
      /* Step 2: decode a line */
      if next word is not EOCC then
        (x,y) ← (read(), read())   /* get x- and y-coordinate */
        f(x,y) ← read()            /* assign radius */
        Σ ←  Σ ∪ {(x,y)}
        initAbs ← read()    /* get initial absolute direction */
        (x,y) ← decode((x,y), initAbs) /* decode second point */
        Σ ← Σ ∪ {(x,y)}
        /* get next points depending on relative direction */
        abs ← initAbs
        while next word is not EOS do
          relAbs ← read()
          abs ← decodeAbs(abs, rel)
          (x,y) ← decode((x,y), abs)
          Σ ← Σ ∪ {(x,y)}
        end while
        f(x,y) ← read()                     /* get last radius */
      else
        /* Step 3: decode the single points and MAF mappings */
        while not reached EOF decoded do
          (x,y) ← (read(), read())
          f(x,y) ← read()
        end while
      end if
    end while
    /* Step 4: approximate the missing MAF information */
    lines ← segmentation(Σ)
    for line ∈ lines do
      k ← 1
      while k < |line| do
        l ← k + 1
        while f($line_{line}$(l)) = 0 ∧ l < |line| do
          l ← l + 1
        end while
        /* approximate missing radii */
        for i ∈ {k+1,...,l-1} do
          radius ← f($line_{line}$(k)) + (i - k)
                   * (f($line_{line}$(l)) - f($line_{line}$(k)))/(l - k)
          f($line_{line}$(i)) ← radius
        end for
      end while
    end for
    /* Step 5: reconstruction */
    return reconstruct(Σ, xDim, yDim)
}
```

4.8 Sparsification

For the sparsification, we need to delete a given percentage p of points from the skeleton. Those points are chosen such that they have the least possible impact on the exactness of reconstruction. The point with least impact can be computed as

$$\Sigma_{sparse} := \Sigma \setminus \left\{ \operatorname*{arg\,min}_{point \in \Sigma} \Big| |R(\Sigma)| - |R(\Sigma \setminus \{point\})| \Big| \right\}.$$

Repeating this procedure $\lfloor p \cdot |\Sigma| \rfloor$ times, we remove step by step the points with the least impact from the remaining skeleton Σ_{sparse}. This is done in Algorithm 4.10.

Algorithm 4.10 Sparsify the skeleton Σ, i. e., delete a given percentage of skeleton points and return the sparsified skeleton Σ.

```
procedure sparsify(Σ, percentage) {
    /* compute how many points need to be deleted */
    n ← ⌊percentage * |Σ|⌋

    /* repeat the sparsification n times */
    for i ∈ {1,...,n} do
      min ← ∞
      /* find the point with the least influence */
      for point ∈ Σ
        err ← ||reconstruct(Σ)| - |reconstruct(Σ \ {point})||
        if err < min then
          min ← err
          delete ← point
        end if
      end for

      /* delete the point with the least influence */
      Σ ← Σ \ {delete}
    end for

    return Σ
}
```

5 Evaluation and Discussion

In this chapter, we evaluate the performance of MAT compression. The following three criteria help us to identify a good compression standard:

- Compression ratio: Clearly, the compression ratio plays an important role. In general, the higher it is, the better we consider the performance of the algorithm.
- Error measurement: As MAT compression is near-lossless, we test the quality of the decompressed image. A small difference to the original image indicates a reasonable compression algorithm.
- Runtime: To conclude whether the algorithm is suitable for real-life applications, its runtime has to be analyzed as well. The lower the runtime is, the more suitable is the compression standard.

As MAT compression depends on three user-defined input parameters, we tested the impact of each parameter separately on each of these criteria. This will allows us to formulate a general trend about which parameter combinations yield a reasonable trade-off between the three quality measurements, i. e., which combination gives us a stable compression algorithm.

5.1 Testing Environment

This section covers the description of the system as well as the choice of test images and the test design.

System Details. The tests were executed on a 32 bit VirtualBox running Ubuntu 16.04 using an Intel Core$^{\text{TM}}$ i7-3635QM processor with 2.40 GHz and 1 GB of RAM.

MAT compression was written in C99 just as the modified Huffman encoding that we needed for the comparative experiments. For the experiments with JBIG, JPEG, and JPEG-2000, we used a command-line tool named *GraphicsMagick 1.3.34*. All parameters remained at default settings.

Eight out of ten test images were taken from the MPEG7 Core Experiment database for shape descriptors (CE-Shape-1) that consists of 70 shape classes, each having 20 members. As the database images are binary images in a Graphics Interchange Format (GIF) but the compression algorithm expects them in a Portable Grey Map (PGM) format, they were converted before usage. The PGM values range from 0 to 255 where value 0 represents black and 255 white. We inverted the image colors before usage because the CE-Shape-1 images depict white objects with a black background but our compression algorithm expects an inverse coloring.

Test Data. Figure 5.1 shows the images that we used in the experiments. Figure 5.1a–Figure 5.1h come from the MPEG7 Core Experiment database. Figure 5.1i is a synthetically designed image and Figure 5.1j was taken from Peter [Pet10]. We divide the images into two categories:

- Geometric images: This image type depicts objects that are composed of geometric figures and have a symmetric structure. Their skeletons are simple and might therefore be suitable for compression.
- Complex images: Those images contain objects with a non-symmetric corpus and with detailed object boundaries. Thus, the resulting skeletons have complex structures. We will evaluated if these complex structures are suitable for MAT compression.

Figure 5.1a–Figure 5.1e were chosen as complex test images. Figure 5.1a is interesting as the object's body mainly consists of two large circle-like parts, and we therefore suppose that MAT compression should perform well on it. Figure 5.1b has a very complex object structure because of its detailed boundary and because it is composed of many components of different length and width. In contrast to this, Figure 5.1c is challenging as it contains many holes inside the object which yield a complex skeleton. The image helps us to analyze if MAT compression preserves homotopy. Similarly to Figure 5.1b, Figure 5.1d has also a detailed boundary that is essential for the object. In the best case, these details should be preserved during compression. Together with Figure 5.1a, Figure 5.1e might be the easiest amongst the complex images as there are not many details on the boundaries. As there is only one large hole inside the object, its skeleton is less branched but still complex.

Figure 5.1f–Figure 5.1j are geometric test images. The ability of MAT compression to reproduce sharp tips in combination with a smooth boundary is tested with Figure 5.1f. In addition to this, we examine the ability to reproduce curved shapes with Figure 5.1g. Its skeleton especially benefits from line smoothing during preprocessing as it consists of several long and shaky lines (see Figure 3.3d). In contrast to Figure 5.1g, the challenge of Figure 5.1h is the sufficiently well reproduction of sharp corners and notches. Figure 5.1i demonstrates that MAT compression algorithm also works on a closed skeleton which is a special case as it does not have any end or junction points. Figure 5.1j includes five classical symmetric objects and is therefore meaningful for evaluating the performance of MAT compression with respect to simple geometric structures.

Parameter Choice. The MAT compression algorithm offers many possibilities to tune its performance. There are three important parameters that the user can influence when executing the program:

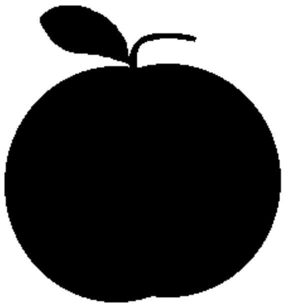
- Thinning method: Peter gives different methods of skeleton thinning that we introduced in Subsection 2.2.1, namely FMDT, MDT, FOA, FOD, and FOT. As thinning is not unique, these methods can yield different skeletons for the same object. Therefore, the choice of a thinning method might influence the performance of MAT compression.
- Pruning parameter p: p is used during skeleton pruning as described in Section 4.2. During this process, the algorithm removes lines that contribute less or equal than p pixels to the original object. As large p will result in a compression that loses too many details and $p = 0$ would not delete any lines at all, we limit the choice of the parameter p to $p \in \{1, 5, 10\}$.
- Approximation error threshold a: a is needed for the MAF approximation as described in Section 4.4. It influences the exactness of reconstruction, i. e., the smaller a is, the more exact is the reconstruction. The selection of the parameter a is limited to $a \in \{0.5, 1, 1.5, 2\}$ as proposed by Kolesnikov et al. Parameter choices exceeding those limits will result in a too lossy or in a complete lossless compression, respectively.

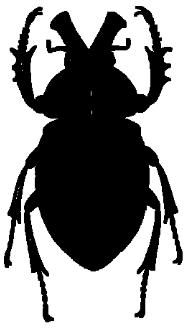
Thus, we examine 60 different parameter combinations in total for each of the input images.

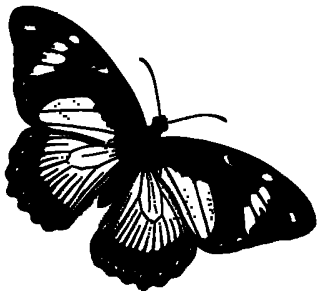
Compression Ratio Tests. For each test image, the compression ratio c for each of the 60 parameter combinations is computed by

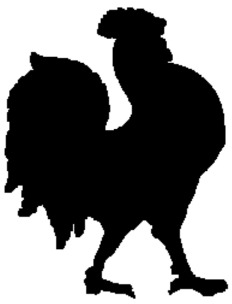
$$c := \frac{\text{size of original file (byte)}}{\text{size of compressed file (byte)}}$$

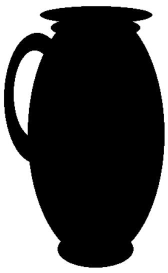
and afterwards rounded to integers. The C function *get_size_of_file(<file>)* was used to determine the file sizes. Furthermore, we compute the average compression ratio for each thinning method on the remaining twelve parameter combinations, i. e., the

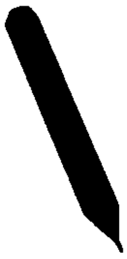

(a) `apple-3` (225x241)

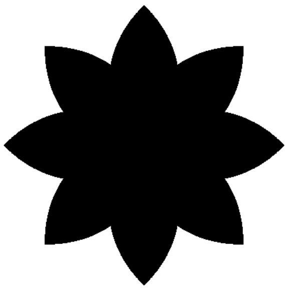

(b) `beetle-4` (264x448)

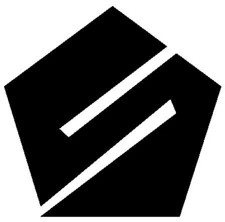

(c) `butterfly-1` (505x463)

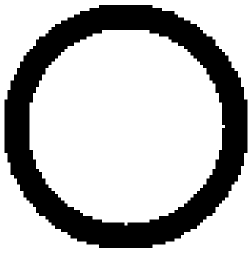

(d) `chicken-19` (197x273)


(e) `jar-12` (518x518)


(f) `pencil-17` (245x364)


(g) `device2-14` (412x412)


(h) `device6-12` (564x564)


(i) `donut` (100x100)

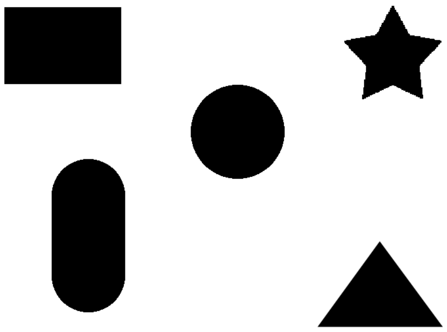

(j) `formen` (800x600)

Figure 5.1: Images used for the experiments. Figure 5.1a–Figure 5.1h are taken from the MPEG7 Core Experiment database for shape descriptors (CE-Shape-1). Figure 5.1i is a synthetically designed image and Figure 5.1j is taken from Peter [Pet10]. Beside the image names, we also give their dimensions.

combinations of the approximation threshold a and the pruning parameter p. The values are rounded mathematically to integers. This allows to draw conclusions about the performance of each thinning method separately.

The compression ratios for the comparative approaches with JBIG, JPEG, JPEG-2000, and modified Huffman encoding were computed the same way.

Quality Tests. Although a high compression ratio is an indicator of a good compression algorithm, the quality might suffer from too strong compression. Hence, we compute the relative number of erroneous pixels. To this end, we use the error measure e from Section 3.4. The smaller e is, the more accurate is the decompressed image. As in the previous paragraph, for all test images and thinning methods, the relative error of the twelve parameter combinations of a and p are computed and averaged to two decimals.

As JBIG and modified Huffman encoding are lossless compression codecs, we only computed the relative errors for JPEG, and JPEG-2000.

Runtime Tests. Since the runtime is an important quality criterion for a good applicable compression algorithm, the runtime of MAT compression is measured with the help of the C function *clock()* for all test scenarios and rounded to one decimal. Again, we average the compression time for each of the five thinning methods. The values are again rounded to one decimal place.

The runtimes of the comparative experiments with JBIG, JPEG, JPEG-2000, and modified Huffman encoding were measured with *clock()* as well.

5.2 Experiments

In this section, the results of the experiments are presented. We discuss and interpret them in Section 5.3.

To reduce the amount of collected data, we provide averaged results of the compression ratio, the relative error, and the runtime measurement for all images of the test set in Table 5.4, Table 5.5, and Table 5.6, respectively. To this end, we only average the outcomes that result from the same thinning method such that we can evaluate more easily if FMDT, MDT, FOD, FOT, and FOA are suitable for MAT compression. More explicitly, this means that for a fixed method, the outcomes of the twelve possible parameter combinations of a and p are averaged. In addition to this, Appendix B presents explicit results of all parameter combinations and quality criteria for Figure 5.1a, Figure 5.1b, Figure 5.1c, Figure 5.1h, and Figure 5.1j. We refer to them for a more detailed parameter analysis.

		Compression Method			
	Test Image	JPEG	JPEG-2000	Modified Huffman	JBIG
complex	`apple-3`	12	14	61	175
	`beetle-4`	8	15	34	148
	`butterfly-1`	7	16	36	127
	`chicken-19`	9	14	43	138
	`jar-12`	32	26	136	488
geometric	`pencil-17`	25	21	145	435
	`device2-14`	20	15	100	396
	`device6-12`	24	20	131	573
	`donut`	5	9	28	43
	`formen`	41	34	145	720

Table 5.1: The compression ratios of JPEG, JPEG-2000, modified Huffman encoding, and JBIG for the test data set.

Table 5.1 and Table 5.3 show the compression ratios and the runtime of JPEG, JPEG-2000, modified Huffman encoding, and JBIG for all images of the test set. We find the relative errors produced by JPEG and JPEG-2000 in Table 5.2. These tables are used for a direct comparison to MAT compression.

5.2.1 Comparative Results

In this section, we compare the performances of JPEG, JPEG-2000, JBIG, and modified Huffman encoding to each other.

Compression Ratio. Table 5.1 reveals that JBIG clearly outperforms JPEG-2000, JPEG, and modified Huffman encoding with respect to its compression rate. Modified Huffman encoding is superior to JPEG and JPEG-2000. Furthermore, for complex test images, JPEG-2000 compresses slightly stronger than JPEG and vice versa for the geometric test set. In contrast to the small differences between the compression ratios of JPEG and JPEG-2000, the ratios of modified Huffman encoding and JBIG are considerably larger.

Relative Error. We can see in Table 5.2 that the relative errors of JPEG-2000 are higher than those from JPEG for the complex data—expect for the image `jar-12`. In contrast to this, JPEG-2000 is rather superior to JPEG with respect to the geometric

	Test Image	Compression Method	
		JPEG	JPEG-2000
complex	apple-3	0.070410	0.222757
	beetle-4	0.114237	0.330416
	butterfly-1	0.120741	0.291637
	chicken-19	0.098195	0.345233
	jar-12	0.024049	0.000988
geometric	pencil-17	0.030063	0.005506
	device2-14	0.039960	0.010740
	device6-12	0.032242	0.000899
	donut	0.174900	0.400900
	formen	0.030110	0.000192

Table 5.2: The relative errors (%) of JPEG, JPEG-2000, modified Huffman encoding, and JBIG for the test data set are collected in this table.

	Test Image	Compression Method			
		JPEG	JPEG-2000	Modified Huffman	JBIG
complex	apple-3	0.000062	0.000057	0.012465	0.000067
	beetle-4	0.000065	0.000079	0.022796	0.000073
	butterfly-1	0.000065	0.000069	0.041892	0.000076
	chicken-19	0.000066	0.000071	0.008007	0.000071
	jar-12	0.000070	0.000070	0.014454	0.000072
geometric	pencil-17	0.000098	0.000069	0.005150	0.000068
	device2-14	0.000081	0.000075	0.012522	0.000083
	device6-12	0.001260	0.000103	0.019535	0.000140
	donut	0.000091	0.000069	0.002466	0.000086
	formen	0.000061	0.000063	0.027261	0.000075

Table 5.3: The compression times (sec) of JPEG, JPEG-2000, modified Huffman encoding, and JBIG for the test data set.

		Thinning Method				
	Test Image	FMDT	FOA	FOD	FOT	MDT
complex	`apple-3`	265	43	283	285	300
complex	`beetle-4`	82	35	84	84	90
complex	`butterfly-1`	60	35	58	58	63
complex	`chicken-19`	83	37	110	110	103
complex	`jar-12`	669	148	690	681	689
geometric	`pencil-17`	655	167	792	789	803
geometric	`device2-14`	429	78	406	497	455
geometric	`device6-12`	852	83	919	918	898
geometric	`donut`	93	92	93	93	101
geometric	`formen`	1537	195	1675	1675	1686

Table 5.4: The average compression ratios for all thinning methods and all test images.

data set. Nevertheless, all relative errors are very small, i. e., yield $< 0.5\%$ of erroneous pixels.

Runtime. Regarding the runtimes depicted in Table 5.3, we see that all methods besides modified Huffman encoding are similarly fast. Nevertheless, also the runtime of modified Huffman encoding is still within some milliseconds.

5.2.2 Compression Ratio Results

Averaged Compression Ratio. The averaged compression ratios for all ten test images with respect to the used five thinning methods FMDT, FOA, FOD, FOT, and MDT are displayed in Table 5.4.

On the geometric data set, MDT and FOT tend to yield the highest average compression ratios. Nevertheless, FOD yields similar results to those of FOT. The performances of these methods are followed by FMDT, and FOA.

Similarly to the geometric data set, MDT is slightly superior to the other methods. But FOD and FOT are comparable to MDT for all tested images. The averaged values for MAT compression using FOA are inferior to the other methods, up to a factor of eleven for `device6-12`.

The highest absolute compression ratio is achieved for `formen`, with some distance followed by `device6-12`.

Explicit Compression Ratio. The explicit experimental data tables from Appendix B reveal that $a = 0.5$ yields the lowest compression ratio among all $a \in \{0.5, 1, 1.5, 2\}$ within each thinning method independent of the choice of p. In particular, p has only a small influence on the absolute compression ratio for $a = 0.5$ whereas the choice of the thinning method influences the competitiveness as we have seen in the last paragraph.

For a fixed thinning method and $a \in \{1, 1.5\}$, the choice of p does not seem to affect the compression ratio. Inversely, for fixed a and increasing p, the compression ratio hardly increases. Only for `formen`, the increasing compression ratios can be observed more easily.

Comparison to Existing Standards. All averaged compression ratios exceed the ratios achieved by JPEG and JPEG-2000.

MAT compression with FOA is strongly inferior to the Huffman approach for the images `apple-3`, `device2-14`, and `device6-12` and clearly superior for `pencil-17`, `donut`, and `formen`. For the other test images, its performance is comparable. Additionally, its performance is almost always much weaker than that of JBIG. The only exception from that is given for `donut`.

MAT compression with FOD, FOT, FMDT, and MDT is, on average, always superior to modified Huffman encoding, independent of the image type. On the geometric test set, our approach even achieves compression ratios that exceed JBIG's performance up to a factor larger than two. In contrast to this, on the complex data set, MAT compression is outperformed by JBIG for the test images `beetle`, `butterfly-1`, and `chicken-19`.

With respect to the explicitly computed compression ratios, the choice of a significantly influences if the chosen compression method can compete with the selected compression standards. Figure 5.2 demonstrates how a suitable choice of parameters influences the competitiveness of MAT compression using FOA. For that, we pick $a = 1$ and $p = 5$. With this parameter configuration, the compression ratio of MAT compression increases above its computed average performance, and competes now with all tested compression standards except for JBIG. In general, the higher a is, the better becomes the performance. Nevertheless, FMDT, FOD, FOT, and MDT outperform JPEG and JPEG-2000 even for $a = 0.5$ and free choice of p on all test data. When comparing the performance of these parameter choices to modified Huffman encoding, we see that they are slightly inferior to this standard except for the test images `apple-3` and `formen`. This changes for larger a. If $a \in \{1, 1.5, 2\}$, $p \in \{1, 5, 10\}$ and FMDT, FOD, FOD, or MDT as thinning method, all compression standards including JBIG are outperformed by MAT compression. Only for `beetle-4` and `butterfly-1` are the results still inferior. This behavior is visualized by means of an example in Figure 5.3 for the parameter configuration $a = 1$, $p = 5$ and the thinning methods MDT, FOD and FMDT. The figure compares the compression ratios only to JBIG as all other compression standards are inferior to our approach.

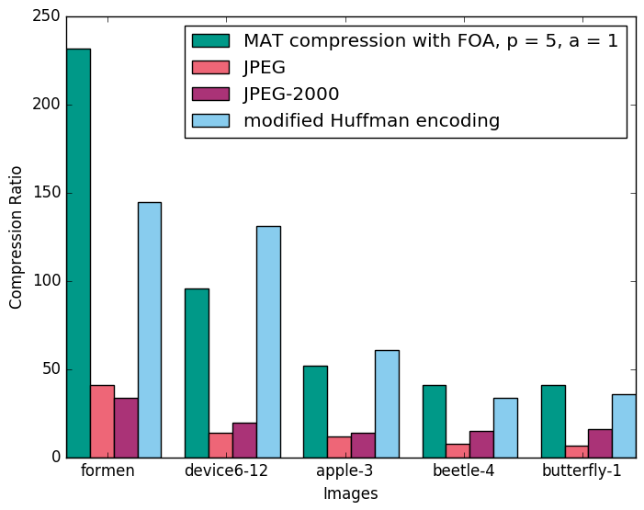


Figure 5.2: Comparison of the compression ratio between MAT compression using FOA, $a = 1$ and $p = 5$ and JPEG, JPEG-2000, and modified Huffman encoding on a subset of the test data. The values are taken from the explicit result tables of Appendix B.

5.2.3 Quality Results

Averaged Relative Error. The averaged relative errors for all ten test images in dependency on the five thinning methods are displayed in Table 5.5.

On the geometric test set, we cannot find a clear tendency which thinning method produces superior results. Nevertheless, FMDT, FOD, and FOT yield similarly low average relative errors. Especially the average results for FOD and FOT are almost always identical.

In contrast to the geometric test set, FOA performs on average slightly better than the remaining methods on the complex set. Again, FOD and FOT perform equally well but are inferior to FMDT and MDT.

The highest average reconstruction errors are achieved for the complex images `beetle-4`, `butterfly-1`, and for the geometric image `donut`. `formen` gives us the lowest average reconstruction errors.

Explicit Relative Error. Given $a \in \{1, 1.5\}$ and p not fixed, the relative error does not change visibly within a chosen thinning method. If $a = 0.5$, the relative error is

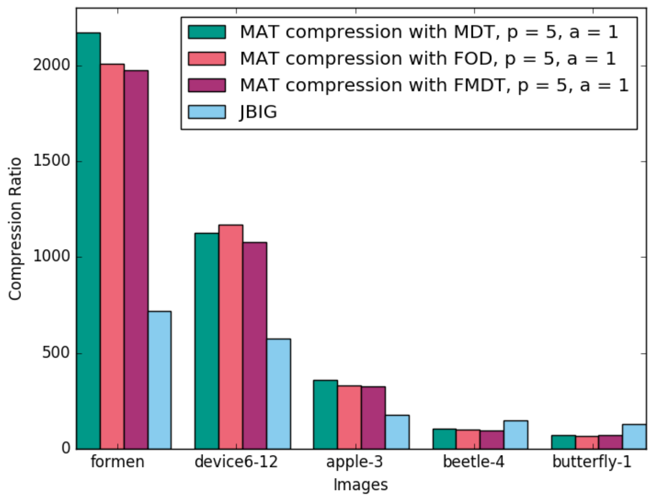


Figure 5.3: The performances of MAT compression using MDT, FOD, and FMDT with $a = 1$ and $p = 5$ are compared to the compression ratio of JBIG on a subset of the test data set. The values for FOD and FMDT are taken from the explicit result tables of Appendix B. MAT compression is superior to JBIG for the images `formen`, `device6-12`, and `apple-3`.

		Thinning Method				
	Test Image	FMDT	FOA	FOD	FOT	MDT
complex	`apple-3`	0.80	0.81	1.09	1.09	0.77
	`beetle-4`	1.47	1.11	1.69	1.69	1.62
	`butterfly-1`	1.67	1.64	1.75	1.77	1.77
	`chicken-19`	1.13	0.76	1.59	1.59	1.42
	`jar-12`	0.24	0.26	0.24	0.24	0.24
geometric	`pencil-17`	0.48	0.29	0.51	0.51	0.55
	`device2-14`	0.52	0.44	0.49	0.56	0.50
	`device6-12`	0.24	0.42	0.39	0.39	0.30
	`donut`	1.37	1.65	1.37	1.37	1.65
	`formen`	0.20	0.22	0.22	0.22	0.20

Table 5.5: The average relative error (in %) for all thinning methods.

		Thinning Method				
	Test Image	FMDT	FOA	FOD	FOT	MDT
complex	`apple-3`	0.4	47.2	0.3	0.3	0.4
	`beetle-4`	7.7	88.0	6.7	6.7	7.5
	`butterfly-1`	16.0	66.6	19.7	17.1	15.5
	`chicken-19`	1.0	7.6	0.6	0.6	0.6
	`jar-12`	4.6	294.0	4.2	4.8	4.6
geometric	`pencil-17`	0.8	0.7	0.9	0.9	1.0
	`device2-14`	4.7	94.9	5.2	4.4	4.7
	`device6-12`	2.5	197.5	3.9	4.6	3.0
	`donut`	0.3	0.2	0.3	0.3	0.3
	`formen`	1.8	62.2	1.5	1.4	1.4

Table 5.6: The average runtimes (sec) for all thinning methods and all test images are collected in this table.

almost always smaller than for $a \in \{1, 1.5, 2\}$, but again, FOA forms an exception. Analogously, if $a = 2$, the relative error tends to be larger than for $a \in \{0.5, 1, 1.5\}$.

Comparison to Existing Standards. As modified Huffman encoding and JBIG are both lossless, they are superior to MAT compression. Furthermore, with respect to the relative error, lossy JPEG and JPEG-2000 outperform our approach as well.

5.2.4 Runtime Results

Averaged Runtime. The averaged runtime for all ten test images in dependency on the five thinning methods FMDT, FOA, FOD, FOT, and MDT are displayed in Table 5.6.

On both data sets, there is no method cleary outperforming the others with respect to their runtime. Only for FOA, the average runtime is an order of magnitude higher than the runtimes achieved by the other methods, i. e., it is up to three minutes slower. In contrast to this, the remaining methods differ at most by four seconds on our test set.

When not using FOA as thinning method, the highest average runtimes are achieved for `beetle-4` and `butterfly-1` which are also the most complex test objects.

Explicit Runtime. The tables from Appendix B reveal that the runtimes for all thinning methods besides FOA are almost always minimal for $a = 0.5$. In this case, the choice of p has no influence on this tendency. For all combinations of $a \in \{1, 1.5, 2\}$ and $p \in \{1, 5, 10\}$ within a thinning method, we found that the runtimes are quite close within each other and in the order of some seconds. For FOA, we cannot clearly identify the runtime behavior.

For the two most complex images `beetle-4`, and `butterfly-1`, the runtimes of MAT compression using FOD, FOT, FMDT, and MDT are significantly higher than for the remaining test images.

Comparison to Existing Standards. As can be seen in Table 5.3, the chosen compression standards are much faster than the fastest outcome of MAT compression.

5.3 Discussion

In this section, we interpret and discuss the observations from Section 5.2. Subsection 5.3.4 combines the results to draw a conclusion about a reasonable choice of parameters yielding a stable compression algorithm. In this context, stable means that our parameter choice does not neglect one of the aforementioned criteria.

5.3.1 Compression Ratio Discussion

We observed that the lowest performance is achieved when choosing $a = 0.5$. This phenomenon is explained by the number of points that we pick during the MAF approximation. For small a, many points within each skeleton line must be selected in order to not exceed the allowed error tolerance threshold. Additionally, those interpolants must be saved separately at the end of the compressed file. For a large number of data points that are chosen for approximation, this creates an overhead resulting in a weak compression ratio. With this knowledge and from the results obtained in the previous section, it is clear that the simpler the skeleton and the object are, the higher becomes the compression ratio. The simplest skeletons are obtained for objects with few details, in particular, if they depict geometric forms such as `formen`. This explains as well why the complex images `beetle-4` and `butterfly-1` yield low performances.

For increasing a, we found that the compression ratio increases slightly. The reason for that is that the approximation algorithm selects less points within a line. Additionally, for increasing pruning parameter p, the preprocessing algorithm potentially removes more lines from the skeleton leading to less data that needs to be stored.

To understand why MAT compression using FOD and FOT produces similar skeletons, we have a look at Table 5.8 and Table 5.11. The first column compares the skeletons resulting from the two thinning methods. It reveals that there is hardly any difference between them. Thus, the skeletons remain similar even after preprocessing. This behavior implicates a similar storing effort as well as nearly identical compression ratios.

The striking bad performance of FOA is explained by the spurious skeleton that the thinning method creates. Such complex skeletons are depicted in the first columns of Table 5.9 and Table 5.12. In comparison to skeletons produced, e. g., by FOD and FOT as depicted in Table 5.8 and Table 5.11, more branches need to be processed and stored. This results in a smaller compression rate.

Surprisingly, `donut` gives a low compression ratio in comparison to the other images although it is a simple and geometric object. This is a phenomenon arising from the image size. As `donut` is the smallest image, the gain of applying a compression algorithm is not as large as if it is applied to a large image such as `formen`. Thus, we suggest that the performance of MAT compression with respect to the compression rate depends on the image size, the object size, the complexity of its skeleton, and is also largely influenced by the choice of a.

5.3.2 Quality Discussion

For $a = 0.5$, we found that the relative error is minimal and enlarges for increasing a. As the relative error is correlated to the amount of information loss of the object, the compression ratio increases together with the error. Thus, we can explain the increasing error measure dependent on a exactly as we explained the behavior of the compression ratio in the last section.

It is to mention that the preprocessing and approximation step might destroy skeleton homotopy. An example of the loss of homotopy can be found in Table 5.7. Therefore, we have a closer look at the cell that depicts the butterfly that was compressed with MDT, $a = 2$ and $p = 10$. A close-up of this example can be seen in Appendix C. When comparing the reconstructed butterfly to the original image, we observe that small holes, e. g., on the bottom right wing are missing. Nevertheless, further quality issues are not visible. In contrast to this, serious quality issues arise for `device6-12` during decompression, as can be seen, e. g., in Table 5.11. Especially the lower right edge of the object suffers as in the corresponding skeletons, a branch representing this corner is missing. This is not an issue resulting from skeletal pruning but from the chosen thinning method itself. Again, this proves that the choice of a thinning method might considerably influence the quality of MAT compression.

5.3.3 Runtime Discussion

In Subsection 5.2.4, we found that the runtime differences of MAT compression using FMDT, MDT, FOD, and FOT are small. This phenomenon is explained as in Subsection 5.3.1, i. e., similar skeletons result in a similar necessary computational effort.

When comparing the skeletons of Table 5.11 to the skeletons resulting from FMDT and MDT depicted in the first column of Table 5.10, one observes that there are two branches that differ. Therefore, also the preprocessing phases of MAT compression with FMDT and MDT are distinct from those with FOD and FOT. Nevertheless, as the absolute number of branches does not differ largely, the runtimes of all four methods are still close within each other.

Within the different parameter combinations, the number of selected points and pruned lines does not vary significantly, and hence, also results in similar runtimes for $a \in \{1, 1.5, 2\}$ and $p \in \{1, 5, 10\}$. For $a = 0.5$, we observed shorter runtimes in Paragraph 5.1—besides for FOA. This trend is explained by the way a line is scanned in order to find interpolants. For this, remember that Kolesnikov's algorithm scans the points of a line consecutively. We start at an endpoint and continue scanning until the approximation error between the straight line from the recent point to the corresponding endpoint and the original curve exceeds twice the allowed approximation parameter. As $a = 0.5$ is a small approximation threshold, it is exceeded fast, i. e., fewer points must be considered during scanning. Therefore, it is a time saver resulting in shorter runtimes. However, MAT compression with FOA is an exception from the phenomenon. The large runtimes result from a very spurious skeleton as depicted by way of example in Table 5.12. Hence, in comparison to skeletons resulting from one of the other thinning methods, the compression algorithm has to process a multiple of branches.

Moreover, the experiments revealed a general increase of runtime for the complex images `beetle-4` and `butterfly-1`. For the latter, corresponding skeletons are provided in Table 5.12, Table 5.8, and Table 5.7. By comparing their number of branches to the number of branches that can be found in a geometric image such as depicted in Table 5.10, Table 5.12, or Table 5.11, we conclude that this large difference explains why complex images have a higher computational effort. Consequently, this means that the more complex the object is, the more time needs MAT compression for its computations.

5.3.4 Parameter Discussion

As discussed before in Subsection 5.3.2, the quality performance behaves inversely proportional to the compression ratio. In particular, the parameter combinations that yield high compression ratios lead at the same time to high error rates. Thus, those parameters are not a good choice for a stable compression algorithm. If we aim to find a reasonable universal parameter choice for MAT compression, this disqualifies all combinations including $a = 2$ and $p = 10$. On the other hand, for very small parameters a and p, we risk pushing the compression ratio down too much. Therefore, we should also eliminate $a = 0.5$ from the possible parameter combinations. As we found that the choice between $a = 1$ and $a = 1.5$ does not make a difference on our test data set, we propose to choose $a = 1$.

For the choice of p, we seem to be a bit more free as $p = 1$ and $p = 5$ do not influence the relative error as well as the compression ratio visibly. If we need to determine a fixed variable assignment, we could just take the median of our predefined interval of p and pick $p = 5$.

Finally, we have to choose a suitable thinning method. Obviously, FOA is not taken into account as it yields unacceptable high runtimes. Among the remaining four methods, none of them clearly dominates over the others when looking at the averaged results for the three quality criteria. Indeed, their performance strongly varies between the two test sets and among the three compression criteria. Figure 5.4 compares the performance of the remaining four thinning methods with respect to their compression ratios and relative errors. The other two parameters were chosen as described above, i. e., $a = 1$ and $p = 5$. FOT was not displayed in the figure as it is hardly differentiable from FOD. We can see that FOD and MDT are slightly superior to FMDT with respect to their compression ratios. In contrast to this, FMDT yields rather smaller relative errors than MDT and FOD. But with increasing image complexity, the errors and compression ratios of all methods slowly align. Therefore, MDT and FOD seem to be reasonable choices for thinning methods in MAT compression. Thus, we could just pick MDT for our universal parameter chonfiguration.

If a compromise between the three criteria is not necessary, i. e., if one criterion is favored, we can interchange the parameters as needed. For example, if the quality is the most important, a good parameter configuration might be FOA, $p = 1$, and $a = 0.5$. At the same time, one would neglect the runtime and compression ratio. Nevertheless, the compression ratio can still compete with JPEG and JPEG-2000. If we only aim at a high compression ratio, we could increase a and p until the desired rate is reached, supposed that the reconstruction quality still suffices our needs.

Finally, we see that we can tune the parameters such that our needs, i. e., a good universal compression, a great compression ratio, or few erroneous pixels, are fulfilled. The above determined universal parameters give a stable compromise between the mentioned criteria.

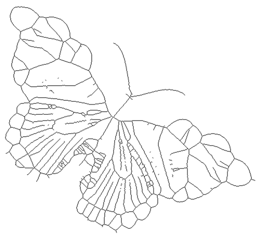
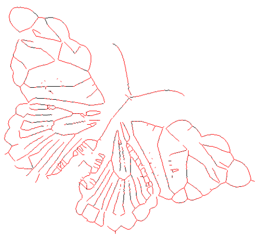
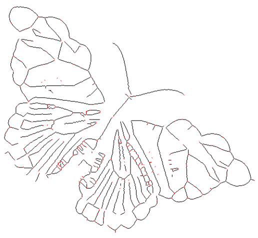
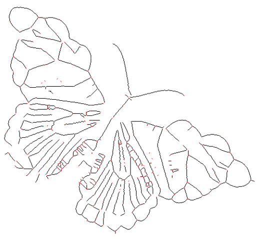
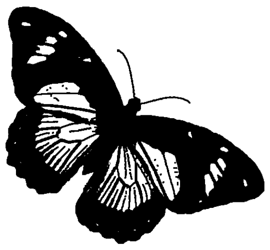
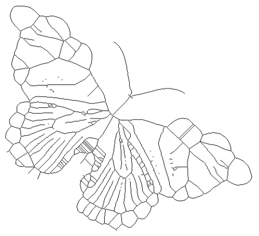
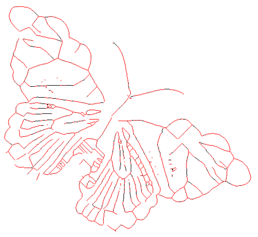
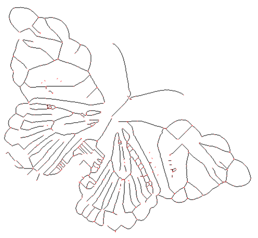
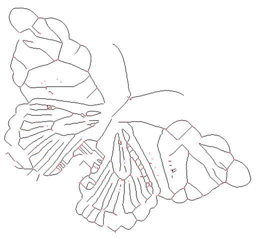
Skeleton and Thinning Method	Para-meter	Preprocessed Skeleton	Decompressed Object
 FMDT	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		
 MDT	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		

Table 5.7: The skeletons of Figure 5.1c computed with FMDT and MDT. Each skeleton has been preprocessed with three different parameter assignments of a and p. The result can be seen in column three. The red pixels represent the data points that are chosen for approximation during the compression step. The fourth column shows the reconstructed object from the decompressed data.

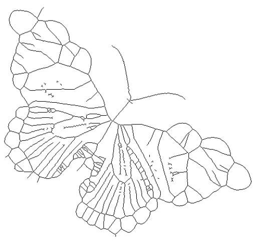
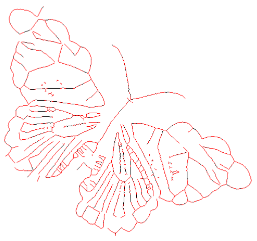
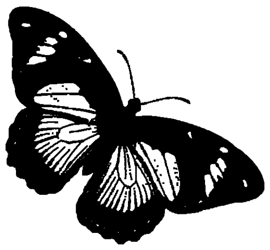
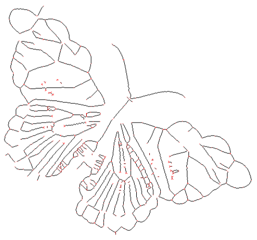
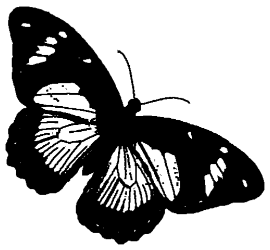
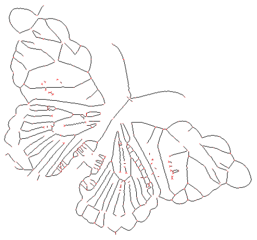
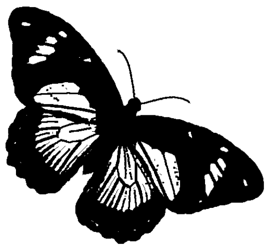
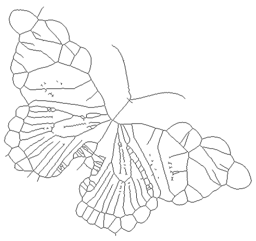
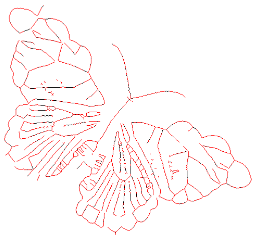
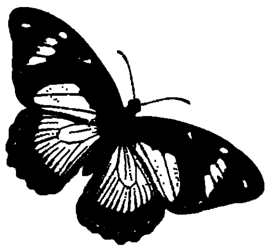
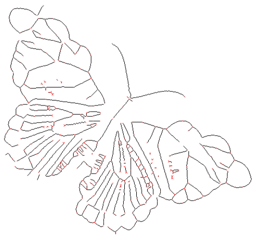
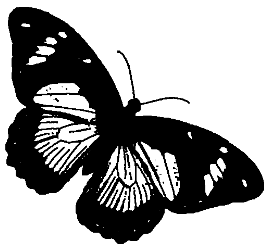
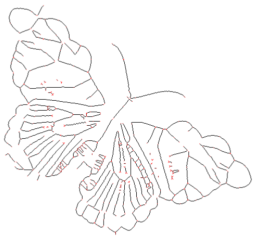
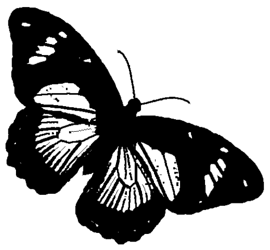
Skeleton and Thinning Method	Parameter	Preprocessed Skeleton	Decompressed Object
 FOD	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		
 FOT	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		

Table 5.8: The skeletons of Figure 5.1c computed with FOD and FOT. Each skeleton has been preprocessed with three different parameter assignments of a and p. The result can be seen in column three. The red pixels represent the data points that are chosen for approximation during the compression step. The fourth column shows the reconstructed object from the decompressed data.

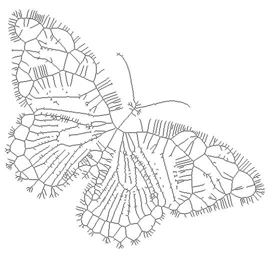
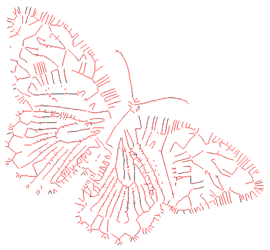
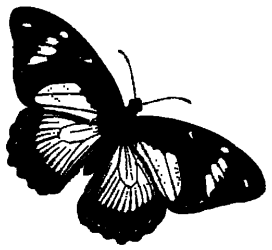
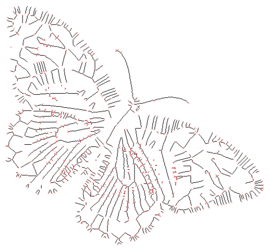
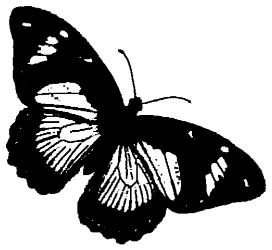
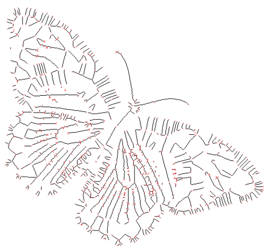
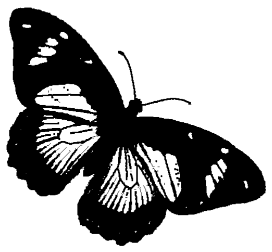
Skeleton and Thinning Method	Para-meter	Preprocessed Skeleton	Decompressed Object
 FOA	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		

Table 5.9: The skeletons of Figure 5.1c computed with FOA. Each skeleton has been preprocessed with three different parameter assignments of a and p. The result can be seen in column three. The red pixels represent the data points that are chosen for approximation during the compression step. The fourth column shows the reconstructed object from the decompressed data.

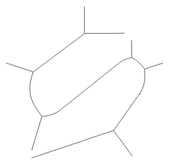
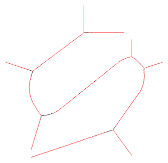
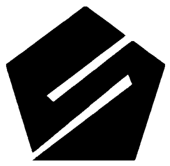
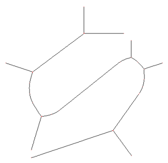
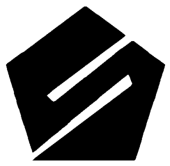
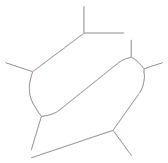
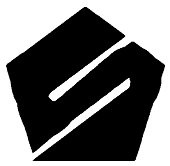
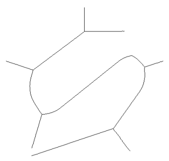
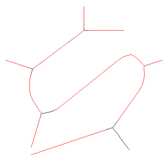
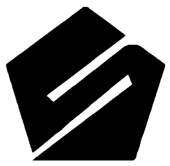
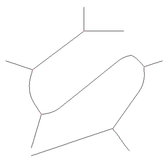
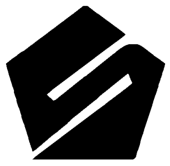
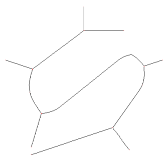
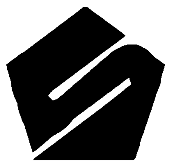
Skeleton and Thinning Method	Para-meter	Preprocessed Skeleton	Decompressed Object
 FMDT	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		
 MDT	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		

Table 5.10: The skeletons of Figure 5.1h computed with FMDT and MDT. Each skeleton has been preprocessed with three different parameter assignments of a and p. The result can be seen in column three. The red pixels represent the data points that are chosen for approximation during the compression step. The fourth column shows the reconstructed object from the decompressed data.

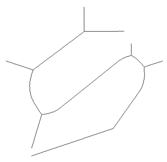
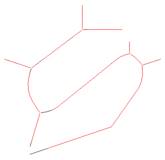
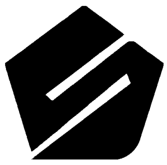
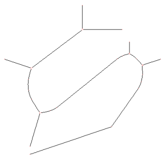
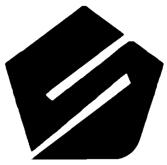
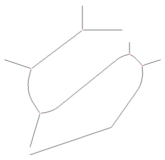
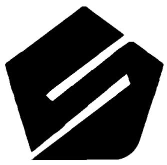
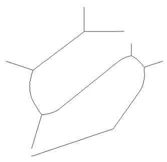
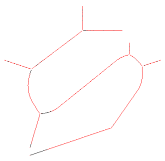
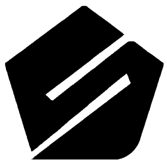
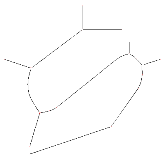
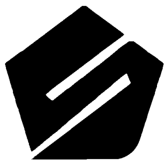
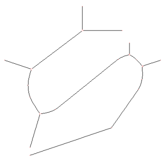
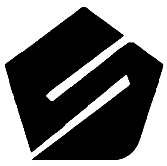
Skeleton and Thinning Method	Para- meter	Preprocessed Skeleton	Decompressed Object
 FOD	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		
 FOT	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		

Table 5.11: The skeletons of Figure 5.1h computed with FOD and FOT. Each skeleton has been preprocessed with three different parameter assignments of a and p. The result can be seen in column three. The red pixels represent the data points that are chosen for approximation during the compression step. The fourth column shows the reconstructed object from the decompressed data.

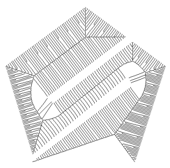
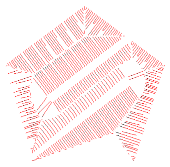
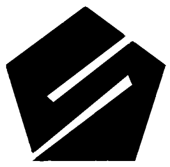
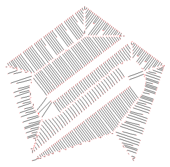
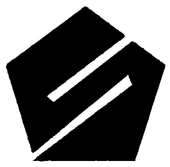
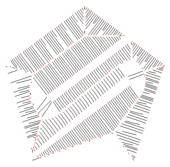
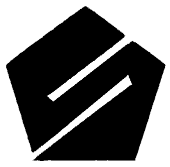
Skeleton and Thinning Method	Parameter	Preprocessed Skeleton	Decompressed Object
 FOA	$a = 0.5$, $p = 1$		
	$a = 1$, $p = 5$		
	$a = 2$, $p = 10$		

Table 5.12: The skeletons of Figure 5.1h computed with FOA. Each skeleton has been preprocessed with three different parameter assignments of a and p. The result can be seen in column three. The red pixels represent the data points that are chosen for approximation during the compression step. The fourth column shows the reconstructed object from the decompressed data.

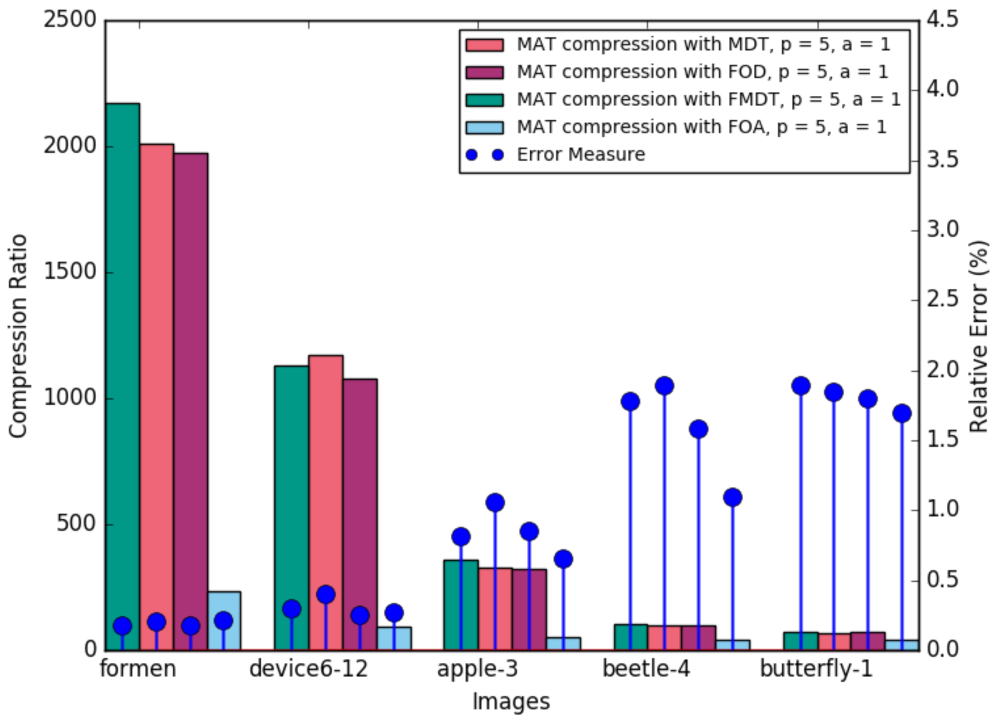


Figure 5.4: The performances of MAT compression using MDT, FOD, FMDT, and FOA, $a = 1$, and $p = 5$ are compared with respect to their compression ratios and the corresponding relative number of erroneous pixels.

5.4 Sparsification

In this section, we examine the sparsification approach from Section 3.5 and compare it to MAT compression. Remember that sparsification requires to determine the percentage of pixels that we want to remove from the skeleton. To do so, we choose the percentage such that the relative error is close to the error achieved by MAT compression with the parameter configuration MDT, $p = 5$, and $a = 1$. We restricted the comparative experiments to the images `apple-3` and `pencil-17` as they nicely illustrate the advantages and disadvantages of sparsification. Table 5.13 provides information about the outcomes of both of the mentioned approaches. The outcomes of MAT compression will be used as a reference for our comparisons.

Although the relative errors for `pencil-17` of MAT compression and the sparsification approach are nearly identical, the image quality of the decompressed sparsification approach suffers. This is illustrated by Figure 5.5b and Figure 5.5d. The unpleasant, round boundary artifacts of Figure 5.5d occur due to many discon-

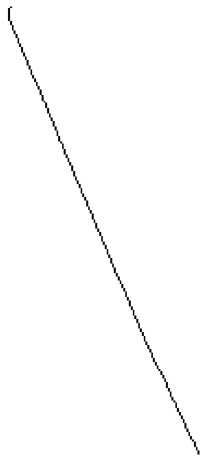
Test Image	Universal Approach			Sparsification		
	Relative Error	Comp. Ratio	Time (sec)	Comp. Ratio	Deleted Points (%)	Time (sec)
`apple-3`	0.82	357	0.4	493	92.00	264.9
`pencil-17`	0.58	1002	1.2	858	92.15	79.8

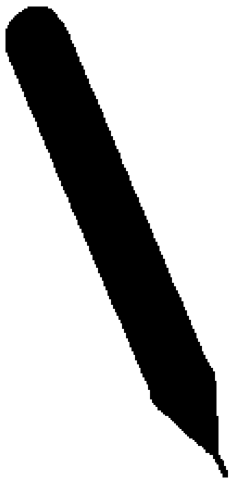
Table 5.13: This table contains the relative errors produced by MAT compression with MDT, $p = 5$, and $a = 1$ as well as the corresponding compression ratios. The fourth column provides the compression ratios of the sparsification approach that achieved a similar relative error on the corresponding input image. For the approximation threshold, we took $a = 1$. The fifth column shows the percentage of pixels that were deleted from the skeleton during sparsification and the last one depicts the runtime of sparsification.

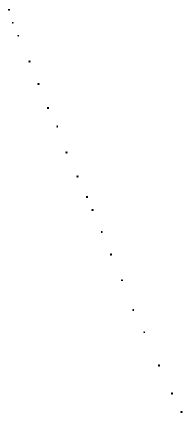
nected, single points that remain after sparsification. See Figure 5.1f and Figure 5.5c for details. Besides this, also the compression ratio suffers from the skeleton structure because the singletons create an overhead when transferring them into the encoding scheme from Subsection 3.2.2.

We observe another disadvantage of the sparsification approach in Figure 5.6. We can see in Figure 5.6d, that the object's connectivity is destroyed as parts of the apple's stalk disappear whereas the reference image Figure 5.6b fully preserved the structure. Apart from that, the sparsification approach gives us a very good reconstruction of the apple's body. In particular, we see in Figure 5.6c that only few points are needed for the representation of its skeleton. To be precise, we could delete more than 90% of the skeleton points. The reason for that is that the body mainly consists of two circle-like parts that benefit from the way we reconstruct the MAT.

Consequently, for objects that are composed of a few round structures, it is possible to remove a huge number of pixels without affecting the reconstruction quality. This results in a compression ratio that tends to be higher than the compression ratio achieved by MAT compression. Nevertheless, we risk disconnections of the object if we choose the percentage of sparsification pixels too high. Additionally, for objects with sharp edges and corners, such as `pencil-17`, MAT compression seems to provide a visually more pleasant image quality and higher compression ratio. More explicitly, our reference does not decompose the connected skeleton into single pixels that possibly lead to a storage overhead. Finally, we must also remark that the runtime of the recent implementation of the sparsification approach is remarkably higher than the runtime achieved by MAT compression.


(a) preprocessed skeleton


(b) decompressed reconstruction


(c) sparsified skeleton

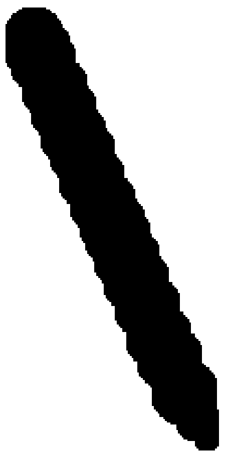
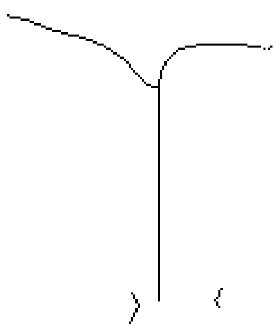
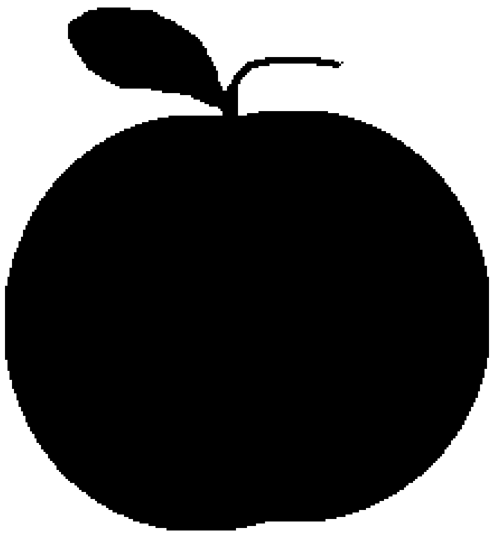

(d) decompressed reconstruction

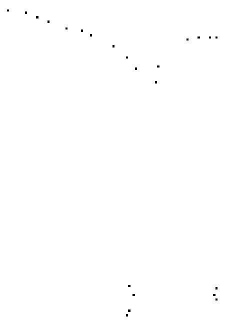
Figure 5.5: Figure 5.5a and Figure 5.5b depict the skeleton of `pencil-17` preprocessed according to MAT compression with MDT, $p = 5$, and $a = 1$ and its reconstruction, respectively. Figure 5.5c shows the MDT thinned skeleton of `pencil-17` from which 92.15% of the points were removed. The remaining skeleton was compressed with MAT compression with $a = 1$. Figure 5.5d shows the decompressed object.



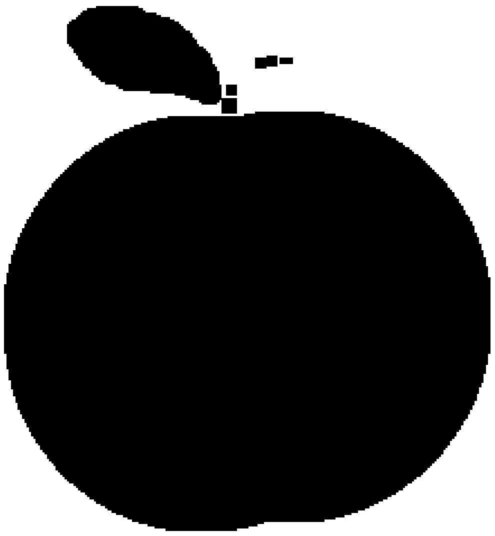
(a) preprocessed skeleton



(b) decompressed reconstruction



(c) sparsified skeleton



(d) decompressed sparsification

Figure 5.6: Figure 5.6a depicts the skeleton of `apple-3` preprocessed according to MAT compression with MDT, $p = 5$, and $a = 1$. Figure 5.6b shows the corresponding decompressed object. The sparsified skeleton and its reconstructed object can be seen in Figure 5.6c and Figure 5.6d, respectively.

6 Conclusions and Outlook

In this chapter, we summarize the main contributions of this thesis and draw conclusions about the suitability of the medial axis transform for binary image compression. Further, we discuss possible future research based on MAT compression.

6.1 Summarizing Remarks

In this work, we developed a near-lossless binary image compression codec named MAT compression. The codec compresses the skeleton that is obtained after applying one of the five thinning methods FOT, FOD, FMDT, MDT, and FOA that were presented in Subsection 2.2.1 to the input image. Based on the resulting skeleton, we proposed to first reduce spurious branches that arise from fine-scaled details and boundary perturbations in order to increase the efficiency of the subsequent compression. To this end, we introduced a measure that assigns importance values to skeleton branches. MAT compression removes those branches whose importance values fall below a user-defined threshold p. To further minimize discretization artifacts arising from skeleton thinning, we proposed a line smoothing algorithm that is applied to all remaining skeleton lines. Next, we used the near-optimal linear approximation approach proposed by Kolesnikov et al. [KF03] to thin the amount of data that represents the MAF. This approach depends on an approximation parameter a with which the user can tune the exactness of approximation. Afterwards, we introduced an 8-directional relative chain code that describes the skeleton line by line. Finally, MAT compression transferred the chain coded skeleton and the remaining MAF into an encoding scheme and compresses it with LPAQ.

We conducted a series of tests on ten images to evaluate the performance of MAT compression with respect to the compression ratio, the exactness of reconstruction, and the runtime. For each image, we tested 60 different combinations of the three parameters on which MAT compression depends. After extensively evaluating the influence of all parameters on the test data set, we could determine a universal parameter configuration for MAT compression that seems to yield a stable trade-off between the above mentioned quality criteria. This parameter configuration consists of the skeletonization method MDT, the pruning parameter $p = 5$, and the approximation parameter $a = 1$.

Afterwards, we compared MAT compression to JPEG, JPEG-2000, JBIG, and modified Huffman encoding. We found that the existing compression standards yield faster and more exact results. With respect to the compression ratio, MAT compression outperformed JPEG and JPEG-2000 on our test set. The competitiveness of MAT compression to modified Huffman encoding and JBIG was strongly influenced by the choice of the thinning method, i. e., choosing FOA, the obtained results could only compete with the Huffman approach for a suitable choice of a and p, whereas no parameter configuration containing FOA could compete with JBIG. In contrast to FOA, all other thinning methods could rival with these standards independent of the choice of p and a. Especially on geometric image data, the tests confirmed the strong performance of MAT compression. For $a \geq 1$, our approach achieved compression ratios that were significantly larger than those of JBIG but we also observed that the more complex the image data becomes, the more does JBIG outperform our standard unless we increase the values of a and p accordingly. But we have to be careful as large values worsen the exactness of reconstruction. Consequently, MAT compression seems to be a reasonable alternative to JBIG if we are given binary images that depict objects that are composed of geometric structures. This demonstrates as well that the MAT has potential for image compression. In particular, we believe that the performance of MAT compression can be improved even further. We discuss some ideas in the next section.

Finally, we compared MAT compression to the sparsification approach presented in Section 3.5. We found that this approach does not seem to be a suitable replacement of MAT compression due to its large runtime and undesired artifacts that arise from the many scattered points that remain after compression with sparsification.

6.2 Future Work

As mentioned in the previous section, MAT compression has the potential for further advancement. Therefore, future research could continue to explore the influence of preprocessing methods in order to improve the overall quality of MAT compression. Additional methods, such as a boundary presmoothing, can be used to refine the preprocessing step. We could also replace our rather simple threshold-based pruning process by a more advanced one, such as the method introduced by Heidrich et al. [TH02].

The encoding scheme developed in Subsection 3.2.2 constitutes another option for further studies as the encoding of the MAF and scattered points at the end of the file is a source of storage overhead. One could prevent such problems by incorporating information into the line encoding, for instance, with the help of a suitable, invertible pairing function.

Further studies might also examine if the compression approaches by Lukač et al. [ZL14; ZML15] yield an improvement in contrast to the LPAQ encoded relative chain codes.

Furthermore, a disadvantage of the MAT compression are the three tunable parameters as they require the user to estimate the size of noise that is present on the object boundaries. Thus, future research could automate the selection of these parameters.

Finally, the sparsification approach should be pursued in future work. To speed up the runtime, we could parallelize the computation of the importance measure. Furthermore, the artifacts that were observed in Section 5.4 rather negatively affect the sparsification approach. Thus, methods that support the avoidance of such artifacts need to be introduced.

A Command Line Tools

In this appendix, we give an overview of the use of the command line tools used for MAT compression and the comparative experiments. Thereby, the first paragraph introduces the tool for MAT compression and decompression. The remaining paragraphs explain the tools used in the evaluation chapter. Each paragraph starts with the name of the tool in bold, followed by its arguments. The arguments in square brackets are optional. If there are several options within square brackets separated by vertical bars, we have to choose at most one of those options. Below the list of arguments, we explain their meanings as well as their relations to each other if present.

`compress` `[-fod|-fot|-fmdt|-mdt|-foa] [-rec] [-sparse` n`] [-color] [-preprocess` p`] [-compress` a`] [-decompress] input_file`

- `[-fod|-fot|-fmdt|-mdt|-foa]`: Exactly one of those thinning methods must be chosen if we want to use the parameters `rec`, `sparse` n, `color`, `preprocess` p, and `compress` a. It returns a PGM file that depicts the thinned skeleton.
- `rec`: The optional parameter produces a PGM file that contains the reconstructed image from the original skeleton.
- `-sparse` n: This parameter deletes those n% of the skeleton points that are least important for the reconstruction quality. $0 \leq n < 1$ has to hold for the sparsification parameter n.
- `-color`: Produces a PGM that depicts the original or sparsified skeleton with colored line segments.
- `-preprocess` p: This is the preprocessing part of MAT compression. It prunes lines that have an influence smaller than p pixels and smooths the remaining

skeleton lines. It outputs a PGM file that shows the preprocessed skeleton, and a PGM file depicting the reconstructed image based on the computed skeleton.

- `-compress` a: This is the second part of MAT compression. It approximates the MAF with the help of the approximation threshold $a > 0$ and outputs an LPAQ encoded file.
- `-decompress` a: It decompresses the previously LPAQ encoded file and returns the reconstructed object in a PGM format. Additionally, it writes information about the reconstruction quality, compression ratio, and decompression times in the output file if it is preceded by `-compress` a.
- `input_file`: This is the name of the input file. It is either a binary PGM file or the previously LPAQ encoded file. If the file is a PGM, the use of `-decompress` must at least be preceded by the call of one of the five thinning methods and `-compress a`. If the filename ends with *_lpaq1*, we must call `-decompress`. In this case, no other argument is allowed.

Runtime information is written in an output file for all of the above mentioned tool arguments. To execute MAT compression as described in the previous chapters, the tool could be called, for instance, as follows:

```
./compress -mdt -preprocess 5 -compress 1 -decompress example.pgm
```

`gmTests`

This tool compresses all test images with JPEG, JPEG-2000, and JBIG via GraphicsMagick. It outputs a text file for each image providing information about the compression times, compression ratios, and relative errors for all three tested compression standards.

`huffman`

We used `huffman` for the comparative experiments as it compresses the test images automatically with modified Huffman encoding and creates text files containing information about the compression times and compression ratios.

`testscript`

`testscript` executes MAT compression with all 60 parameter combinations for each of the ten test images. It provides for each MAT compression a text file containing detailed information about reconstruction errors (including the relative error that we used for evaluation in Chapter 5) as well as runtime and compression ratio information.

B Experimental Results

In the following, five tables are depicted that provide exact experimental results for all 60 parameter combinations and all three quality criteria for the images:

- `formen`: Table B.1
- `device6-12`: Table B.2
- `apple-3`: Table B.3
- `beetle-4`: Table B.4
- `butterfly-1`: Table B.5

Parameter Choice		Thinning Method														
		FMDT			FOA			FOD			FOT			MDT		
approx. error a	preproc. param. p	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time
$a = 0.5$	$p = 1$	328	0.13	1.6	51	0.16	63.7	337	0.13	1.3	337	0.13	1.3	343	0.14	1.3
	$p = 5$	339	0.13	1.7	51	0.16	64.1	329	0.13	1.3	329	0.13	1.3	350	0.14	1.3
	$p = 10$	353	0.13	1.7	52	0.16	61.4	338	0.13	1.3	338	0.13	1.3	350	0.14	1.3
$a = 1$	$p = 1$	1758	0.18	1.8	229	0,22	63.9	2143	0.21	1.5	2143	0,21	1.5	2034	0.18	1.5
	$p = 5$	1975	0.18	1.8	232	0.22	62.5	2008	0.21	1.5	2009	0.21	1.5	2172	0.18	1.5
	$p = 10$	1967	0.18	1.8	234	0.22	60.8	2134	0.21	1.5	2134	0.21	1.5	2172	0.18	1.5
$a = 1.5$	$p = 1$	1758	0.18	1.8	229	0.22	61.9	2143	0.21	1.5	2143	0.21	1.5	2034	0.18	1.5
	$p = 5$	1975	0.18	2.2	232	0.22	62.3	2009	0.21	1.5	2009	0.21	1.5	2172	0.18	1.5
	$p = 10$	1967	0.18	1.8	234	0.22	61.5	2134	0.21	1.5	2134	0.21	1.5	2172	0.18	1.5
$a = 2$	$p = 1$	1846	0.30	1.8	262	0.27	61.9	2212	0.33	1.5	2212	0.33	1.4	2143	0.30	1.4
	$p = 5$	2087	0.31	2.0	265	0.27	62.9	2105	0.33	1.5	2105	0.33	1.4	2297	0.31	1.4
	$p = 10$	2096	0.31	1.7	268	0.27	60.8	2202	0.33	1.5	2202	0.33	1.4	2297	0.31	1.4

Table B.1: The exact experimental results for all combinations of thinning methods, approximation error thresholds, and pruning thresholds with respect to the compression ratio, the relative error, and the runtime are depicted for the image *formen*

Parameter Choice		Thinning Method														
		FMDT			FOA			FOD			FOT			MDT		
approx. error a	preproc. param. p	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time
$a = 0.5$	$p = 1$	122	0.19	1.3	13	0.38	183.9	132	0.34	0.9	132	0.34	1.0	134	0.25	1.0
	$p = 5$	122	0.19	1.3	13	0.38	203.5	132	0.34	1.0	132	0.34	1.0	134	0.25	1.0
	$p = 10$	122	0.19	1.3	13	0.38	187.1	132	0.34	0.9	132	0.34	1.3	134	0.25	1.3
$a = 1$	$p = 1$	1078	0.25	2.9	96	0.26	187.9	1170	0.40	4.9	1170	0.40	5.8	1128	0.30	3.9
	$p = 5$	1078	0.25	2.9	96	0.27	189.7	1170	0.40	5.0	1170	0.40	5.7	1128	0.30	4.0
	$p = 10$	1078	0.25	2.9	97	0.26	207.3	1170	0.40	5.0	1170	0.40	5.8	1128	0.30	3.3
$a = 1.5$	$p = 1$	1078	0.25	2.9	96	0.26	220.7	1170	0.40	5.0	1170	0.40	5.4	1128	0.30	4.0
	$p = 5$	1078	0.25	2.9	96	0.27	188.8	1170	0.40	4.9	1170	0.40	5.7	1128	0.30	4.0
	$p = 10$	1078	0.25	2.9	97	0.26	187.9	1170	0.40	4.9	1170	0.40	6.2	1128	0.30	3.9
$a = 2$	$p = 1$	1128	0.27	2.8	125	0.75	225.5	1210	0.41	4.9	1205	0.41	6.2	1200	0.34	3.2
	$p = 5$	1128	0.27	2.8	126	0.76	187.2	1210	0.41	4.8	1205	0.41	5.7	1200	0.34	3.3
	$p = 10$	1128	0.27	2.8	127	0.76	200.9	1210	0.41	4.9	1205	0.41	5.0	1200	0.34	3.3

Table B.2: The exact experimental results for all combinations of thinning methods, approximation error thresholds, and pruning thresholds with respect to the compression ratio, the relative error, and the runtime are depicted for the image *device6-12*

Parameter Choice		Thinning Method														
		FMDT			FOA			FOD			FOT			MDT		
approx. error a	preproc. param. p	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time
$a = 0.5$	$p = 1$	90	0.53	0.3	9	0.76	46.8	100	0.81	0.2	101	0.80	0.2	100	0.53	0.3
	$p = 5$	92	0.53	0.3	9	0.75	45.3	100	0.81	0.2	101	0.80	0.2	101	0.54	0.3
	$p = 10$	92	0.53	0.3	9	0.75	44.7	100	0.81	0.2	101	0.80	0.2	102	0.54	0.3
$a = 1$	$p = 1$	303	0.85	0.4	52	0.66	47.9	329	1.06	0.3	331	1.06	0.3	352	0.81	0.4
	$p = 5$	325	0.85	0.4	52	0.66	46.4	329	1.06	0.3	331	1.06	0.3	357	0.82	0.4
	$p = 10$	325	0.85	0.4	53	0.66	49.5	329	1.06	0.3	331	1.06	0.3	369	0.82	0.4
$a = 1.5$	$p = 1$	303	0.85	0.4	52	0.66	47.4	329	1.06	0.3	331	1.06	0.3	352	0.81	0.4
	$p = 5$	325	0.85	0.4	52	0.66	46.6	329	1.06	0.3	331	1.06	0.3	357	0.82	0.4
	$p = 10$	325	0.85	0.4	53	0.66	48.1	329	1.06	0.3	331	1.06	0.3	369	0.82	0.4
$a = 2$	$p = 1$	319	0.97	0.4	58	1.15	47.4	374	1.42	0.3	377	1.42	0.3	374	0.92	0.4
	$p = 5$	343	0.98	0.4	59	1.15	46.6	374	1.42	0.3	377	1.42	0.3	377	0.93	0.4
	$p = 10$	343	0.98	0.4	59	1.15	49.3	374	1.42	0.3	377	1.42	0.3	393	0.93	0.4

Table B.3: The exact experimental results for all combinations of thinning methods, approximation error thresholds, and pruning thresholds with respect to the compression ratio, the relative error, and the runtime are depicted for the image *apple-3*

Parameter Choice		Thinning Method														
		FMDT			FOA			FOD			FOT			MDT		
approx. error a	preproc. param. p	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time
$a = 0.5$	$p = 1$	24	0.82	6.6	12	0.87	94.0	24	0.98	5.4	24	0.98	5.3	25	1.03	5.7
	$p = 5$	24	0.83	6.5	12	0.88	84.0	24	0.99	5.4	24	0.99	5.4	25	1.02	6.7
	$p = 10$	24	0.83	6.5	12	0.89	100.4	24	0.99	5.4	24	0.99	5.4	25	1.04	5.6
$a = 1$	$p = 1$	96	1.56	8.0	41	1.10	98.7	99	1.86	7.0	98	1.86	7.1	107	1.76	8.5
	$p = 5$	97	1.58	7.8	41	1.10	85.8	99	1.89	7.1	99	1.89	7.0	106	1.78	8.6
	$p = 10$	97	1.58	8.1	41	1.12	89.8	100	1.89	7.1	100	1.90	7.0	108	1.75	7.3
$a = 1.5$	$p = 1$	96	1.56	8.0	41	1.10	97.8	99	1.86	7.0	98	1.86	7.0	107	1.76	8.5
	$p = 5$	97	1.58	7.8	41	1.10	84.6	99	1.89	7.1	99	1.89	7.0	106	1.78	8.7
	$p = 10$	97	1.58	8.1	41	1.12	83.8	100	1.89	7.0	100	1.90	7.0	108	1.75	7.2
$a = 2$	$p = 1$	108	1.88	8.2	44	1.36	88.9	113	2.00	7.3	113	2.00	7.2	121	1.92	7.6
	$p = 5$	109	1.89	8.0	44	1.36	94.9	114	2.02	7.2	114	2.02	7.2	121	1.96	8.5
	$p = 10$	109	1.90	8.3	44	1.32	82.9	115	2.02	7.3	115	2.02	7.3	123	1.86	7.3

Table B.4: The exact experimental results for all combinations of thinning methods, approximation error thresholds, and pruning thresholds with respect to the compression ratio, the relative error, and the runtime are depicted for the image *beetle-4*

Parameter Choice		Thinning Method														
		FMDT			FOA			FOD			FOT			MDT		
approx. error a	preproc. param. p	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time	Ratio	Quality	Time
$a = 0.5$	$p = 1$	19	1.17	16.0	13	1.20	63.3	18	1.16	17.1	18	1.17	15.7	19	1.26	13.6
	$p = 5$	19	1.17	13.9	13	1.17	67.0	18	1.16	18.2	18	1.17	15.7	19	1.26	14.1
	$p = 10$	19	1.16	14.9	13	1.20	58.9	18	1.15	17.0	18	1.16	15.4	19	1.25	16.2
$a = 1$	$p = 1$	71	1.80	15.8	41	1.70	71.3	68	1.85	20.0	69	1.86	17.8	74	1.89	14.5
	$p = 5$	71	1.80	16.8	41	1.70	63.5	68	1.85	20.8	69	1.86	18.7	73	1.89	18.8
	$p = 10$	71	1.80	17.7	41	1.70	70.5	68	1.85	19.6	69	1.85	17.4	74	1.89	14.4
$a = 1.5$	$p = 1$	71	1.80	15.7	41	1.70	73.5	68	1.85	20.1	69	1.86	19.0	74	1.89	14.5
	$p = 5$	71	1.80	16.7	41	1.70	59.8	68	1.85	19.9	69	1.86	17.2	73	1.89	18.1
	$p = 10$	71	1.80	16.5	41	1.70	74.4	68	1.85	20.1	69	1.85	16.4	74	1.89	14.4
$a = 2$	$p = 1$	80	1.90	16.7	44	1.97	74.4	77	2.13	21.7	77	2.19	16.6	84	2.04	14.5
	$p = 5$	80	1.90	15.6	44	1.98	59.0	77	2.13	21.4	77	2.19	16.6	84	2.05	18.8
	$p = 10$	80	1.89	15.5	44	1.97	64.0	77	2.12	20.7	77	2.18	18.1	84	2.03	14.3

Table B.5: The exact experimental results for all combinations of thinning methods, approximation error thresholds, and pruning thresholds with respect to the compression ratio, the relative error, and the runtime are depicted for the image *butterfly-1*

C Close-Up of `butterfly-1`

According to Table 5.7, we show a close-up of the skeleton that was thinned with MDT, $p = 10$, and $a = 2$ (Figure C.1) and the corresponding decompressed reconstruction (Figure C.2a). Below Figure C.2a, we show the original object (Figure C.2b) to enable a direct comparison.

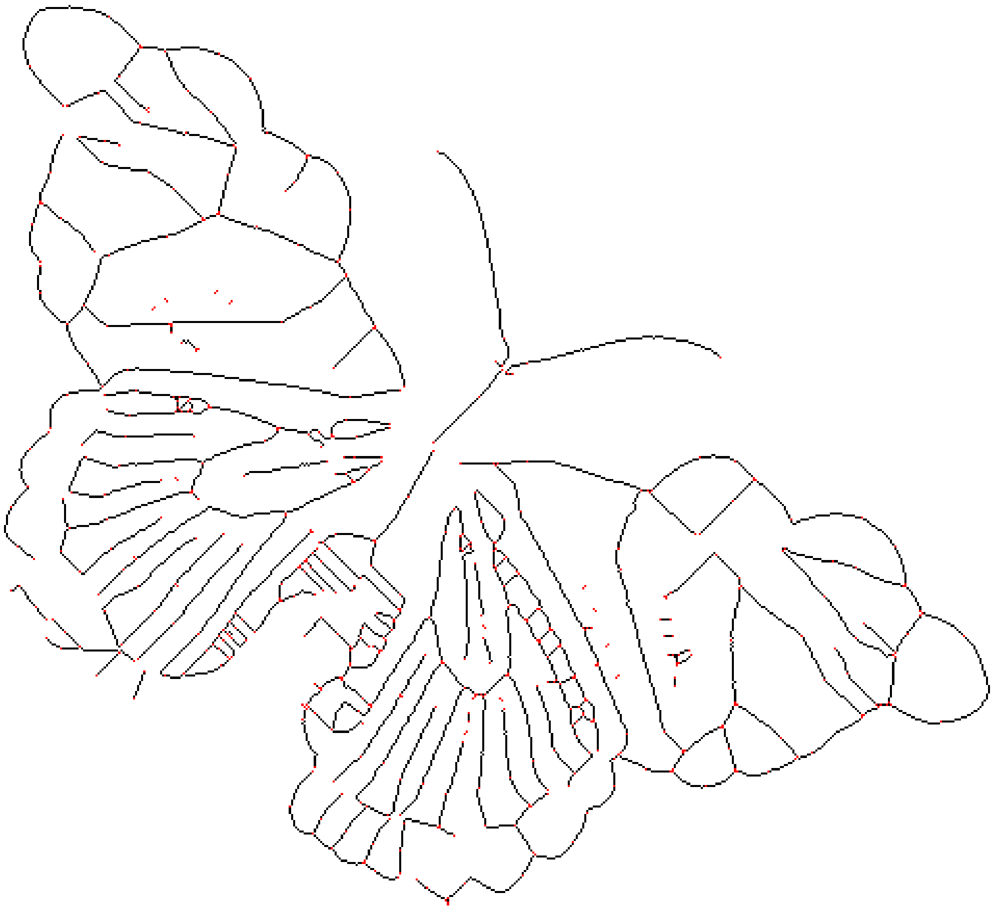
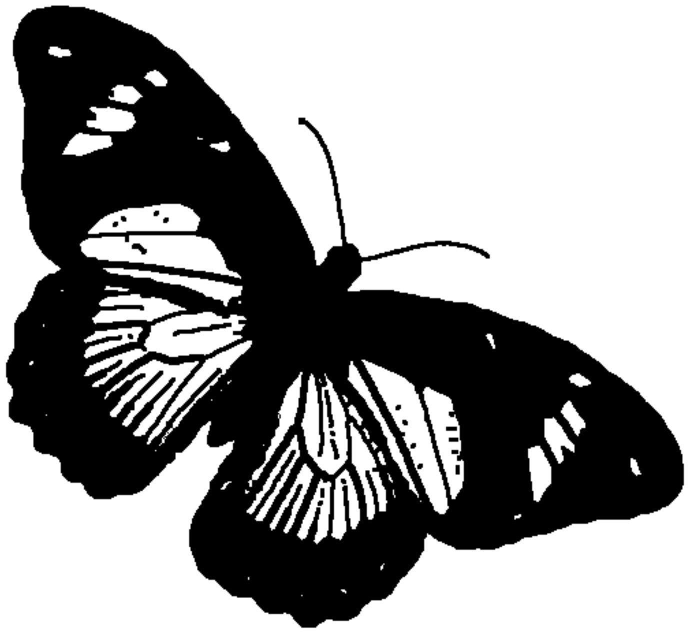
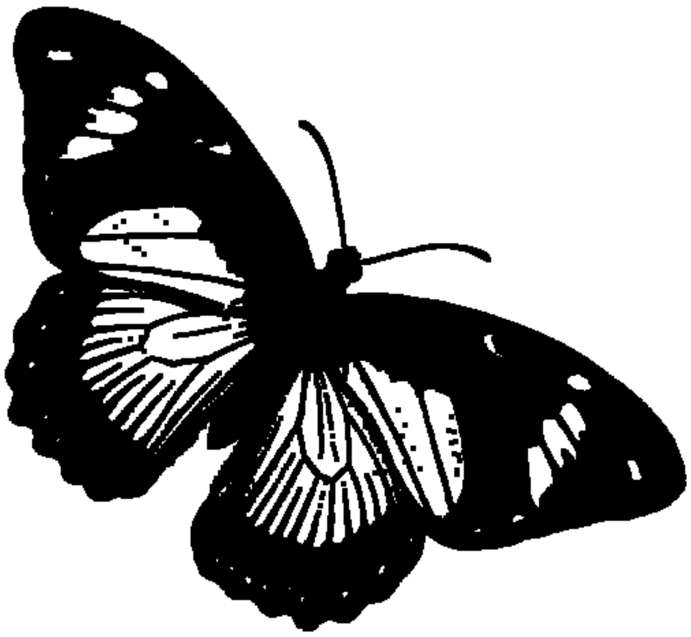


Figure C.1: A close-up of the skeleton for `butterfly-1`. Some of the original skeleton lines were pruned or smoothed during preprocessing. The red pixels represent the skeleton points that were selected for approximation.



(a) decompressed reconstruction



(b) original `butterfly-1`.

Figure C.2: Figure C.2a is a close-up of the decompressed reconstruction based on the skeleton of Figure C.1. Figure C.2b is a close-up of the original `butterfly-1`.

D Implementations

In the following, we list all external program codes that were not written by the author, Marie Mühlhaus.

Basic Tools. We used and extended the code for memory allocation, file input and output as well as the data structures provided by Pascal Peter.

Coloring Algorithm. For skeleton line coloring, we took colors that were computed by a slightly modified version of the random color algorithm by Martin Ankerl (`https://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/`).

JBIG. For the comparative experiments, we used the *convert* tool that is provided by GraphicsMagick 1.3.34 (`http://www.graphicsmagick.org/`). It uses the JBIG-KIT 2.1 by Markus Kuhn (`https://www.cl.cam.ac.uk/~mgk25/jbigkit/`). All parameters remained at their default settings.

JPEG. For the comparative experiments with JPEG, we converted the images with GraphicsMagick 1.3.34 with all parameter at their default settings.

JPEG-2000. For the comparative experiments with JPEG-2000, we converted the images with GraphicsMagick 1.3.34 with all parameter at their default settings. Thereby, we used the Jasper library v2.0.14 (`https://www.ece.uvic.ca/~frodo/jasper/`).

LPAQ. The reference implementation of LPAQ provided by Matt Mahoney (`http://mattmahoney.net/dc/`) was used as a last step in MAT compression.

Modified Huffman Encoding. RLE was self-written. Its output was then encoded with Huffman encoding with the help of a code that was given by Pascal Peter.

Thinning Methods. In order to skeletonize the input data, we used the implementation by Pascal Peter that includes the codes for FMDT, MDT, FOD, FOT, and FOA.

E Glossary

FMDT	Flux-Ordered Maximal Disc Thinning.
FOA	Flux-Ordered Adaptive Thinning.
FOD	Flux-Ordered Thinning.
FOT	Flux-Ordered Thinning (Modification of FOD).
GIF	Graphics Interchange Format.
MAF	Medial Axis Function.
MAT	Medial Axis Transform.
MDT	Maximal Disc Thinning.
PGM	Portable Graymap.
PPM	Prediction by Partial Matching.
R8CC	8-Directional Relative Chain Code.
RLE	Runlength Encoding.

F List of Figures

G List of Tables

H

List of Algorithms

Bibliography

[BJA91] J. Brandt, A. Jain, and V. Algazi. “Medial axis representation and encoding of scanned documents”. In: *Journal of Visual Communication and Image Representation* 2 (1991), pp. 151–165.

[Bla+12] K. Blaszczyk et al. *PAQ compression algorithm*. 2012.

[Blu67] H. Blum. “A transformation for extracting new descriptors of shape”. In: *Models for the perception of speech and visual form* 19 (1967), pp. 362–380.

[Bri99] E. Bribiesca. “A new chain code”. In: *Pattern Recognition* 32 (1999), pp. 235–251.

[Dun86] J. Dunham. “Optimum Uniform Piecewise Linear Approximation of Planar Curves”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8 (1986), pp. 67–75.

[Eli75] P. Elias. “Universal codeword sets and representations of the integers”. In: *IEEE Transactions on Information Theory* 21 (1975), pp. 194–203.

[Fre74] H. Freeman. “Computer Processing of Line-Drawing Images”. In: *ACM Computing Surveys* 6 (1974), pp. 57–97.

[HC94] B. Hamann and J. Chen. “Data point selection for piecewise linear curve approximation”. In: *Computer Aided Geometric Design* 11 (1994), pp. 289–301.

[Huf06] D. Huffman. “A method for the construction of minimum-redundancy codes”. In: *Resonance* 11 (2006), pp. 91–99.

[KF02] A. Kolesnikov and P. Fränti. *A fast near-optimal min-# polygonal approximation of digitized curves*. 2002.

[KF03] A. Kolesnikov and P. Fränti. “Reduced-search dynamic programming for approximation of polygonal curves”. In: *Pattern Recognition Letters* 24 (2003), pp. 2243–2254.

[Kyr99] V. Kyrki. *JBIG image compression standard*. 1999.

[LD91] C. Lu and J. Dunham. “Highly Efficient Coding Schemes for Contour Lines Based on Chain Code Representations”. In: *IEEE Transactions on Communications* 39 (1991), pp. 1511–1514.

[Liu+07] Y. Liu et al. “Compressed vertex chain codes”. In: *Pattern Recognition* 40 (2007), pp. 2908–2913.

[Lon92] J. Longin. *MPEG-7 Core Experiment CE-Shape-1 Test Set*. 1992. URL: `http://www.dabi.temple.edu/~shape/MPEG7/dataset.html` (visited on 02/02/2020).

[Mah13] M. Mahoney. *Data compression explained*. 2013. URL: `http://mattmahoney.net/dc/dce.html` (visited on 02/02/2020).

[MRH00] A. Meijster, J. Roerdink, and W. Hesselink. “A General Algorithm For Computing Distance Transforms In Linear Time”. In: *Computational Imaging and Vision,ISMM*. Vol. 18. Springer, 2000, pp. 331–340.

[MS86] P. Maragos and R. Schafer. “Morphological skeleton representation and coding binary images”. In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 34 (1986), pp. 1228–1244.

[NPM97] P. Nunes, F. Pereira, and F. Marques. “Multi-Grid Chain Coding of binary shapes”. In: *Proceedings 1997 International Conference on Image Processing, ICIP*. Vol. 3. IEEE, 1997, pp. 114–117.

[PB13] Pascal Peter and Michael Breuß. “Refined Homotopic Thinning Algorithms and Quality Measures for Skeletonisation Methods”. In: *Innovations for Shape Analysis, Models and Algorithms*. Springer, 2013, pp. 77–92.

[Pet10] P. Peter. “Hamilton-Jacobi Skeletonisationin Image Processing”. Bachelor’s thesis. Saarland University, 2010.

[RA18] Y. A. Raj and P Alli. “Pattern-based Chain Code for Bi-level Shape Image Compression”. In: *Taga Journal of Graphic Technology* 14 (2018), pp. 3069–3080.

[RT05] E. Remy and E. Thiel. “Exact medial axis with euclidean distance”. In: *Image and Vision Computing* 23 (2005), pp. 167–175.

[SB98] D. Shaked and A. Bruckstein. “Pruning Medial Axes”. In: *Computer Vision and Image Understanding* 69 (1998), pp. 156–169.

[SBR07] H. Sánchez-Cruz, E. Bribiesca, and R. Rodríguez-Dagnino. “Efficiency of chain codes to represent binary objects”. In: *Pattern Recognition* 40 (2007), pp. 1660–1674.

[SCE01] A. Skodras, C. Christopoulos, and T. Ebrahimi. “JPEG2000: the new still picture compression standard”. In: *Signal Processing Magazine,IEEE* 18 (2001), pp. 36–58.

[She+11] W. Shen et al. “Skeleton growing and pruning with bending potential ratio”. In: *Pattern Recognition* 44 (2011), pp. 196–209.

[Sid+02] K. Siddiqi et al. “Hamilton-Jacobi Skeletons”. In: *International Journal of Computer Vision* 48 (2002), pp. 215–231.

[SL99] K. Schröder and P. Laurent. “Efficient polygon approximations for shape signatures”. In: *Proceedings of the 1999 International Conference on Image Processing, ICIP*. Vol. 2. IEEE, 1999, pp. 811–814.

[SR05] H. Sánchez-Cruz and R. Rodríguez-Dagnino. "Compressing bi-level images by means of a 3-bit chain code". In: *Optical Engineering* 44 (2005), pp. 1–8.

[TH02] R. Tam and W. Heidrich. "Feature-Preserving Medial Axis Noise Removal". In: *Proceedings of the European Conference on Computer Vision,ECCV*. Vol. 2351. Springer, 2002, pp. 672–686.

[Wal92] G. Wallace. "The JPEG still picture compression standard". In: *Transactions on Consumer Electronics, IEEE* 38 (1992), pp. 18–34.

[WH10] A. Ward and G. Hamarneh. "The Groupwise Medial Axis Transform for Fuzzy Skeletonization and Pruning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2010), pp. 1084–1096.

[ZD07] S. Zahir and K. Dhou. "A new chain coding based method for binary image compression and reconstruction". In: *PCS 2007 - 26th Picture Coding Symposium* (2007), pp. 1321–1324.

[ZL14] B. Zalik and N. Lukac. "Chain code lossless compression using move-to-front transform and adaptive run-length encoding". In: *Signal Processing Image Communication* 29 (2014), pp. 96–106.

[ZML15] B. Zalik, D. Mongus, and N. Lukac. "A universal chain code compression method". In: *Journal of Visual Communication and Image Representation* 29 (2015), pp. 8–15.