



University of Padova

DEPARTMENT OF MATHEMATICS "TULLIO LEVI-CIVITA"

MASTER THESIS IN DATA SCIENCE

hp-Adaptive Image Approximation using Binary Wedge Partitioning Trees

SUPERVISOR

PROF. WOLFGANG ERB
UNIVERSITY OF PADOVA

CO-SUPERVISOR

GIACOMO ELEFANTE
UNIVERSITY OF ITALIAN SWITZERLAND

MASTER CANDIDATE

PAOLO GOTTI
2163420

ACADEMIC YEAR

2025–2026

Abstract

This thesis develops and studies an adaptive algorithm for the lossy compression of signals defined on a metric space, with digital images as its primary application. The method partitions the domain hierarchically and approximates the signal locally by polynomials of variable degree, following the paradigm of hp -adaptivity, which pairs a spatial refinement of the partition (h -adaptivity) with an increase of the local polynomial order (p -adaptivity). The partition is encoded through *Binary Wedge Partitioning* (BWP) trees, a data-adapted, geometrically meaningful representation that costs a single new centre per split. The choice between refining the partition and enriching a local polynomial is governed by Binev’s near-best tree-approximation framework, which we adapt to the multi-channel image setting and complement with a number of practical design choices. We describe the resulting algorithm and its parameters in detail and assess it experimentally on image data.

Contents

ABSTRACT	iii
LIST OF FIGURES	vii
LIST OF TABLES	ix
1 INTRODUCTION	I
2 THEORETICAL FRAMEWORK	3
2.1 Graph Signals and the Metric Space	3
2.2 Binary Wedge Partitioning Trees	5
2.3 Local Polynomial Approximation	7
2.4 <i>hp</i> -Adaptivity and the Binev Approach	9
2.4.1 Complexity, the oracle, and Binev’s two requirements	9
2.4.2 One tree, and the subtree that is returned	10
2.4.3 Selecting the next leaf: modified errors and the trackers	12
2.4.4 The incremental algorithm	14
3 A PROPOSED ALGORITHM: AN <i>hp</i> -EXTENSION OF THE BINEV FRAMEWORK TO IMAGE COMPRESSION	19
3.1 Setup: Initialization, Colour Space and Initial Partition	21
3.2 The Refinement Loop: Which Tree, Which Leaf	27
3.3 The <i>h</i> -step: Splitting a Cell	29
3.4 The <i>p</i> -step: Growing the Polynomials	32
3.5 Stopping Criteria	34
3.6 Extraction and Reassembly	37
4 IMPLEMENTATION AND RESULTS	39
4.1 Implementation and Representation	40
4.2 First-Level Analysis	43
4.2.1 The Reference Configuration, and a First Look	44
4.2.2 The Colour Space	47
4.2.3 The Initial Partition	48
4.2.4 The Refinement Loop: Which Tree, Which Leaf	51
4.2.5 The <i>h</i> -step: How a Cell Is Split, and What It Costs	53

4.2.6	The p -step: Growing the Polynomials	55
4.3	Secondary Analyses and Real Compression	58
4.3.1	The Anatomy of the Representation	59
4.3.2	From Parameters to Bytes	60
4.3.3	Decoding: the Time to Open the File	65
4.4	The Optimal Configuration	66
5	CONCLUSIONS	71
	REFERENCES	75

Listing of figures

2.1	Dyadic splits and the growing BWP tree	6
2.2	The two kinds of subtree	6
2.3	A main tree pruned to its returned subtree	12
3.1	Colour-space transforms of the test image	23
3.2	Initial-partition methods	26
3.3	Per-channel initial partitions	27
3.4	A small partition under different metrics	30
4.1	Progressive reconstruction	46
4.2	Colour-space transform	48
4.3	Initial partition	50
4.4	Tree-selection rule	52
4.5	Centre-selection rule	53
4.6	Distribution of polynomial sizes	60
4.7	Rate–distortion versus JPEG	65
4.8	The optimal configuration	68

Listing of tables

3.1	The parameters of the algorithm.	36
4.1	The reference configuration, from which each first-level study departs one parameter at a time.	45
4.2	Progressive reconstruction: parameters and relative error at each step count. .	47
4.3	Colour-space transform: end-of-run figures on the test image.	48
4.4	Initial partition at ≈ 50 initial regions (per channel for the per-channel variants; test image). Every row is read at the <i>same</i> coefficient budget — the 99 634 coefficients of the free run, Binev’s own measure: each pre-partitioned run is stopped at the first step where its stored coefficients reach that budget, which happens a hundred-odd steps before its 100 000th, since it starts with coefficients already stored. The cells column counts the cells (and hence the centres) each run has built at that point. The time refers to the full run and includes building the partition.	51
4.5	Selection rules: end-of-run figures on the test image.	53
4.6	Number of sampled candidates n (test image, 100 000 steps): more candidates buy quality at a near-proportional cost in time.	54
4.7	Splitting metric d_α : end-of-run figures on the test image. At equal budget the quality differences are within noise ($\alpha = \infty$ marginally the worst); the times differ more, and $\alpha = 2$ is the fastest by construction (its boundary is a line, so assignment is a single sign test).	55
4.8	Monomial-ordering rule: end-of-run figures on the test image. Both adaptive orderings lower the budget and raise the quality; the deterministic one is kept only because it needs no monomial-identity bits (Section 4.3).	56
4.9	Distribution of polynomial size under the three monomial orderings — share of final cells (%) carrying 1, . . . , 6 monomials, and the resulting cell count. Adaptive orderings shift mass onto larger polynomials, lowering the cell (and centre) count.	56
4.10	Initial order k at a matched <i>coefficient</i> budget ($\approx 99\,600$ coefficients, the quantity the near-best theory bounds). A larger k reaches it with far fewer cells — hence fewer centres, and a smaller stored N — but at a <i>higher</i> error: on the modelling budget $k = 1$ is the more efficient, as forcing order $\geq k$ wastes coefficients on the flat majority.	57
4.11	Maximum order $P_{\max}$: end-of-run figures on the test image. Quality barely moves; the time rises steeply.	58

4.12	Estimated byte budget for the reference reconstruction of the test image. The centres are bracketed by the balanced and comb estimates; the coefficients are shape-independent, and are shown both in the plain two-byte accounting and in the refined one (one byte for the constant cells).	64
4.13	Decoding versus encoding on the test image (single thread).	65
4.14	The ladder of levers: the reference, each quality lever alone, and their combinations, at an equal 100 000 steps.	66
4.15	The optimal configuration with and without an initial partition (100 000 steps; the time includes building the partition).	68

1

Introduction

Data compression is a central theme in the digital era. Most digital data, by its nature, carries a large amount of redundant information; removing this redundancy is therefore a primary objective, essential to optimize storage and transmission efficiency. This process inherently requires a trade-off between three competing aspects: information loss, the compression ratio, and the computational speed of the algorithm during both the encoding and decoding phases.

In this thesis, we investigate a data-compression algorithm primarily developed for images but generalizable to any signal defined on a metric space, such as graph signals. The main advantage of the proposed method lies in its ability to exploit the geometric structure of the domain on which the signal is defined, under the reasonable assumption that, on such a metric space, the signal is strongly position-related: nearby points tend to carry similar values. In essence, the algorithm partitions the domain into regions and approximates the signal locally, region by region, with a polynomial. This class of approaches is known as *hp-adaptivity*, an approximation paradigm that combines two complementary techniques: spatial refinement (*h-adaptivity*), which uses increasingly fine partitions to approximate the signal more accurately where it is complex, and local polynomial approximation of increasing complexity (*p-adaptivity*), which raises the degree of the local polynomials to fit the signal better.

To maintain a partition that is both geometrically meaningful and efficiently compressible, we rely on *Binary Wedge Partitioning* (BWP) trees [1], which adapt to graphs the geometric, wedge-based partitioning ideas developed for images by Donoho [2]. These structures allow an extremely compact representation of the partition, favouring splits aligned with the geometric

features of the underlying domain. The price of this adaptivity is computational: relative to a fixed, predetermined partition, such as a regular grid whose cells are known in advance, a BWP tree is more costly to build, since good splits must be searched for during encoding, and, being data-dependent, its partition must also be reconstructed from the stored centres when decoding. This overhead is nonetheless modest, and is repaid by a partition far better matched to the signal.

Furthermore, the algorithm follows a greedy approach, incrementally improving the approximation quality by adding new parameters at each step. To decide whether to prioritize a finer partition or a more complex polynomial for a given region, we employ the theoretical framework proposed by Peter Binev [3]. This framework provides an incremental strategy that ensures near-best approximation performance under specific assumptions on the local errors.

The following chapters are organized as follows: Chapter 2 establishes the mathematical foundations, ranging from graph theory to the Binev framework. Chapter 3 describes the proposed algorithm in detail, including initialization strategies and splitting criteria. Finally, Chapter 4 presents experimental results and performance metrics on image-based datasets.

2

Theoretical Framework

This chapter establishes the mathematical foundations underlying the algorithm of the next chapter. We proceed linearly: from the discrete domain and its metric (Section 2.1), to the binary wedge partitioning trees that organize it (Section 2.2), to the local polynomial approximation error and its two structural properties (Section 2.3), and finally to Binev’s near-best *hp*-adaptive framework (Section 2.4), on which the whole construction rests.

2.1 GRAPH SIGNALS AND THE METRIC SPACE

We work on a discrete domain equipped with a notion of distance, and nothing more. It is convenient to first recall the notion of a weighted graph, standard in the graph signal processing literature [4], from which such a distance naturally arises.

Definition 2.1 (Weighted graph). A *weighted graph* is a triple (V, E, w) where $V = \{v_1, \dots, v_M\}$ is a finite set of vertices, $E \subseteq V \times V$ is a set of edges, and $w : E \rightarrow \mathbb{R}^+$ assigns a positive weight to each edge. We work exclusively with *undirected* graphs, whose edges carry no orientation:

$$(v_1, v_2) \in E \implies (v_2, v_1) \in E. \quad (2.1)$$

The graph induces a distance between its vertices. A *path* joining two vertices u and v is a finite sequence of vertices $u = u_0, u_1, \dots, u_n = v$ in which each consecutive pair (u_i, u_{i+1}) is

an edge of E , and its *length* is the sum of the weights of those edges. On a connected graph the *geodesic distance* $d(u, v)$ is the smallest length among all paths joining u and v . This distance turns V into a metric space.

Definition 2.2 (Metric space). A *metric space* is a pair (V, d) where $V = \{v_1, \dots, v_M\}$ is a finite set of points and $d : V \times V \rightarrow \mathbb{R}^+ \cup \{0\}$ is a *distance function* satisfying, for all $u, v, w \in V$,

- $d(u, v) \geq 0$, with $d(u, v) = 0$ if and only if $u = v$ (non-negativity / identity);
- $d(u, v) = d(v, u)$ (symmetry);
- $d(u, w) \leq d(u, v) + d(v, w)$ (triangle inequality).

On a connected weighted graph these three properties are all satisfied by the geodesic distance defined above, which is therefore a genuine metric.

Definition 2.3 (Signal). A *signal* on V is a function $f : V \rightarrow \mathbb{R}$. Since V is finite, we identify f with the vector $(f(v_1), \dots, f(v_M)) \in \mathbb{R}^M$.

These abstractions apply directly to a digital image, following the graph-wedgelet framework of Erb [1], which we build up in the following order. We regard the image as a *fully-connected graph* whose vertices are the pixels: every pair of pixels is joined by an edge. Writing a pixel by its coordinates $\mathbf{p} = (x, y)$, we weight each edge by the Minkowski-type distance between its two endpoints,

$$d_\alpha(\mathbf{p}_1, \mathbf{p}_2) = \left(|x_1 - x_2|^\alpha + |y_1 - y_2|^\alpha \right)^{1/\alpha}, \quad \alpha \geq 1, \quad (2.2)$$

a genuine metric for every $\alpha \geq 1$. On a fully-connected graph the geodesic distance between two pixels then coincides with the weight of the *single edge* joining them directly: the direct edge is always a shortest path, since by the triangle inequality any detour through other pixels can only be longer. The geodesic distance is therefore just d_α , and we obtain the final metric space (V, d_α) at once, without ever having to build or store the graph. The graph viewpoint is nonetheless valuable, because it shows that the construction is not tied to images: it extends verbatim to any signal defined on a graph endowed with a distance: graph signals in the general sense. The colour content of an RGB image is carried by three such signals, one per channel R, G, B , and the coordinates x, y may themselves be regarded as two further signals on the same vertex set. From here on we work directly with the point set V , its metric d , and the signals on it.

2.2 BINARY WEDGE PARTITIONING TREES

We now describe how the domain V is partitioned hierarchically into cells, using only the metric. A sequence of such splits, together with the BWP tree they build, is illustrated in Figure 2.1.

Definition 2.4 (Dyadic split). Given a cell $\Delta \subseteq V$ of a metric space (V, d) and two distinct elements $c', c'' \in \Delta$, called the *centres*, the *dyadic split* assigns each point of Δ to its nearer centre, producing the bipartition $\{\Delta', \Delta''\}$ with

$$\begin{aligned}\Delta' &:= \{v \in \Delta : d(v, c') \leq d(v, c'')\}, \\ \Delta'' &:= \Delta \setminus \Delta'.\end{aligned}\tag{2.3}$$

Definition 2.5 (BWP tree). A *binary wedge partitioning (BWP) tree* [1] over V is a full binary tree whose root is V and each of whose internal nodes Δ has two children $\{\Delta', \Delta''\}$ obtained by a dyadic split of Δ , for some pair of centres. The set of leaves of the tree forms a partition of V .

A first observation is worth spelling out, since the compactness of the whole representation rests on it. As a BWP tree is grown, every leaf carries an assigned *centre*, where the one for the root is chosen arbitrarily. Splitting a leaf therefore requires selecting only *one* new point: the two children then inherit, respectively, the parent's centre and the newly selected one. Each split thus adds exactly one centre, so a tree with m leaves is described by exactly m centres in total (the initial one plus one per split). Since a single point of V encodes each split, the correctly ordered sequence of these centres is all that is needed to reconstruct the entire tree: an extremely compact encoding of a hierarchical partition.

Two notions of subtree will accompany us throughout, and we introduce them now. The first is the *pruned subtree*. Pruning a BWP tree at an internal node Δ , that is, discarding the two children of Δ and all of their descendants so that Δ becomes a leaf again, yields once more a BWP tree over V , coarser than the original. More generally, pruning at any set of internal nodes gives a BWP tree whose leaves still partition V . This is the operation used to extract the optimal tree in Binev's framework (Section 2.4); to reconstruct such a tree from its centres, one keeps them in their original order of creation, so that each split is replayed on the correct cell.

The second notion is the subtree *rooted at* a node. Dually, taking a node Δ together with all of its descendants, which we denote $\mathcal{T}_\Delta$, gives itself a BWP tree, now over the sub-domain Δ rather than over the whole of V . This downward view is convenient to examine a single cell

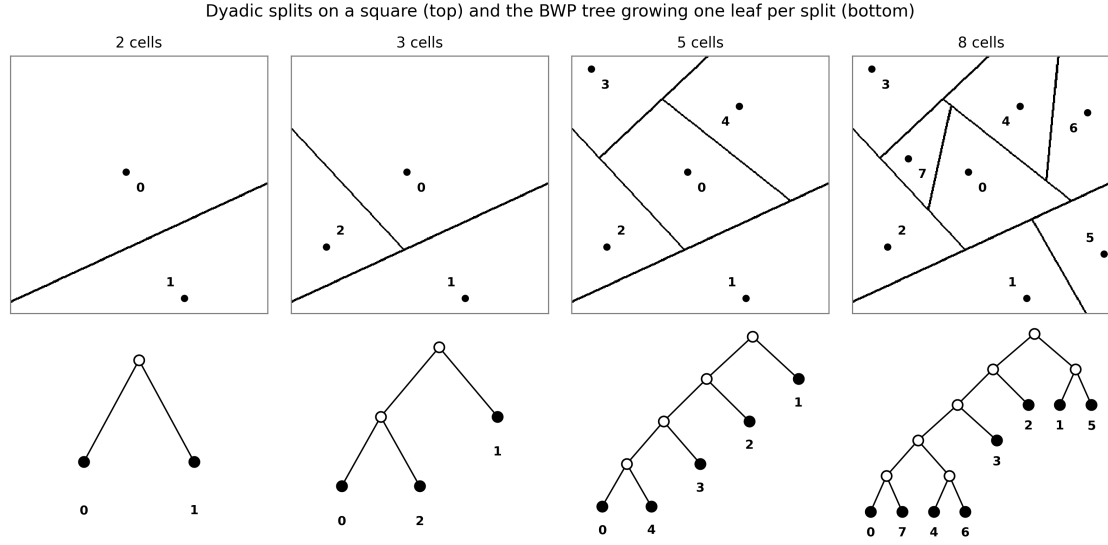


Figure 2.1: *Top:* a partition of a square built by dyadic splits, shown after 2, 3, 5, 8 cells. At each step one existing cell is split by a single new centre (one new centre per split); the centres, once placed, never move, and only the chosen cell is subdivided (with $\alpha = 2$). *Bottom:* the corresponding BWP tree, which grows one leaf per split: internal nodes hollow, leaves (the cells) filled. The number beside each centre labels its cell and matches the corresponding leaf of the tree.

and its own sub-partition in isolation, independently of the rest of the tree. The two notions are distinct (Figure 2.2): pruning looks *upward*, choosing where to stop refining from the root, whereas the rooted subtree $\mathcal{T}_\Delta$ looks *downward*, describing the partition induced inside one cell.

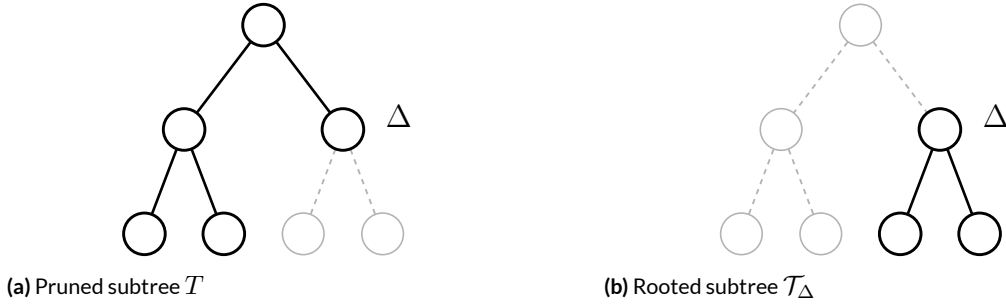


Figure 2.2: Two distinct notions of subtree of the same BWP tree; in each panel the subtree in question is drawn in full black and the rest of the tree is faded. (a) A *pruned* subtree T : branches are cut from the root (here the node Δ is kept as a leaf and its descendants, dashed, are discarded); its leaves form a partition of V . (b) The subtree *rooted* at the same node Δ : the node together with all of its descendants, itself a BWP tree over the cell Δ .

2.3 LOCAL POLYNOMIAL APPROXIMATION

The building block of the framework is the *local approximation error* on a subset (a *cell*) $\Delta \subseteq V$. We have seen that the cell is the basic element on which the signal will be approximated by a local polynomial model; it is useful to note that, on a general graph, a cell is always a *set* of vertices, and the polynomial approximates one specific signal on the basis of other signals defined on those same vertices; in the image setting, these are the two coordinate signals x_v, y_v of the pixels. Nothing in this construction is therefore specific to images: on an arbitrary graph, any choice of vertex signals to serve as coordinates makes the polynomial spaces below well defined, and the whole framework carries over unchanged.

Definition 2.6 (Local approximation spaces). For a cell $\Delta \subseteq V$, let $\mathcal{F}^{(1)}(\Delta) \subset \mathcal{F}^{(2)}(\Delta) \subset \dots$ be a nested sequence of finite-dimensional spaces of functions on Δ , with $\dim \mathcal{F}^{(p)}(\Delta) = p$. We call p the *order* of the local model. Such nested families are the standard setting of non-linear approximation theory [5]. In our setting these spaces are polynomial: we shall fix, in Chapter 3, an *ordering* of the bivariate monomials $x^a y^b$ in the (normalized) cell coordinates, a sequence $m_1, m_2, \dots$, and take $\mathcal{F}^{(p)}(\Delta)$ to be spanned by its first p terms $m_1, \dots, m_p$. The particular ordering is a design choice; the results here depend only on the nesting $\mathcal{F}^{(p)} \subset \mathcal{F}^{(p+1)}$, which holds for any such sequence.

Definition 2.7 (Local error). For an approximant $g \in \mathcal{F}^{(p)}(\Delta)$, its *squared error* on Δ is

$$\mathcal{E}_\Delta(g) := \sum_{v \in \Delta} (f(v) - g(v))^2. \quad (2.4)$$

The optimal order- p approximation is the one that minimizes this error over the whole space,

$$\hat{f}_{\Delta,p} := \arg \min_{g \in \mathcal{F}^{(p)}(\Delta)} \mathcal{E}_\Delta(g), \quad (2.5)$$

that is, the least-squares fit of f on Δ . The *local error of order p* is then the value attained at that minimizer,

$$e_p(\Delta) := \mathcal{E}_\Delta(\hat{f}_{\Delta,p}) = \min_{g \in \mathcal{F}^{(p)}(\Delta)} \sum_{v \in \Delta} (f(v) - g(v))^2. \quad (2.6)$$

The lowest order deserves a word of its own, since it admits a closed form. By definition $\mathcal{F}^{(1)}(\Delta)$ is the space of constant functions on Δ . In this space the optimal approximation of

the signal is simply its mean,

$$\bar{f}_\Delta = \frac{1}{|\Delta|} \sum_{v \in \Delta} f(v), \quad (2.7)$$

so that $\hat{f}_{\Delta,1} \equiv \bar{f}_\Delta$, and the local error is, by definition, the *deviance* of the cell, that is, the total squared deviation of the signal from its mean:

$$e_1(\Delta) = \sum_{v \in \Delta} (f(v) - \hat{f}_{\Delta,1}(v))^2 = \sum_{v \in \Delta} (f(v) - \bar{f}_\Delta)^2 =: \text{Dev}(\Delta) = |\Delta| \cdot \text{Var}(f|_\Delta). \quad (2.8)$$

Thus, for $p \equiv 1$, selecting the cell of maximal local error is the same as selecting the cell of maximal deviance, that is, of maximal size-weighted variance.

The entire framework rests on only two properties of e_p , which we now prove for this construction.

Lemma 2.1 (Subadditivity). If $\Delta = \Delta' \cup \Delta''$ with $\Delta' \cap \Delta'' = \emptyset$, then

$$e_1(\Delta) \geq e_1(\Delta') + e_1(\Delta''). \quad (2.9)$$

Proof. Let $\hat{f}_{\Delta,1}$ be the constant optimal fit on Δ . Since Δ' and Δ'' partition Δ , the error of this fit splits over the two parts:

$$e_1(\Delta) = \sum_{v \in \Delta'} (f(v) - \hat{f}_{\Delta,1}(v))^2 + \sum_{v \in \Delta''} (f(v) - \hat{f}_{\Delta,1}(v))^2. \quad (2.10)$$

Now $\hat{f}_{\Delta,1}$ restricted to Δ' is itself a constant, hence an *admissible* (but not necessarily optimal) candidate on Δ' . As $e_1(\Delta')$ is the minimum over all such constants, it can only be smaller:

$$e_1(\Delta') = \min_{g \in \mathcal{F}^{(1)}(\Delta')} \sum_{v \in \Delta'} (f(v) - g(v))^2 \leq \sum_{v \in \Delta'} (f(v) - \hat{f}_{\Delta,1}(v))^2, \quad (2.11)$$

and the same inequality holds on Δ'' . Substituting both into (2.10) gives $e_1(\Delta) \geq e_1(\Delta') + e_1(\Delta'')$, as claimed. $\square$

Lemma 2.2 (Monotonicity in the order). For every cell Δ and every $k \geq 1$,

$$e_k(\Delta) \geq e_{k+1}(\Delta). \quad (2.12)$$

Proof. By construction the approximation spaces are nested, $\mathcal{F}^{(k)}(\Delta) \subseteq \mathcal{F}^{(k+1)}(\Delta)$. Let

$\hat{f}_{\Delta,k}$ be the order- k optimum. It belongs to $\mathcal{F}^{(k)}(\Delta)$ and hence also to $\mathcal{F}^{(k+1)}(\Delta)$, so it is an admissible candidate for the order- $(k+1)$ problem. Since $e_{k+1}(\Delta)$ is the minimum over that larger space, it cannot exceed the error of any particular candidate, and in particular

$$e_{k+1}(\Delta) = \min_{g \in \mathcal{F}^{(k+1)}(\Delta)} \sum_{v \in \Delta} (f(v) - g(v))^2 \leq \sum_{v \in \Delta} (f(v) - \hat{f}_{\Delta,k}(v))^2 = e_k(\Delta). \quad (2.13)$$

Equivalently, appending one basis function to the fit can only lower the residual, since the previous optimum, with a zero coefficient on the new function, remains feasible. $\square$

Properties (2.9) and (2.12) are exactly conditions (1.2) and (1.3) of Binev [3], and are the *only* assumptions required by the near-best result of Section 2.4.

2.4 *hp*-ADAPTIVITY AND THE BINEV APPROACH

Suppose we want to approximate a target by a *piecewise* model over a domain. There are two independent ways to make such a model richer:

- refine the *partition* into smaller cells, so each local model covers a smaller region;
- increase the *complexity* (the order) of the local model on a cell, so a richer function is used on the same region.

Strategies that combine both are called *hp-adaptive*. A difficulty absent from single-refinement problems appears immediately: a greedy procedure must now also decide, at each step, *which* of the two refinements to make, so that even selecting a good next refinement is no longer automatic. An algorithmic approach that finds a near-best solution to this kind of problem was proposed by Peter Binev [3]; we summarize his method and results here, and in Chapter 3 we implement and adapt them to our purposes. In our setting a cell is a subset $\Delta \subseteq V$ and the local model the order- p polynomial of Section 2.3.

2.4.1 COMPLEXITY, THE ORACLE, AND BINEV'S TWO REQUIREMENTS

Throughout, we denote by $\mathcal{L}(T)$ the set of leaves of a tree T and by $\mathcal{P}(T) := (p(\Delta))_{\Delta \in \mathcal{L}(T)}$ the sequence of polynomial orders assigned to those leaves.

Given a partition by a BWP tree together with an order assignment $\mathcal{P}(T)$, we define the *total error* of the resulting piecewise-polynomial model as the sum of the local errors over the final

cells [3, eq. (1.1)],

$$\mathcal{E}(T, \mathcal{P}(T)) := \sum_{\Delta \in \mathcal{L}(T)} e_{\mathcal{P}(\Delta)}(\Delta). \quad (2.14)$$

Its *complexity*, or total parameter count, written with the counting symbol $\#$, which we shall later also use for the number of pixels of a cell, $\#\Delta$, is

$$\#\mathcal{P}(T) := \sum_{\Delta \in \mathcal{L}(T)} p(\Delta), \quad (2.15)$$

the total memory footprint of the representation, a mild abuse of the counting symbol, since applied to the partition $\#$ totals its parameters rather than counting its cells: each centre produced by a partition refinement and each coefficient produced by a complexity increase contributes exactly one unit.

Given a budget N , the *best* hp -adaptive approximation error of complexity N is the oracle quantity [3, eq. (1.5)]

$$\sigma_N := \inf_T \inf_{\#\mathcal{P}(T) \leq N} \mathcal{E}(T, \mathcal{P}(T)). \quad (2.16)$$

The aim is an *incremental* algorithm producing, for every N , a concrete pair $(T_N, \mathcal{P}(T_N))$ whose error stays provably close to σ_n for *every* $n \leq N$, not merely for $n = N$. This “for every $n \leq N$ ” is what makes the guarantee powerful: a single run yields a whole *family* of approximations, each near-best at its own size, so one need not re-run the algorithm to obtain a coarser, smaller-budget representation: it is already contained, near-optimally, in the same construction.

The near-best guarantee (Theorem 2.1) rests on exactly the two properties of the local error established above: subadditivity (Lemma 2.1) and monotonicity (Lemma 2.2). It is essential to note what Binev’s framework does *not* prescribe: it does not say *how* a cell, once selected for refinement, is split into two: it requires only that a cell *can* be bipartitioned. The concrete geometric rule for that bipartition is a design choice introduced with our algorithm (Chapter 3); *which* cell to split, by contrast, is governed by Binev’s own selection, described in what follows. Throughout this section, “split” means an abstract binary subdivision.

2.4.2 ONE TREE, AND THE SUBTREE THAT IS RETURNED

The difficulty raised above, namely that a greedy would have to choose between the two kinds of refinement, is dissolved by Binev’s central device: encode both refinements as the growth of

a *single* tree, so that the algorithm only ever “adds a leaf” and never explicitly chooses between them.

The algorithm builds one large *main tree* $\mathcal{T}$: at each step it adds a single leaf, by splitting one of the existing leaves. Its output, however, is not the whole tree but a subtree $T \subset \mathcal{T}$ obtained by pruning the branches at optimal vertices (as we will see shortly). It is worth fixing, once and for all, the three trees we shall keep referring to:

- $\mathcal{T}$, the *main tree*, grown one leaf at a time by the algorithm;
- T , the *returned subtree*, obtained by pruning $\mathcal{T}$ while keeping its root ($T \subset \mathcal{T}$, itself a BWP tree, as noted in Section 2.2); its leaves are the final cells;
- $\mathcal{T}_\Delta$, the subtree *rooted at* a node Δ , i.e. Δ together with all of its descendants (Section 2.2).

A pruned leaf carries the information on the polynomial order to use on its cell, through the number of main-tree leaves lying below it [3, eq. (1.6)],

$$P(\Delta, \mathcal{T}) := \#(\mathcal{L}(\mathcal{T}) \cap \mathcal{T}_\Delta) = \#\mathcal{L}(\mathcal{T}_\Delta), \quad (2.17)$$

that is, the number of leaves in the rooted subtree $\mathcal{T}_\Delta$; when the main tree is clear from the context, we abbreviate it to $P(\Delta)$. Thus each cell receives a number of polynomial degrees of freedom equal to the total number of leaves in its rooted subtree: those leaves do not represent finer spatial cells, but the degrees of freedom of the polynomial on Δ . As we will see, the algorithm automatically locates the optimal points at which to prune the tree so as to reach the near-best result; Figure 2.3 shows a main tree pruned to its returned subtree.

This reformulation is completely general: *any* *hp*-partition can be described by such a pair, a main tree $\mathcal{T}$ and a subtree $T \subset \mathcal{T}$ read off through the order rule (2.17), so that the whole best-approximation problem (2.16) can be rewritten in terms of the two trees alone. Writing $\mathcal{E}(\mathcal{T}, T) := \mathcal{E}(T, \mathcal{P}(T))$ for the total error of a pruned subtree, with the orders on $\mathcal{L}(T)$ read from $\mathcal{T}$ via (2.17),

$$\sigma_N = \min_{\mathcal{T} : \#\mathcal{L}(\mathcal{T}) \leq N} \min_{T \subset \mathcal{T}} \mathcal{E}(\mathcal{T}, T) \quad [3, \text{eq. (1.7)}]. \quad (2.18)$$

So the algorithm grows a single main tree $\mathcal{T}_N$ one leaf at a time, then extracts the best pruned subtree $T_N \subset \mathcal{T}_N$. The returned sequence of pairs $(\mathcal{T}_N, T_N)$ is near-best simultaneously for *all* intermediate budgets $n \leq N$ (Theorem 2.1). (This two-tree rewriting is the device on which

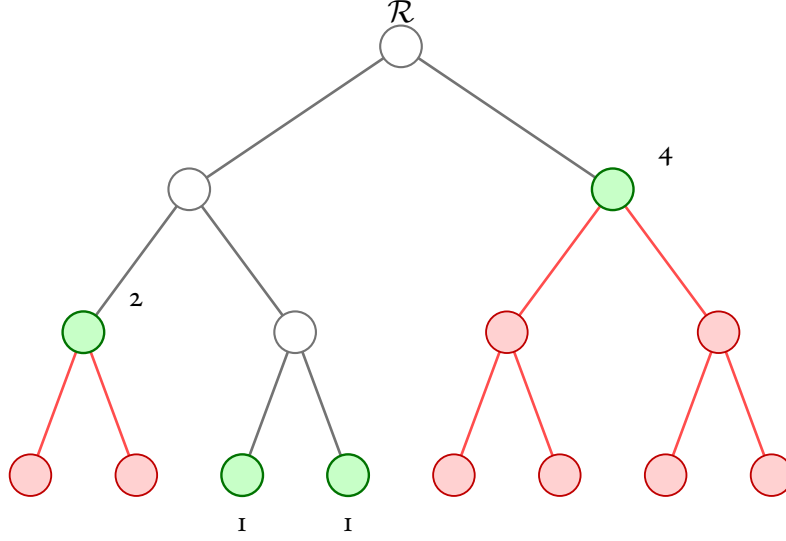


Figure 2.3: A main BWP tree $\mathcal{T}$ (root $\mathcal{R}$) pruned to the returned subtree T . The *green* nodes are the leaves of T , i.e. the final cells; the number beside each is its order $P(\Delta, \mathcal{T})$ (2.17), the count of main-tree leaves lying below it. The *red* nodes and edges are the parts discarded by the pruning: each green leaf Δ together with the red subtree hanging below it is one rooted subtree $\mathcal{T}_\Delta$. Here the four final cells carry orders 2, 1, 1, 4.

Binev builds his proof; since we do not reproduce that proof here, we use it only as a compact way to state the goal.)

2.4.3 SELECTING THE NEXT LEAF: MODIFIED ERRORS AND THE TRACKERS

Binev’s framework thus reduces the whole *hp*-adaptivity problem to a *classic greedy* structure. By growing a single tree and deferring the extraction of the optimal subtree to the end, the process unfolds along one direction only: at each step, choose the leaf to split. This is far from a trivial choice. The obvious greedy rule of expanding the leaf of largest current local error gives no guarantee: a single localized feature (a sharp edge, an isolated high-contrast pixel) keeps exhibiting the largest local error split after split, so the algorithm keeps refining that one feature and neglects the rest of the domain, without an *essential* reduction of the global error. What we really want to estimate, for each leaf, is not its current error but the *potential* error reduction still attainable by refining it further. Binev captures this through a small set of interdependent quantities, which we now introduce in turn.

MODIFIED h -ERROR $\tilde{e}(\Delta)$. The way to avoid fixating on a leaf that keeps showing a large error simply because it is intrinsically hard to approximate is to *further penalize* cells on which

many refinements have already been spent with little payoff. Concretely, the modified h -error attenuates the raw error $e_1(\Delta)$ by the errors already met along the path from the root: with Δ^* the parent of Δ [3, eqs. (2.3)–(2.4)],

$$\begin{aligned} \tilde{e}(\mathcal{R}) &:= e_1(\mathcal{R}), \\ \tilde{e}(\Delta) &:= \frac{e_1(\Delta) \tilde{e}(\Delta^*)}{e_1(\Delta) + \tilde{e}(\Delta^*)} \quad \Longleftrightarrow \quad \frac{1}{\tilde{e}(\Delta)} = \frac{1}{e_1(\Delta)} + \frac{1}{\tilde{e}(\Delta^*)}, \end{aligned} \quad (2.19)$$

with $\tilde{e}(\Delta) := 0$ when both terms vanish. Unrolling along the ancestors $\mathcal{A}(\Delta)$ [3, eq. (2.7)],

$$\frac{1}{\tilde{e}(\Delta)} = \sum_{\Delta' \in \mathcal{A}(\Delta) \cup \{\Delta\}} \frac{1}{e_1(\Delta')}, \quad (2.20)$$

so $\tilde{e}(\Delta)$ is the harmonic sum of the local errors along the path from the root to Δ : it is at most $e_1(\Delta)$ and decreases with depth, so a cell reached through many refinements is strongly discounted.

RECURSIVE SUBTREE ERROR $E_j(\Delta)$. Because the algorithm ultimately returns only a *subtree*, the genuine error attributable to a cell is not its own approximation error. That error would be the correct measure only if the final subtree kept the cell whole; if instead the cell were split, the error truly attributable to its region would be that of its two children combined. Applying this reasoning recursively along the tree yields the *recursive subtree error*: the best error attainable on the subtree rooted at Δ using a total budget j . With $E_1(\Delta) := e_1(\Delta)$ on leaves, and for a node with children Δ', Δ'' [3, eq. (3.1)],

$$E_{P(\Delta)}(\Delta) := \min \left\{ E_{P(\Delta')}(\Delta') + E_{P(\Delta'')}(\Delta''), e_{P(\Delta)}(\Delta) \right\}. \quad (2.21)$$

The min records whether it is better to keep Δ whole (right term) or to subdivide it (left term), which, as we will see, is precisely the criterion by which the tree is later pruned.

MODIFIED hp -ERROR $\tilde{E}_j(\Delta)$. Combining the two ideas above, the depth discounting of $\tilde{e}$ and the recursive subtree error E_j , gives the main error measure the algorithm uses to rank

leaves. With $\tilde{E}_1(\Delta) := \tilde{e}(\Delta)$, and for $j > 1$ [3, eqs. (3.2)–(3.3)],

$$\begin{aligned}\tilde{E}_j(\Delta) &:= \frac{E_j(\Delta) \tilde{E}_{j-1}(\Delta)}{E_j(\Delta) + \tilde{E}_{j-1}(\Delta)} \\ \iff \frac{1}{\tilde{E}_j(\Delta)} &= \sum_{i=1}^j \frac{1}{E_i(\Delta)} + \sum_{\Delta' \in \mathcal{A}(\Delta)} \frac{1}{e_1(\Delta')},\end{aligned}\tag{2.22}$$

the same harmonic discounting as before, now aware of both kinds of refinement.

Besides these discounted errors, Binev defines two functions that dynamically track the worst error still reachable in a subtree and the leaf that realizes it. They are worth stating carefully: when we later introduce our own leaf-selection rules (Chapter 3), we will define them formally as modifications of exactly these two functions, so as to keep the treatment rigorous.

SELECTION FUNCTIONS $q(\Delta)$ AND $s(\Delta)$. On leaves [3, eq. (3.4)], $q(\Delta) := \tilde{E}_1(\Delta) = \tilde{e}(\Delta)$ and $s(\Delta) := \Delta$; recursively, for an internal node with children Δ', Δ'' [3, eq. (3.5)],

$$q(\Delta) := \min \left\{ \max\{q(\Delta'), q(\Delta'')\}, \tilde{E}_{P(\Delta)}(\Delta) \right\},\tag{2.23}$$

$$s(\Delta) := s(\arg \max\{q(\Delta'), q(\Delta'')\}).\tag{2.24}$$

Here $q(\Delta)$ is the largest modified error still reachable inside the subtree rooted at Δ , and $s(\Delta)$ points to the leaf that realizes it; propagating both quantities up to the root, $q(\mathcal{R})$ is the priority of the single globally most promising leaf and $s(\mathcal{R})$ is that leaf: the one the algorithm expands next. The min against $\tilde{E}_{P(\Delta)}(\Delta)$ in (2.23) is what caps the priority of a branch already well served by refinement, so that the greedy does not keep pouring effort into it.

2.4.4 THE INCREMENTAL ALGORITHM

Putting everything together, we can now summarize Binev’s algorithm in words. At each step the most promising leaf $s(\mathcal{R})$ is selected and split into two children; the two children are initialized (their order-1 error and discounted error $\tilde{e}$); then the tree is *climbed* from the split node back to the root, at each ancestor raising the order and re-fitting its polynomial, recomputing the corrected errors E and $\tilde{E}$, and refreshing the trackers q, s . Only this single root-to-node path is touched per step. Algorithm 2.1 states the procedure verbatim (following [3, steps (i)–(xiv)]); we then comment it.

Algorithm 2.1 Binev's incremental *hp*-algorithm

```

1: (i)  $\mathcal{T}_1 := \{\mathcal{R}\}$ ;  $\tilde{e}(\mathcal{R}) := e_1(\mathcal{R})$ ;  $E_1(\mathcal{R}) := e_1(\mathcal{R})$ ;  $\tilde{E}_1(\mathcal{R}) := \tilde{e}(\mathcal{R})$ ;  $q(\mathcal{R}) := \tilde{e}(\mathcal{R})$ ;
    $s(\mathcal{R}) := \mathcal{R}$ ;  $P(\mathcal{R}) := 1$ 
2: for  $N = 1$  to  $N_{\max} - 1$  do ▷ (ii)
3:   (iii) expand  $\mathcal{T}_N$  to  $\mathcal{T}_{N+1}$  by splitting  $\Delta_N := s(\mathcal{R})$  into two children  $\Delta'_N, \Delta''_N$ 
4:   for  $\nabla \in \{\Delta'_N, \Delta''_N\}$  with parent  $\Delta_N$  do ▷ (iv)
5:      $\tilde{e}(\nabla) := \frac{e_1(\nabla) \tilde{e}(\Delta_N)}{e_1(\nabla) + \tilde{e}(\Delta_N)}$ ;  $E_1(\nabla) := e_1(\nabla)$ ;  $\tilde{E}_1(\nabla) := \tilde{e}(\nabla)$ ;  $q(\nabla) :=$ 
        $\tilde{e}(\nabla)$ ;  $s(\nabla) := \nabla$ ;  $P(\nabla) := 1$ 
6:   end for
7:   (v)  $\Delta := \Delta_N$ 
8:   while  $\Delta \neq \emptyset$  do ▷ (vi)
9:     (vii)  $P(\Delta) := P(\Delta) + 1$ ; compute  $e_{P(\Delta)}(\Delta)$ 
10:    (viii) let  $\Delta', \Delta''$  be the children of  $\Delta$ 
11:    (ix)  $E_{P(\Delta)}(\Delta) := \min\{E_{P(\Delta')}(\Delta') + E_{P(\Delta'')}(\Delta''), e_{P(\Delta)}(\Delta)\}$ 
12:    (x)  $\tilde{E}_{P(\Delta)}(\Delta) := \frac{E_{P(\Delta)}(\Delta) \tilde{E}_{P(\Delta)-1}(\Delta)}{E_{P(\Delta)}(\Delta) + \tilde{E}_{P(\Delta)-1}(\Delta)}$ 
13:    (xi)  $\mathcal{D} := \arg \max\{q(\Delta'), q(\Delta'')\}$ ;  $q(\Delta) := \min\{q(\mathcal{D}), \tilde{E}_{P(\Delta)}(\Delta)\}$ ;  $s(\Delta) :=$ 
        $s(\mathcal{D})$ 
14:    (xii) replace  $\Delta$  with its parent (or  $\emptyset$  if  $\Delta = \mathcal{R}$ )
15:   end while
16: end for

```

The leaf to expand is always $s(\mathcal{R})$; it is split into two children, which is the partition-refinement step. Each new leaf receives its order-1 error and, via (2.19), its modified error discounted by the parent's. On the climb from the split node to the root, splitting a leaf adds one leaf to the subtree of *every* ancestor, so every ancestor's order rises by one: this is the *implicit complexity-increase step* (vii), after which E , $\tilde{E}$, and q , s are updated by (2.21), (2.22), (2.23)–(2.24). Neither kind of refinement is ever chosen explicitly. The backward update touches one root-to-node path, so it costs $O(\text{depth})$ per step; growing $\mathcal{T}_N$ costs $O(\sum_{\Delta \in \mathcal{T}_N} P(\Delta, \mathcal{T}_N))$ [3, Lemma 3.2], between $O(N \log N)$ and $O(N^2)$.

EXTRACTING THE OPTIMAL SUBTREE. The main tree $\mathcal{T}_N$ encodes many candidate hp -partitions; the one returned is the best pruned subtree $T_N \subset \mathcal{T}_N$ (Section 2.2), obtained by trimming. A node Δ is declared a leaf of T_N (and its descendants discarded) exactly when

$$E_{P(\Delta)}(\Delta) = e_{P(\Delta)}(\Delta), \quad (2.25)$$

i.e. when the single order- $P(\Delta)$ model on Δ is at least as good as the best combination of models on its descendants, the case in which the min of (2.21) kept Δ whole. Note that this decision is taken on the *true* recursive error E , never on the penalized $\tilde{E}$: the modified errors serve only to *steer the leaf selection* during growth, by discounting over-refined branches, and carry no qualitative meaning for approximation quality, so that the extraction reads the genuine error.

THE NEAR-BEST GUARANTEE.

Theorem 2.1 (Near-best hp -approximation; [3, Thms. 1.2/3.3]). Let the local errors satisfy Lemmas 2.1 and 2.2. Then the pair $(\mathcal{T}_N, T_N)$ produced by the incremental algorithm of Section 2.4.4 satisfies, for every integer $1 \leq n \leq N$,

$$\mathcal{E}(\mathcal{T}_N, T_N) \leq \frac{2N-1}{N-n+1} \sigma_n, \quad (2.26)$$

where σ_n is the best hp -adaptive approximation error of complexity n from (2.16).

We omit the proof, which bounds σ_n from below and $\mathcal{E}(\mathcal{T}_N, T_N)$ from above in terms of the root threshold $q_N := q(\mathcal{R})$ (see [3, §3]). The bound (2.26) is strong: for $N \gg n$ the factor $\frac{2N-1}{N-n+1}$ approaches 2, so the constructed approximation is within roughly a factor of

two of the best possible approximation using only $n \leq N$ parameters, simultaneously for every intermediate budget n . The guarantee relies on nothing beyond Lemmas 2.1 and 2.2.

We close with a forward pointer. Our implementation may initialize each new leaf directly at a fixed order $k \geq 1$ rather than 1. The extension of the framework to this case, namely the validity of the hypotheses, the budget accounting on the coarser sequence $\{k, 2k, \dots\}$, and the initialization of $\tilde{E}_k$, is developed as a design choice in Chapter 3.

3

A Proposed Algorithm: an *hp*-Extension of the Binev Framework to Image Compression

The algorithm developed here compresses an image by *partitioning* it into regions and approximating the signal *locally*, on each region, by polynomial functions of the pixel coordinates, following the *hp*-adaptive framework of Binev (Chapter 2) with several deliberate adaptations. Binev’s framework is abstract in three respects that a concrete algorithm must resolve, and each corresponds to a different kind of design choice:

- some points are *left open* by Binev and it is up to us to fill them, respecting his logic — for instance *how* a cell is split into two, or *which* monomial to add when a polynomial grows. Such a choice *completes* the framework;
- some choices are genuine *modifications*: departures from Binev’s rules, introduced to test and empirically verify whether a different approach can bring practical improvements — for instance capping the polynomial order, or selecting the leaf to split by a rule other than his;
- some aspects Binev does not treat at all but which matter to us — chiefly that we approximate several signals at once (the *R*, *G*, *B* channels). These *extend* the framework to a setting it was not designed for.

A fourth, smaller category is *representational*: choices that change only the encoding of the result, not the decisions the algorithm makes — for instance the colour space in which the signals are approximated, or the normalization of the polynomial coefficients. Finally, the *stopping*

criteria decide *when* to stop, not *how* to refine: we may want to halt not only at a fixed budget, but also when a target error or a target compression is reached. The rest of the chapter walks the algorithm in the order a run executes it, stating for each parameter its category and whether it acts on the partition (an *h-step*) or on the polynomial (a *p-step*). We keep the notation of Chapter 2 and write $\mathcal{S}(\Delta; c', c'') = \{\Delta', \Delta''\}$ for the dyadic split of a cell Δ with centres c', c'' (Definition 2.4).

We first give the proposed algorithm in full (Algorithm 3.1), in a deliberately generic form; it builds on Binev's incremental *hp*-algorithm (Section 2.4.4) with several adaptations, and each of its degrees of freedom — the parameters written in italics — is defined and discussed, one at a time, in the order a run encounters them through the rest of the chapter.

Algorithm 3.1 The proposed algorithm

- 1: convert the image $\mathcal{I}$ into the forest $\{\mathcal{T}^{(c)}\}$ *according to the colour transform and the initial partition*
 - 2: while no stopping criterion is reached (parameter budget, step budget, or error threshold) do
 - 3: select the tree to advance, $c^* \in \mathcal{I}$, *according to the tree-selection rule*
 - 4: select the leaf $\Delta^* := s(\mathcal{R}^{(c^*)})$, or an alternative, *according to the leaf-selection rule*
 - 5: split Δ^* into two children, $\{\Delta', \Delta''\} := \mathcal{S}(\Delta^*; c', c'')$, *according to the centre-selection rule and the metric α*
 - 6: fit the two children $\hat{f}_{\Delta'}, \hat{f}_{\Delta''}$ at order k *according to the initial order*
 - 7: $\Delta := \Delta^*$
 - 8: while $\Delta \neq \emptyset$ do
 - 9: raise the order of Δ , enlarging its polynomial *according to the monomial-ordering rule*, and update $(e, E, \tilde{E}, q, s)$; the order and the model are updated only while below the maximum order $P_{\max}$, beyond which only the errors change
 - 10: $\Delta := \text{parent of } \Delta$
 - 11: end while
 - 12: end while
 - 13: for every tree extract the subtree T with $\Delta \in \mathcal{L}(T) \iff E_{P(\Delta)}(\Delta) = e_{P(\Delta)}(\Delta)$
 - 14: return the leaves $\mathcal{L}(T)$ of each tree and the coefficients of their polynomials
-

3.1 SETUP: INITIALIZATION, COLOUR SPACE AND INITIAL PARTITION

An image is a set of signals defined on the same pixel grid — one per colour channel. By the framework of Chapter 2, each such signal can be approximated *independently* as a BWP tree grown by Binev’s algorithm (Section 2.4). Moreover, as we will see, we may also wish to *pre-partition* the image before starting, so that several regions are refined in parallel; each region of each signal is then grown as its own tree. In either case the image is represented by a *set* of trees — the different signals over the (possibly several) partitions — which together form a *forest*:

$$\mathcal{I} = \{\mathcal{T}^{(c)}\}_c, \quad (3.1)$$

indexed by c over the channels (or, when a pre-partition is used, over channel $\times$ region), with roots $\mathcal{R}^{(c)}$. The trees are coupled only through the shared budget $N = \sum_c \#\mathcal{P}(T^{(c)})$.

We now present the parameters of the algorithm one at a time, in the order a run encounters them, describing for each its admissible *levels*. The two parameters of this section act at *setup* time, before the refinement loop begins: the colour space in which the signals are approximated, and the initial partition from which the trees are grown.

COLOUR-SPACE TRANSFORM

The three channels of a natural image are strongly correlated, so approximating R, G, B separately wastes budget on redundant signal. We therefore apply a reversible linear transform to the colour of each pixel *before* the approximation, fitted once and inverted at reconstruction. Writing X_{rgb} for the original colour and $\tilde{X}$ for the transformed one,

$$\tilde{X} = W X_{\text{rgb}} + b, \quad X_{\text{rgb}} = W^{-1}(\tilde{X} - b), \quad (3.2)$$

for a transform matrix W and an offset b . The two instances we consider are shown in Figure 3.1.

- *Principal-component transform (PCA)*. Here W is the matrix of eigenvectors of the colour covariance $\text{Cov}(X_{\text{rgb}})$ [6, 7], ordered by decreasing eigenvalue (with the offset $b = -W\mu$ subtracting the mean colour μ); this rotates the colour space onto its axes of maximal variance and is optimal in the decorrelation sense. Being data-dependent, it must be stored — W (k^2 numbers) and μ (k numbers), a fixed overhead $k^2 + k$ for k channels. A key property is that W is *orthonormal* ($W^{-1} = W^\top$), so the transform preserves

the total squared error. Denoting by $\hat{e}_v \in \mathbb{R}^k$ the reconstruction error at pixel v in the transformed domain, the error in R, G, B is $\hat{e}_v W^\top$, and

$$\|e\|_{RGB}^2 = \sum_v \|\hat{e}_v W^\top\|^2 = \sum_v \hat{e}_v W^\top W \hat{e}_v^\top = \sum_v \|\hat{e}_v\|^2 = \|\hat{e}\|_Y^2. \quad (3.3)$$

This matters both theoretically and practically: theoretically, when we study the algorithm's results the error it reports is the true error; practically, the quantity the algorithm minimizes in the transformed domain coincides exactly with the reconstruction error we obtain in R, G, B , independently of the transform used.

- *Luminance-chrominance transform (YCbCr)*. Here W and b are fixed, non-data-dependent (the ITU-R BT.601 standard [8]), mapping the three colours to one luminance Y and two chrominance signals Cb, Cr :

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \end{pmatrix}, \quad (3.4)$$

where Y is a weighted average of the channels (the perceived brightness) and Cb, Cr measure the blue and red departures from it. The backward pass subtracts the offset and applies the inverse of the matrix. Mathematically this transform has little value in our framework: the matrix is *not* orthonormal, so it does *not* preserve the error (the minimized error no longer equals the true R, G, B error). It is nonetheless the de-facto standard for representing images in this domain, and is included for completeness and comparison.

Colour-space transforms: channels of the same image

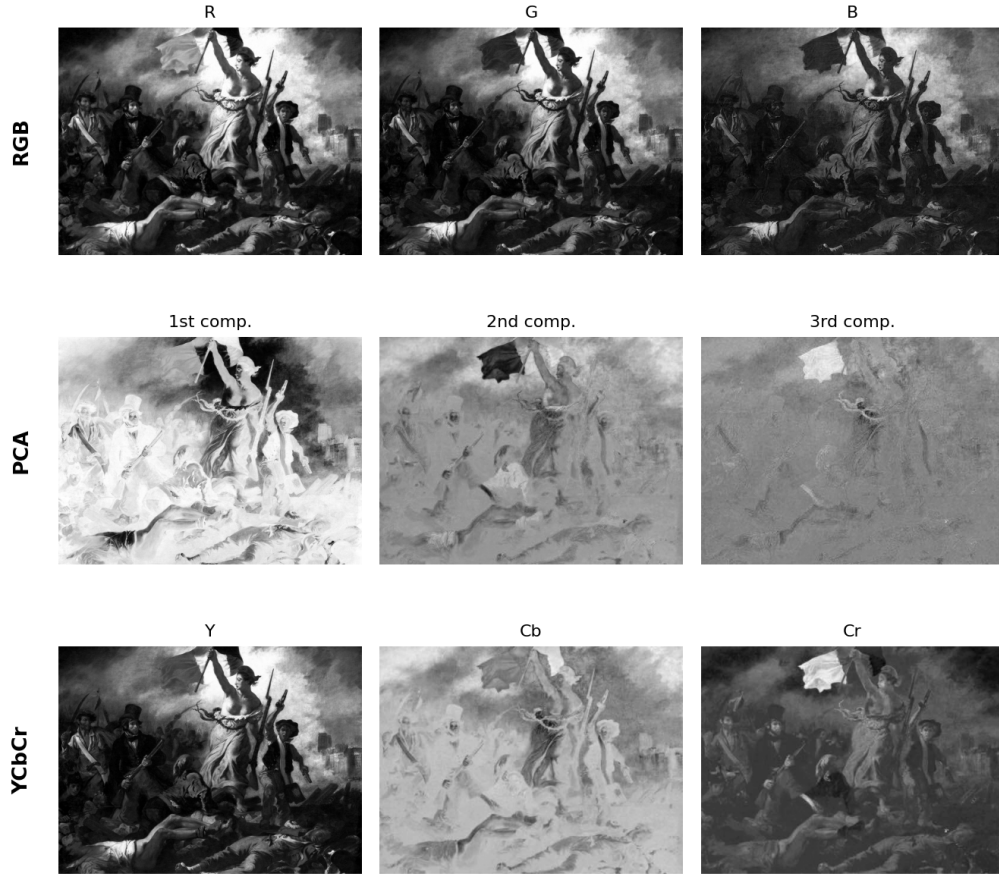


Figure 3.1: The three channels of the test image in each colour space: the raw R , G , B (top), the PCA components (middle), and the YCbCr components (bottom). Each channel is a scalar signal, so all are shown in greyscale. Both transforms concentrate most of the energy in the first component and leave the other two nearly flat — the decorrelation that makes the channels cheaper to approximate. Test image: Delacroix, *La Liberté guidant le peuple* (1830), public domain, Wikimedia Commons.

INITIAL PARTITION

Binev’s algorithm starts every tree from the whole image and lets it partition itself, one split at a time; this is optimal, because the near-best guarantee (Theorem 2.1) holds for the single tree grown from the whole domain. We may instead *pre-partition* the image into blocks $\{V_1, \dots, V_k\}$ and grow one tree per block, in parallel. This *lowers the approximation quality*: instead of one global near-best approximation of the whole image, we obtain k separate near-best approximations, one per block, and their union over a *fixed* partition is in general worse than the single

global optimum — the algorithm can no longer shift budget freely across the imposed block boundaries. In exchange it *speeds up the run considerably*, since the independent blocks can be processed in parallel and each starts from a smaller domain. Some of the methods below do adapt to the signal, and can already group the image sensibly; but none of them is *natively* aligned with the algorithm’s own splitting rule, so the imposed cuts are never quite the ones the algorithm itself would have made. The extra information needed to reconstruct the initial partition counts against the budget N . The methods, from least to most adaptive, are shown in Figure 3.2.

- *Regular blocks* — a uniform $k \times k$ grid of rectangles. It is the simplest and cheapest option: no computation is needed to build the partition, and the only information to store is the grid resolution (a single integer). In return it is entirely content-blind — it ignores the signal and its distribution, so it cannot favour the algorithm in any way.
- *k-means clustering* [9] — k -means partitions a set of feature vectors into k groups by alternately assigning each point to its nearest of k centroids and recomputing each centroid as the mean of its group, minimizing

$$\min_{\{\mu_i\}} \sum_{v \in V} \min_i \|\phi(v) - \mu_i\|^2, \quad (3.5)$$

for a per-pixel feature $\phi(v)$. The choice of feature gives three variants. On the pixel coordinates alone, $\phi(v) = (x_v, y_v)$ (*spatial*), it yields regions roughly uniform in size, blind to colour but cheap. On the augmented feature $\phi(v) = (x_v, y_v, w f(v))$ (*joint spatial-colour*) it groups chromatically similar — hence easily approximated — regions, at the price of more irregular blocks, the centroids then projected back onto the coordinate plane. Coordinates and signal live on very different raw scales — pixel indices run to the thousands, whereas a channel value is a single byte in $[0, 255]$ — so we first normalize both to $[0, 1]$ (the coordinates as $x_v/W, y_v/H$, the signal as $f(v)/255$) before clustering, which already puts geometry and colour on a common footing; the weight w then tilts that balance, larger values favouring colour-homogeneous regions and smaller ones spatially compact ones. A *per-channel* variant uses $\phi(v) = (x_v, y_v, C(v))$, clustering each channel C independently on its own values. The cost is $O(k M)$ per iteration.

- *Lloyd relaxation* [10] — Lloyd’s algorithm runs the same assign–update loop as k -means, but the update is a *density-weighted* mean,

$$\mu_i \leftarrow \frac{\sum_{v \in C_i} \rho(v) \mathbf{p}_v}{\sum_{v \in C_i} \rho(v)}, \quad \rho(v) \propto \|\nabla f(v)\|, \quad (3.6)$$

with the density ρ following the image gradient. This is the crucial difference from k -means: whereas k -means clusters the signal *values*, Lloyd works explicitly on the signal's *spatial density of variation*, drawing centroids towards high-gradient regions. The expected result is a partition whose cells carry a roughly homogeneous amount of signal detail — small where the image varies most, large where it is flat. A further advantage is that no feature weight is needed: the density term balances geometry against content automatically. As for k -means, a per-channel variant drives the density from each channel's own gradient. Its cost is that of k -means times the number of relaxation iterations.

Each of these methods introduces sub-parameters, already glimpsed above: the number of regions k , the feature weight w of the joint k -means, and the number of Lloyd iterations. A word on cost is in order. The adaptive methods place effort where the signal is complex, but produce irregular boundaries that are more expensive to store than a regular grid. The per-channel variants are even more specific — each channel gets a partition tailored to its own content — but this specificity has a price: a separate partition must be stored for *every* signal, rather than one shared across the channels, which is considerably more costly. The per-channel variants are shown in Figure 3.3.

Initial partitions on the same image (shared across channels)

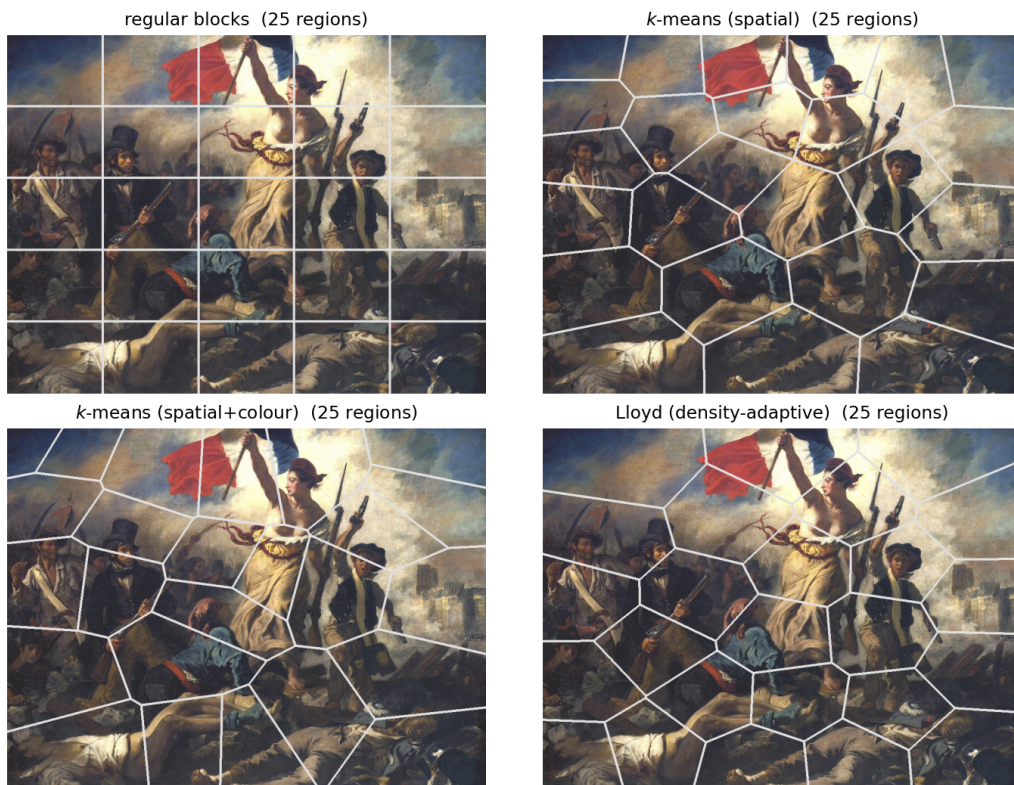


Figure 3.2: The four initial-partition methods on the same image, each into 25 regions: regular blocks, spatial k -means, joint spatial–colour k -means, and density-adaptive Lloyd. The adaptive methods place small cells where the image is complex and large cells in flat regions.

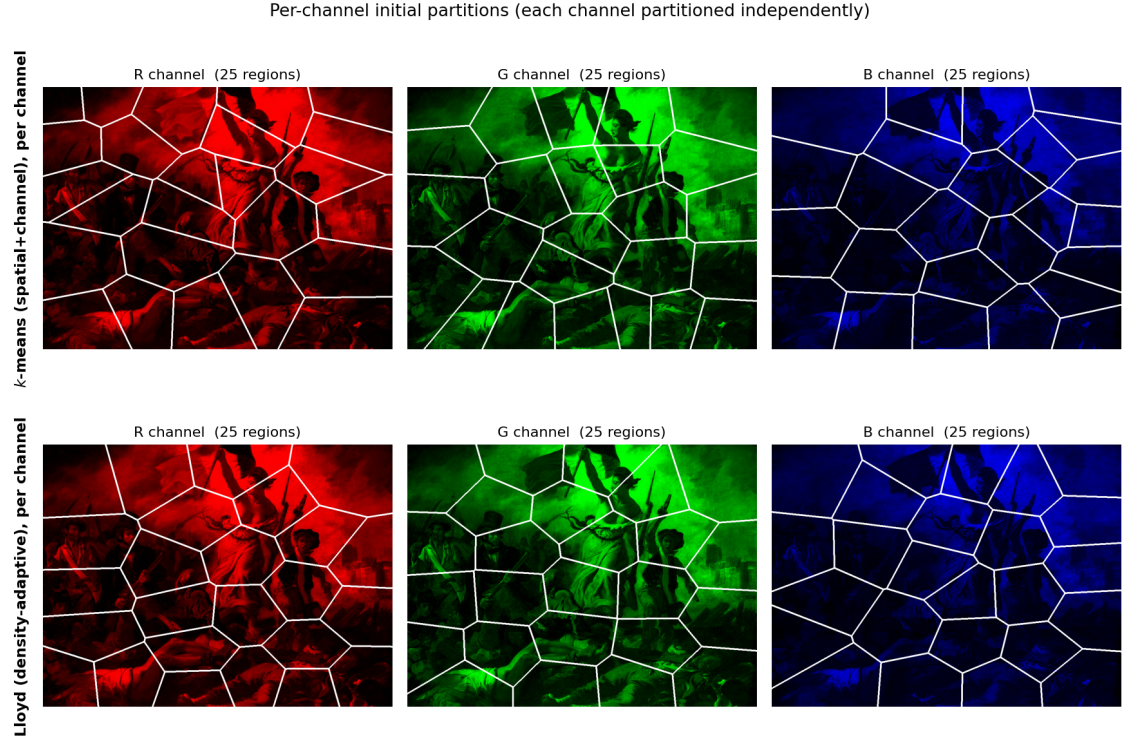


Figure 3.3: Per-channel variant: the joint k -means and Lloyd partitions computed *independently* on each colour channel, so each channel's partition adapts to its own content (the boundaries differ across R, G, B).

3.2 THE REFINEMENT LOOP: WHICH TREE, WHICH LEAF

TREE-SELECTION RULE

As already noted, a single image gives rise to *several* trees: one per colour channel (R, G, B , or the components produced by the colour transform) and, if a pre-partition is used, one per region of each channel. Since they share one budget, each step advances a single tree, chosen as the one maximizing a root score,

$$c^* = \arg \max_c u(\mathcal{R}^{(c)}), \quad u \in \{q, E, \tilde{E}\}. \quad (3.7)$$

The three choices for u are:

- *Binev value* $u = q$ (eq. (2.23)) — the choice most faithful to the framework. As we saw, $q(\mathcal{R}^{(c)})$ is the largest *modified* (penalized) error among the leaves of tree c , and thus, indirectly, an estimate of how much that tree could still be improved. Selecting by q amounts

to regarding all the trees as the children of one virtual super-root, turning the whole forest into a single large Binev tree: at each step the algorithm then descends straight to the globally most promising leaf — the one with the best margin of improvement according to the corrected error — exactly as the single-tree rule would. We make no formal claim for this multi-channel extension: we adopt it simply as the natural, well-motivated choice.

- *maximum error* $u = E$ — the purely greedy choice. Recall that $E(\mathcal{R}^{(c)})$ is precisely the approximation error of the *optimal subtree* currently extractable from tree c ; selecting the largest E therefore advances the tree whose present approximation is worst, with no discount for the effort already spent on it.
- *modified-error ceiling* $u = \tilde{E}$ — here $\tilde{E}(\mathcal{R}^{(c)})$ is the error of the whole tree *penalized* by how much the tree has already been expanded, mirroring at the level of trees the very logic Binev applies to leaves: give priority to the smaller, less developed trees. Since $\tilde{E}(\mathcal{R}) \geq q(\mathcal{R})$ always and the gap widens as a tree grows, this rule could in principle over-prioritize already heavily-refined trees; whether that bias matters in practice is a question we settle empirically in Chapter 4.

None of these changes the compression cost N as Binev defines it, but the choice does affect the *quality* of the approximation reached at a given budget, since it decides which tree is advanced and hence the whole trajectory of the run. Computationally the three rules are almost identical, since the root scores q , E and $\tilde{E}$ are all quantities the algorithm already maintains at every step.

LEAF-SELECTION RULE

Binev’s selection acts at two distinct levels: the *value a leaf exposes when it is born*, and the *rule by which values travel up the tree* to decide which leaf the root’s pointer $s(\mathcal{R})$ ends up selecting. We state the two separately: the alternative rules below are obtained by switching them off.

First level: the value at birth (leaf level). A newborn leaf ∇ does not enter the competition with its raw error $e_1(\nabla)$. It enters with the modified error of eq. (2.19),

$$q(\nabla) = \tilde{e}(\nabla),$$

the raw error attenuated by the harmonic factor accumulated along the ancestors of ∇ . The deeper the leaf, the stronger the attenuation: a cell reached only after many splits starts already damped.

Second level: the propagation (internal nodes). This level does not touch the value stored in any leaf; it governs the *pointer*, i.e. the selection criterion. As the priorities travel from the leaves towards the root, at each internal node Δ the winning child's value is capped by that node's modified subtree error:

$$q(\Delta) = \min \{ \max\{q', q''\}, \tilde{E}_{P(\Delta)}(\Delta) \},$$

and $s(\Delta)$ follows the winning child. Since $\tilde{E}_{P(\Delta)}(\Delta)$ shrinks as the branch grows and its order rises, a branch that has already received much refinement forwards a small priority — whatever its leaves report — and the pointer is steered elsewhere.

Together, the two levels mean that the selection does not simply chase the leaf of largest current error: it discounts leaves that are deep, and branches that are already well served. These are exactly the properties on which the near-best guarantee rests (with subadditivity and monotonicity). We nonetheless want to measure what this machinery buys on real images, so we switch the two levels off — one at a time, then both. All three rules split the leaf pointed to by $s(\mathcal{R})$; they differ only in the computation of q :

- *Binev* — $q(\nabla) = \tilde{e}(\nabla)$ at birth, and Binev's capped propagation at every internal node. The faithful choice, with the near-best guarantee.
- *Greedy-at-birth* (modifies) — a newborn leaf shows its raw error, $q(\nabla) = e_1(\nabla)$; the propagation is *identical* to Binev's, still capped by $\tilde{E}_{P(\Delta)}(\Delta)$ at every ancestor. The rule differs from Binev's only in the instant of birth — hence its name — and it carries *no* near-best guarantee.
- *Maximum error* (modifies) — raw error at birth *and* plain-max propagation, $q(\Delta) = \max\{q', q''\}$, with no cap. The value reaching the root is then simply the largest leaf error, so this rule always splits $\Delta^* = \arg \max_{\Delta \in \mathcal{L}(\mathcal{T})} E(\Delta)$ — the naive greedy, blind to branch history.

The difference between the last two is therefore exactly the propagation: greedy-at-birth keeps Binev's penalty on over-refined branches (dropping only the damping at birth), whereas maximum error discards it and blindly chases the largest error.

3.3 THE h -STEP: SPLITTING A CELL

Binev's framework requires only that a cell can be split into two, and gives no indication of *how*. To realize the split in a compact, geometry-aware form we adopt the binary wedge partitions

introduced by Erb [1] and presented in Chapter 2; a dyadic split, however, needs two ingredients that the framework does not provide — a *metric* to draw the boundary, and a rule to choose the new *centre*. The two parameters of this section fix exactly those ingredients.

SPLITTING METRIC α

The dyadic split assigns each pixel to its nearer centre in d_α (eq. (2.2)), so $\alpha \geq 1$ shapes the boundary. For $\alpha = 2$ the boundary between c' , c'' is their perpendicular bisector,

$$\Delta' = \left\{ v \in \Delta : \left\langle \mathbf{p}_v - \frac{\mathbf{p}_{c'} + \mathbf{p}_{c''}}{2}, \mathbf{p}_{c''} - \mathbf{p}_{c'} \right\rangle \leq 0 \right\}, \quad (3.8)$$

the cheapest to compute; $\alpha = 1$ and $\alpha \rightarrow \infty$ give restrictive, axis-aligned boundaries, and intermediate or larger α give finer, curved boundaries at higher cost (Figure 3.4).

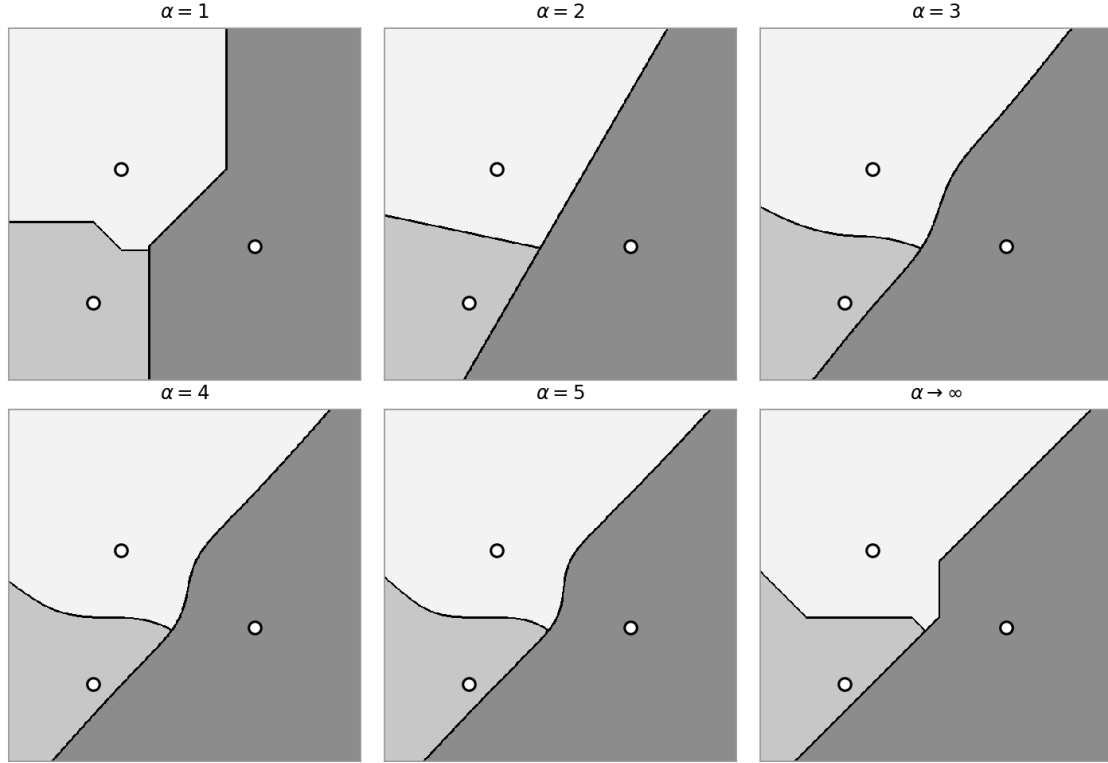


Figure 3.4: A partition built by two dyadic splits (three cells), shown under the metric d_α for several α . The same three centres are used in every panel; only the metric changes, bending the boundaries from the axis-aligned city-block shape ($\alpha = 1$), through the straight perpendicular bisectors ($\alpha = 2$), towards the Chebyshev shape ($\alpha \rightarrow \infty$).

CENTRE-SELECTION RULE

Restricting the abstract binary split to a *dyadic* split already collapses an enormous space of possible bipartitions — in principle, any way of cutting a cell into two — down to those realizable by a pair of centres under the metric. Within this restricted family the split is determined entirely by the choice of the new centre, which therefore becomes the crucial decision of the h -step. Recall (Section 2.2) that the first centre, at the root, is chosen arbitrarily, after which every leaf already carries a centre; splitting a leaf thus requires selecting only *one* new centre, and the old and the new one are inherited by the two children. How that single point is picked is exactly what the rules below decide. For a candidate $v \in \Delta$, write $\{\Delta'_v, \Delta''_v\} = \mathcal{S}(\Delta; c', v)$ for the resulting split and $\epsilon(v) := e_1(\Delta'_v) + e_1(\Delta''_v)$ for its induced combined error.

- *Maximum distance* — $c'' = \arg \max_{v \in \Delta} d_\alpha(v, c')$. Purely geometric and very fast, provided the distance itself is cheap to evaluate ($O(|\Delta|)$); it maximizes the homogeneity of cell *sizes* rather than of the signal.
- *Fully adaptive* — $c'' = \arg \min_{v \in \Delta} \epsilon(v)$, the centre that minimizes the resulting signal error. Evaluating $\epsilon(v)$ means performing the split for the candidate v and measuring the least-squares error on the two resulting cells — one fit per candidate. When the local model is the constant ($k = 1$) this error is proportional to the cell variance, which can be accumulated cheaply and speeds the evaluation up somewhat; even so, one fit for *every* pixel is $O(|\Delta|)$ fits per split, infeasible on the large cells that dominate the early steps of a run.
- *Randomized adaptive* — the same criterion restricted to a random sample $S \subseteq \Delta$:

$$c'' = \arg \min_{v \in S} \epsilon(v).$$

The number of sampled candidates $n = |S|$ is itself a parameter: it trades adaptivity for a fixed per-step cost of n fits, sidestepping the infeasibility of the fully-adaptive rule on large cells.

- *Hybrid (random adaptive + max distance)* — as randomized adaptive, but with the farthest pixel forced into the candidate set, $S = \{\arg \max_u d_\alpha(u, c')\} \cup S_{\text{rand}}$. The maximum-distance point often turns out to be a genuinely good centre, so it is worth always keeping it among the candidates.

Even the adaptive rules are optimal only at the step in which they act — the algorithm is greedy, so a locally optimal split need not remain optimal after later steps.

3.4 THE p -STEP: GROWING THE POLYNOMIALS

On the climb, an ancestor's order rises and its model is re-fitted (Algorithm 2.1, step (vii)). Because the budget is additive on the main tree,

$$P(\Delta) = P(\Delta') + P(\Delta''), \quad (3.9)$$

the two parameters below set, respectively, the rate at which the order grows (the initial order k) and the point at which it stops (the maximum order $P_{\max}$).

INITIAL LOCAL ORDER k

Binev initializes every new leaf at order 1; we allow $P(\nabla) = k$ for a fixed $k \geq 1$ (a constant for $k = 1$, a partial linear model for $k = 2$, the full linear model $a + bx + cy$ for $k = 3$). By the additivity (3.9), a parent of two order- k leaves has order $2k$ at once, so only the budgets $N \in \{k, 2k, 3k, \dots\}$ are reachable and the parameter count climbs in multiples of k with the number of steps. This coarser accounting has a clear practical appeal, best seen in the trade-off between steps and error. At a *given parameter count* a larger k typically yields a worse approximation than $k = 1$, since the near-best sequence is sampled more coarsely; but because each step now buys k parameters at once, a good approximation can be reached in *far fewer steps*, and hence much faster at compression time. This is the substantial advantage of raising k : a gain in speed, to be weighed against the loss in quality. Formally it is the generalization anticipated at the end of Chapter 2: the near-best guarantee still holds on this coarser sequence. Empirically large k rarely helps — most final cells settle on a constant — but it is available; the constraint is $1 \leq k \leq P_{\max}$.

MAXIMUM LOCAL ORDER $P_{\max}$

We cap the climb. Each visited ancestor raises its order by the step k , up to the cap,

$$P(\Delta) \leftarrow \min\{P(\Delta) + k, P_{\max}\}, \quad (3.10)$$

and once $P(\Delta) = P_{\max}$ the model is frozen while the errors keep updating. Several remarks are in order. Empirically it is very rare for the optimal model on a cell to be a high-order polynomial, so a moderate cap costs almost nothing in quality. The cap is, however, not innocent in theory: bounding the order *invalidates* the near-best proof of Theorem 2.1, which assumes

the order may grow without bound — although in practice the difference is minimal. The motivation is computational: re-estimating a polynomial at every ancestor along the whole climb, an $O(\log |\mathcal{T}|)$ -long path for a balanced tree, is expensive, and the cap removes the most costly high-order fits. The parameter satisfies $P_{\max} > 1$ (and $P_{\max} \geq k$), so that some polynomial growth is possible at all.

MONOMIAL-ORDERING RULE

Binev’s framework asks only that the local error never increase as the order grows; since we use nested spaces precisely to guarantee that decrease (Lemma 2.2), we must choose the order in which the basis is built — *which* monomial realizes each increment is left open. We enlarge the basis by increasing total degree $d = a + b$, completing every monomial of degree d before any of degree $d + 1$; the rule chooses, among the monomials of the current degree, the next ϕ_{a^*, b^*} . Writing $\mathcal{B}$ for the current active basis on Δ :

- *Deterministic* — a fixed sequence, independent of the data, generated by $(a, b) \mapsto (a - 1, b + 1)$ if $a \neq 0$, else $(a + b + 1, 0)$; that is,

$$(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2), (3, 0), \dots$$

It is the simplest, fastest, and most intuitive rule, and it stores no metadata: since the order is fixed, the compressed file need not record which monomials were used. Its drawback is that it has the least room to adapt.

- *Geometry-adaptive* — $(a^*, b^*) = \arg \max_{a+b=d} \text{Var}_{v \in \Delta} (x_v^a y_v^b)$. As the algorithm proceeds — and especially under non-Euclidean metrics — cells take on varied, elongated shapes; this rule is greedy on the cell *geometry* rather than on the signal, preferring the monomials that vary most over the cell and therefore carry the most information. A further advantage is that it depends only on the cell: when work on a degree d begins, all its monomials can be sorted by variability once, cached, and then consumed one at a time.
- *Signal-adaptive* — $(a^*, b^*) = \arg \min_{a+b=d} e(\Delta; \mathcal{B} \cup \{\phi_{a,b}\})$, where $e(\Delta; \mathcal{B})$ is the least-squares residual on Δ with basis $\mathcal{B}$. It is the best rule *for a single step*, and also by far the most expensive: for every candidate monomial of the current degree it must re-fit the whole polynomial — an increasingly heavy least-squares problem as the polynomial already grows large. The burden is greatest early in the exploration, when a cell still offers many candidate monomials to try. Worse, this is not done once but along the *entire climb*: at every ancestor visited on the way back to the root the basis is re-selected, and, being data-dependent, the order in which monomials are added is not fixed, so it cannot be cached (unlike the two rules above). And, as always, its per-step optimality need not survive later steps, given the greedy nature of the algorithm.

The two adaptive rules share a compression drawback: the active set is no longer a fixed prefix, so the file must record *which* monomials were used, not merely the coefficients — another piece of information to weigh when deciding which criterion to prefer.

COORDINATE NORMALIZATION

Before fitting, coordinates are rescaled to $[0, 1]$ over a reference set Ω ,

$$\tilde{x}_v = \frac{x_v - \min_{u \in \Omega} x_u}{\max_{u \in \Omega} x_u - \min_{u \in \Omega} x_u}, \quad \tilde{y}_v = \frac{y_v - \min_{u \in \Omega} y_u}{\max_{u \in \Omega} y_u - \min_{u \in \Omega} y_u}, \quad (3.11)$$

with three choices of reference: $\Omega = \Delta$, the cell itself (a per-cell bounding box); $\Omega = V$, the whole image; or no rescaling at all, $\tilde{x}_v = x_v$ and $\tilde{y}_v = y_v$. This changes only the polynomial coefficients — not the split sequence, the errors, or any decision — and is therefore purely representational; its role is the numerical stability of the high-order fits.

3.5 STOPPING CRITERIA

The loop halts when any of three conditions is met.

- *Step budget* — a maximum number of refinement steps, $N_{\max}$: the simplest criterion.
- *Relative-error threshold* — stop once the relative error, the total squared error over the signal variance, falls below ε :

$$\frac{\sum_c \mathcal{E}(\mathcal{T}_N^{(c)}, T_N^{(c)})}{\sum_c \|f^{(c)} - \bar{f}^{(c)}\|^2} \leq \varepsilon.$$

- *Parameter budget* — stop once the total optimal parameter count reaches a bound:

$$\sum_c \#\mathcal{P}(T_N^{(c)}) \geq P_{\text{budget}}.$$

A difference from Binev’s framework concerns these budgets. Binev makes the number of steps coincide with the number of parameters — for him a single step adds exactly one parameter. In our adaptation the two no longer coincide, for two reasons. First, with the order step k and the cap $P_{\max}$, a single refinement step does not necessarily add exactly one parameter. Second, and more fundamentally, the parameter count is not itself a direct, exact measure of

the compressed size — a point we will return to later. Binev counts only the polynomial coefficients, whereas the stored file must also record the partition information (the centres), and coefficients and centres plausibly do not occupy the same number of bits; on top of that sit the configuration, the pre-partition, the metric, the monomial ordering, and other bookkeeping. Some of these are already folded into our count (a small header and the pre-partition overhead), others are not, so the count is best read as a proxy for the compression level rather than its exact value. Even so, the parameter budget is the criterion closest to the actual compressed size.

Table 3.1 summarizes all the parameters introduced in this chapter.

Table 3.1: The parameters of the algorithm.

Parameter	Phase	Category	Type	Level
Colour-space transform	setup	Representation	categorical	none / PCA / YCbCr
Initial partition	setup	Extends	categorical	none / blocks / k -means (spatial, joint) / Lloyd (shared, per-channel)
number of regions	setup	Extends	integer	≥ 1
feature weights	setup	Extends	real	≥ 0
Lloyd iterations	setup	Extends	integer	≥ 1
Tree-selection rule	loop	Extends	categorical	Binev q / error / $\tilde{E}$
Leaf-selection rule	loop	Compl./Modif.	categorical	Binev / greedy-at-birth / error
Splitting metric α	h -step	Completes	real	≥ 1
Centre-selection rule	h -step	Completes	categorical	max distance / fully adap- tive / randomized / hy- brid
sampled candidates n	h -step	Completes	integer	≥ 1
Initial local order k	p -step	Extends	integer	$1 \leq k \leq P_{\max}$
Maximum local order $P_{\max}$	p -step	Modifies	integer	> 1
Monomial-ordering rule	p -step	Completes	categorical	deterministic / geometry / signal
Coordinate normalization	p -step	Representation	categorical	local / global / none
Step budget $N_{\max}$	termination	Operational	integer	≥ 1
Relative-error threshold ε	termination	Operational	real	$[0, 1]$
Parameter budget	termination	Operational	integer	≥ 1 , or none ($= \infty$)

3.6 EXTRACTION AND REASSEMBLY

When the loop stops, the main tree $\mathcal{T}_N$ of each channel is trimmed to its optimal subtree $T_N \subset \mathcal{T}_N$ by the rule (2.25): descending from the root, the tree is cut at the *first* node Δ encountered with $E_{P(\Delta)}(\Delta) = e_{P(\Delta)}(\Delta)$ — that is, whose own order- $P(\Delta)$ model is at least as good as the best combination of models on its descendants — and that node is made a leaf of T_N . One clarification is worth making explicit: it is *not* the case that every node satisfying the equality becomes a leaf. The condition may well hold again at nodes further down (one below the other), but once a node is cut, its entire subtree is discarded and never examined; only the *shallowest* node of each branch on which the equality holds survives, as a leaf. The *leaves* of this returned subtree T_N define the final partition: each such leaf is one cell of the reconstructed image, carries a single polynomial, and together they tile the whole domain (or the pre-partition block). Everything below those leaves — the finer splits still present in the main tree — is discarded; it served only to *measure*, during the growth, how much each region could still be improved.

The output of a tree is thus a compact pair: the ordered sequence of centres that replays its partition down to the leaves of T_N , and the coefficient vector of the polynomial on each such leaf.

It is worth spelling out how the partition is rebuilt from that flat list of centres — an elementary step we have so far taken for granted, and the one in which the compactness of the BWP encoding (Section 2.2) is cashed in. Reconstruction simply reverses the growth. The decoder starts from a single cell, the whole domain V (or the pre-partition block), whose centre is the first entry of the sequence — the root centre, fixed arbitrarily at encoding. It then reads the remaining centres one at a time and, for each centre c , performs exactly one split: it locates the unique current cell Δ that contains c — since the current cells tile V and $c \in V$, exactly one does — and replaces it by the dyadic split $\mathcal{S}(\Delta; c_\Delta, c)$ (Definition 2.4), where c_Δ is the centre already attached to Δ . The part nearer c_Δ keeps that centre; the part nearer c becomes a new cell carrying c . When the sequence is exhausted, the current cells are exactly the leaves of T_N , and the partition is recovered.

The *order* of the centres is essential, and this is the point worth stressing. The topology of the tree — which cell descends from which — is never stored: there are no explicit edges, only the bare list of points. That structure is carried entirely by the order, through the rule above: the cell a centre splits is whichever cell happens to contain it at that moment, and that in turn depends on every split performed before it. Read in the encoding order, each centre falls

back into exactly the cell in which it was born, and is bisected against exactly the sitting centre it was paired with, so the tree is rebuilt identically. Permuted, the same set of centres generally yields a *different* partition: a centre taken too early lands in a larger, not-yet-subdivided cell and splits the wrong region, and the centre it is measured against changes with it. This is precisely the trade behind the one-point-per-split compactness of Section 2.2, with the ordering silently doing the work that a stored list of tree edges would otherwise cost. The decoder must of course also share the metric d_α used at encoding, since the geometry of every split depends on it.

It is worth adding that the order we store is not necessarily the *chronological* one, the order in which the algorithm actually produced those centres. There is nothing unique about that chronological order: *any* sequence in which each centre comes after the centre of the cell it lands in rebuilds the same partition, and the growth order is only one such sequence — the two may even coincide, but nothing requires it. So we do not record the history. Instead, once the run is over, we read the centres straight off the final tree by a depth-first traversal and store *that* order — itself a valid one, since the traversal visits a cell’s centre before any of the descendants that subdivide it. What must be preserved is a *consistent* order, not the true history, and the tree already contains one for free.

Finally, the stored coefficients are attached to the recovered leaves in the same order used at encoding, each leaf’s signal is evaluated from its polynomial, the channels are reassembled and, if a colour transform was applied, inverted via (3.2). The stored representation — the ordered centres, the per-cell coefficients, the small colour-transform overhead, and the active-monomial sets if an adaptive ordering was used — is the compressed image.

4

Implementation and Results

In Chapter 2 we set out the theoretical foundations of our method — the framework of Binev and the near-best approximation results that come with it. In Chapter 3 we turned that theoretical model into a precise, detailed algorithm, together with the whole family of design choices that any real implementation must fix and that the abstract theory leaves open. Both chapters, though, stayed on the mathematical plane. In this chapter we move to the practical side: how the algorithm is actually implemented, and — above all — how it behaves. For the theory does not tell us whether these choices matter in practice, how much, and in which direction: whether Binev’s selection really beats the naive greedy on real images, how the polynomial order trades accuracy for cost, how much a colour transform or a pre-partition actually buys.

This chapter answers those questions empirically. It is organized in three parts. Section 4.1 describes how the objects of Chapter 3 are represented as running code, since that representation is what fixes the costs we then measure. Section 4.2 is the core: it introduces the way we measure error, compression and time, and then studies, one design choice at a time, how each moves those three quantities — reading off the trade-off curves and, where it is illuminating, the precise cost of the operations involved. Section 4.3 then looks at the compressed representation itself — what it is made of, how many cells and polynomials it contains — and at its true size in bytes. Section 4.4 closes the chapter by combining the best levers into a single *optimal configuration*, and showing how the initial partition buys back much of its computational cost. A broader discussion of what the method does well and where it falls short is deferred to the concluding chapter.

Throughout, the test image is Delacroix’s *La Liberté guidant le peuple*, the public-domain image already used in Chapter 3, at a resolution of 1024×768 (786 432 pixels per channel, three channels). It was chosen for its mixed character: large smooth regions (the sky, the ground) next to finely detailed ones (the figures, the flag), so that both kinds of refinement — spatial and polynomial — have work to do.

4.1 IMPLEMENTATION AND REPRESENTATION

The method is realized as a Python library, written for this thesis and publicly available at <https://github.com/GottiPaolo/HP-Image-compression>, where the full source and the scripts that produced the experiments below can be consulted. We describe here only what is needed to read the results — how the algorithm is organized, and how its objects are represented — leaving finer points to the sections where they become relevant.

THE TWO CLASSES THE USER TOUCHES. The library is built around a clear separation between the design choices and the data they act on. A *configuration* object gathers, in one place, all the design choices of Chapter 3 — the colour space, the initial partition, the splitting metric and the selection rules, the way the local polynomials grow, and the stopping criteria — each as a named field with a sensible default. An *image* object, instead, is built from two things: an actual image — read from a file, from an array already in memory, or by other means — and a configuration produced by the class just described. It splits the picture into one signal per colour channel and drives the whole compression, exposing a deliberately small interface: the operations to run the compression, to reconstruct the image, and to save it; a usage example can be found in the repository.

THE CLASSES UNDERNEATH. These two are only the visible surface of a small hierarchy of classes, each with a single responsibility. It is worth listing them, because their division of labour mirrors the structure of the algorithm itself.

- The configuration (`HPConfig`) is a plain data record: one field per hyperparameter of Chapter 3, with defaults and a little validation. It carries no logic — it is simply handed down to everyone else, so that a single object fixes the behaviour of an entire run.
- The image (`HPImage`) is the orchestrator and the only public entry point. It owns the whole *setup* phase — reading the picture, applying the colour transform, and building the initial partition — and from it splits the image into its colour channels, holding *one*

tree per channel (or one per channel and region, when a pre-partition is used) — the *forest* of Chapter 3. Its central task is then the refinement loop: at each step it decides which channel's tree is most worth advancing and asks it to grow, until a stopping criterion is met; it then reconstructs the image or serializes the result (its main methods being `run`, `reconstruct` and `save_hp`). It knows nothing of the internal *hp*-mechanics — those live in the trees.

- The tree (`HPTree`) is the heart of the algorithm: it holds, for a single channel, the *entire* Binev machinery of one tree — the local polynomial models, the incremental *hp*-step, and the near-best bookkeeping (the recursive errors E and $\tilde{E}$, the selection functions q and s) that decides where to act next. Its central operation is exactly the incremental step of Chapter 3: split one leaf, and update, along the path back to the root, the errors and the Binev quantities; a companion routine extracts, at the end, the optimal subtree that becomes the output — together with the optimal subtrees of every other tree in the forest. Every least-squares fit it hands to the approximator below. This correspondence is worth stressing: everything Chapter 2's theory says about a single tree lives, in full, inside this one class. The algorithm as a whole never runs on a tree in isolation — it orchestrates a whole forest of them — but that single-tree picture, embodied here, is the clearest way to see the machinery at work.
- The node (`TreeNode`) is the lightweight record of a single cell, whether a current leaf or an already-split internal node. Each node stores the id of its parent and of its two children, the single representative point that is its *centre*, its local polynomial (its order and its coefficients), its local error, and the running Binev quantities — the recursive optimal error and the pointer to the next leaf to refine. A tree is then, quite literally, a list of such nodes.
- The polynomial approximator (`PolynomialApproximator`) is a small, stateless helper: given the pixels of a cell and a degree, it returns the best-fitting polynomial and its residual error, and nothing else. Keeping it separate means the fitting mathematics lives in one place, independent of the tree that calls it.

This same division fixes how the objects that carry a tree's state are actually held in memory — which matters, because it is exactly what fixes the costs we later measure.

- The partition generated by the binary tree is a labelling, not an explicit subdivision at code level. Rather than storing the cells as regions, every pixel simply carries the label of the cell it currently belongs to. A cell is then read off in a single pass over the grid, and a split only needs to relabel the pixels that actually change side — the compact, one-new-centre-per-split structure of Chapter 2 made operational.

- The tree is a list of nodes. The nodes are indexed by a progressive identifier, so a parent reaches its children, and a child its parent, by id rather than by chasing pointers; this keeps the tree cheap to traverse, to update along a single root-to-leaf path, and to serialize.
- A final cell is a centre plus a few coefficients. The unit that survives into the output is the leaf: one centre and a short vector of polynomial coefficients. The internal nodes are never reconstructed from; they existed only to *measure*, during the growth, how much each region could still be improved, and are discarded at extraction (Chapter 3).

The interactions among the classes are as simple as the list suggests: the configuration flows down into everything; the image owns the trees and drives them; each tree owns its nodes and leans on the approximator for the arithmetic. Nothing ever reaches back up — a tree knows nothing of its siblings or of the image above it — which is exactly what lets the multi-channel refinement be no more than an orchestration of independent single-channel algorithms.

The compressed artefact that all this produces is therefore compact and structured: for each tree, the *ordered* sequences of centres that replay its partition down to the leaves (Chapter 3) and the coefficient vector of each leaf, together with a tiny header and — when a colour transform is used — the few numbers that define it. Stored as a compact binary file, this is the object whose size Section 4.3 will translate into bytes and bits-per-pixel; for most of the chapter we measure it, more simply, by its total number of stored parameters N — in the main the cell centres and their coefficients, but not those alone: it also folds in the small overhead the file carries with them, a short header, the descriptors of the initial pre-partition when one is used, and the few numbers of the colour transform. N is, in short, the whole stored representation. Where it matters we will look past this single figure and distinguish its parts — above all the centres from the coefficients, which behave quite differently.

A word on the implementation's ambitions. A good deal of care, mathematical and computational, went into making the algorithm efficient enough to be genuinely usable: incremental updates in place of recomputation, cached pixel sets, vectorized numerical kernels, and the near-best bookkeeping that spares the algorithm from ever rescanning the whole tree. The result is a genuinely practical tool. A truly state-of-the-art codec, though — shaving off the last constant factors, parallelizing aggressively, dropping down into low-level C — calls for a craft of high-performance engineering that is not the specialty of this work, and lies beyond the scope and the aims of this thesis. The analyses that follow are, accordingly, first and foremost of *mathematical* interest — the quality of the approximation and the behaviour of the algorithm — with the computational times a real but secondary, qualitative concern.

Finally, a note on determinism, because it is not quite total. The algorithm has two sources of randomness, both controlled: the random-adaptive centre-selection rule, which draws a handful of candidate centres at each split, and the data-adaptive pre-partitions, whose k -means and Lloyd clusterings start from randomly chosen centroids. Everywhere in this thesis we fix the random seed to 42, so that both are reproducible; with the seed fixed, a run always yields the same partition, the same errors and the same reconstruction, and the whole dataset behind this chapter can be regenerated from scratch. Everything else in the method — the splits, the fits, the selection, the extraction — is genuinely deterministic.

4.2 FIRST-LEVEL ANALYSIS

This is the core of the chapter. We first fix how the three axes are measured, then study, one design choice of Chapter 3 at a time, how each moves them. The guiding object throughout is the *trade-off curve*: for every run we have recorded the whole history of the approximation, so we can plot the error not only at the end but at every intermediate budget, and read a design choice as the curve it produces rather than as a single final number.

ERROR. The algorithm minimizes, and we report, the total squared error $\mathcal{E} = \sum_v \|f_v - \hat{f}_v\|^2$ between the original image and its reconstruction, summed over pixels and colour channels. In the plots we use the dimensionless *relative error* $\mathcal{E}/\sigma^2$, where $\sigma^2 = \sum_v \|f_v - \bar{f}\|^2$ is the total variance of the signal — its energy about the mean colour $\bar{f}$. The relative error is thus the fraction of the image’s energy still unresolved, so 1 means “no better than a flat image” and small values mean a faithful reconstruction. For the summary tables we also report the *peak signal-to-noise ratio* (PSNR), the standard figure of merit in image compression [11, 12]. It is defined from the mean squared error $\text{MSE} = \mathcal{E}/(WHc)$ over the WHc colour samples as $\text{PSNR} = 10 \log_{10}(\text{MAX}^2/\text{MSE})$, where MAX is the peak intensity value; since we work with intensities normalized to $[0, 1]$, here $\text{MAX} = 1$ and the formula reduces to $10 \log_{10}(1/\text{MSE})$. Higher is better, and a few tenths of a decibel are already visible on natural images. The two, though on different scales, carry the same information — PSNR is a monotone transformation of the error — but PSNR makes the results comparable with the compression literature.

COMPRESSION. We measure the size of the compressed representation, for most of the chapter, by its total number of stored parameters N — chiefly the centres of the final cells and their

polynomial coefficients, but not those alone: a small header and, where used, the pre-partition and the colour transform are stored with them (Chapter 3). It is a proxy, not a byte count, and a deliberately even-handed one, since it counts every parameter as one — whereas in the eventual file the different kinds weigh differently, a centre costing a different number of bits from a coefficient. A thorough study of the compression — what each kind of parameter actually costs in bits, and the resulting file size — is carried out in Section 4.3. But it is the quantity the algorithm actually controls, and it is what makes runs comparable: a run is *better* when it reaches a given error with fewer parameters, i.e. when its trade-off curve lies lower and to the left. We should add that squeezing the representation into the smallest possible file was never the aim here — efficient byte-level coding is a craft of its own, outside our focus — so for the analysis we lean on the clean, algorithmic measure N , and postpone the real bytes to Section 4.3, where we look at them properly.

TIME. We also recorded the computational time of every run. These numbers must be read with care, and only *qualitatively*. As already said, the implementation is a research prototype in Python, not an optimized codec, and each configuration was run once, with the numerical linear-algebra libraries restricted to a single thread, so that every run is measured on a single core and the times are comparable across configurations. The data collection, by design, adds no overhead of its own: the timing counts only the algorithm’s own work, with the bookkeeping (error evaluation, parameter counting) excluded. Absolute seconds are therefore not a benchmark; what *is* meaningful is the comparison between configurations measured the same way — “rule A costs about twice rule B” — and, occasionally, the shape of where the time goes, which we look at directly in a couple of the parameter discussions below. One caveat keeps even those comparisons honest: one technique we deliberately did not use is parallelization, and it would not scale all times down uniformly — the many local polynomial fits are independent and could be done in parallel, whereas the hierarchical splitting is inherently sequential, so a parallel implementation would speed up some rules more than others and reshape some comparison. The single-thread times we report are, in that sense, a conservative and deliberately even-handed baseline.

4.2.1 THE REFERENCE CONFIGURATION, AND A FIRST LOOK

THE REFERENCE CONFIGURATION. Because Chapter 3’s design choices interact, there is no single “neutral” configuration; we instead fix a *reference configuration* and, for each parameter

in turn, vary that one parameter while holding all the others at the reference. It is a deliberately realistic-but-clean choice, listed in full in Table 4.1: in words, the principal-component colour transform, no pre-partition, the metric with $\alpha = 2$, the random-adaptive centre-selection rule, the function q both for choosing the tree and the leaf, the deterministic polynomial progression, and constant initial polynomials capped at quadratic. On the test image, run to one hundred thousand refinement steps, this reference reaches a relative error of about 0.0085 — a PSNR of 33.1 dB — with roughly 178 000 parameters. *Unless a figure says otherwise, every plot in this section is produced from exactly this configuration, at that same number of steps, with the one named parameter varied.*

Table 4.1: The reference configuration, from which each first-level study departs one parameter at a time.

Parameter	Value
Colour-space transform	PCA
Initial partition	none
Splitting metric	$\alpha = 2$
Centre-selection rule	random-adaptive, 10 candidates
Tree-selection rule	Binev (q)
Leaf-selection rule	Binev
Polynomial progression	deterministic
Initial order k	1 (constant)
Maximum order $P_{\max}$	6 (quadratic)
Coordinate normalization	local
Random seed	42
Refinement steps	100 000

Two habits make the curves easy to read. First, *lower and to the left is better*: the same quality for fewer parameters. Second, the final numbers in the tables are read *together with N* : at a fixed number of refinement steps two configurations may stop at different budgets, so a smaller final error at a larger N is not automatically a win — the curve, which controls for N , is the honest comparison, and the table is its summary.

A word on the stopping criterion of these simulations. Unless stated otherwise, every run in this study is stopped at the same number of refinement *steps*. The step count essentially tracks the parameter count, with two qualifications. The number of *centres* depends on how

strongly the final cells merge: each final cell costs one centre, so configurations that end with fewer, larger cells store fewer centres at the same step count. The number of *coefficients*, instead, matches the steps almost exactly — one new coefficient per step — except for the steps in which the algorithm would raise the order of a region that has already reached the cap $P_{\max}$: there the step adds no parameter at all. Finally, in a few specific studies the step count is itself a misleading yardstick; there we will say so explicitly and compare on a different measure.

A FIRST LOOK: PROGRESSIVE RECONSTRUCTION. Before the parameter study, Figure 4.1 shows what the method does: the test image reconstructed at five increasing budgets, with the original in the last panel for comparison. At a few hundred parameters the picture is a coarse mosaic of wedge-shaped cells, each a single colour; as the budget grows the partition refines exactly where the image is busy — the flag, the faces, the edges of the figures — while the smooth sky and background stay large and cheap, and the polynomials begin to bend across the larger cells. By about 178 000 parameters the reconstruction is faithful. Table 4.2 gives the parameter count and relative error at each budget. This is *hp*-adaptivity made visible: spatial refinement and polynomial enrichment concentrated where they pay.



Figure 4.1: The test image reconstructed at five increasing budgets (reference configuration); the original is the last panel. The wedge partition refines where the image is complex.

Table 4.2: Progressive reconstruction: parameters and relative error at each step count.

Steps	Parameters N	Relative error $\mathcal{E}/\sigma^2$	PSNR (dB)
300	541	2.2×10^{-1}	18.9
1 500	2 619	1.1×10^{-1}	22.0
6 000	10 381	5.3×10^{-2}	25.1
25 000	43 723	2.3×10^{-2}	28.8
100 000	178 673	8.6×10^{-3}	33.0

4.2.2 THE COLOUR SPACE

The first choice a run makes is the space in which the three channels are approximated. As everywhere in this first-level study, the runs compared here are stopped at the same number of refinement steps (100 000), so they reach slightly different final budgets N and errors (Table 4.3); the honest comparison, as noted above, controls for N through the curve. Figure 4.2 compares the three options. Approximating the raw R, G, B channels is markedly the worst: the channels are strongly correlated, so budget is spent three times on the same structure. Decorrelating first — rotating onto the principal components, or splitting luminance from chrominance — recovers about two and a half decibels at the same budget, with the principal-component transform slightly ahead of luminance–chrominance (Table 4.3). The gain is essentially free: the transform is a single fixed rotation applied once, stored in a handful of numbers. This is the clearest single lever in the whole study, and it is why the reference configuration keeps the principal-component transform switched on.

One remark on how the error is measured under a colour transform. The algorithm works, and reports its error, in the transformed space — but for the principal-component transform this is mathematically the *exact same number* as the error in R, G, B , because that transform is orthonormal and so preserves the squared error. The luminance–chrominance transform is not orthonormal, so it does not; there, to keep the reported errors genuine, at every step the current approximation is converted back to R, G, B and the error is measured in that space. This is comparatively expensive, but it is purely a matter of data collection and does *not* affect the computational times we report — that extra work is excluded from the timing. Either way, then, the errors are true reconstruction errors, whatever colour space the run worked in.

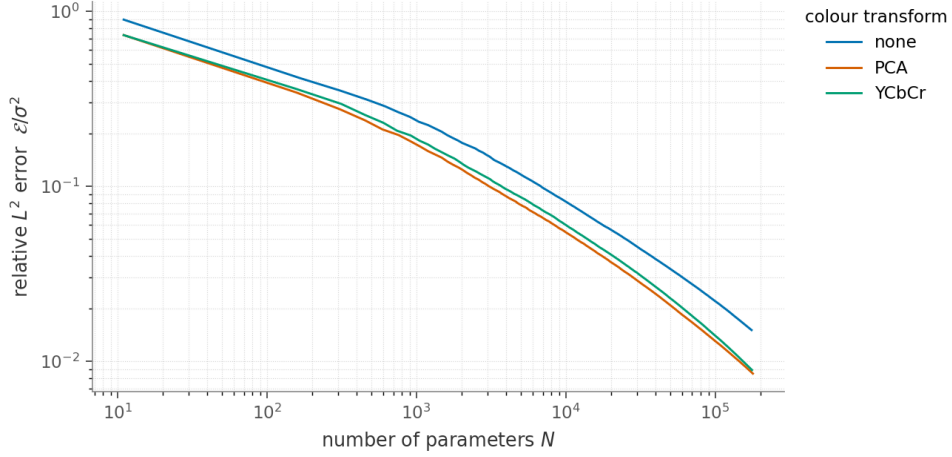


Figure 4.2: Effect of the colour-space transform: relative error against the number of stored parameters N (log-log). Both decorrelating transforms outperform the raw R, G, B representation.

Table 4.3: Colour-space transform: end-of-run figures on the test image.

Colour transform	PSNR (dB)	Parameters N
none	30.58	175 107
PCA	33.07	178 676
YCbCr	32.87	176 232

4.2.3 THE INITIAL PARTITION

The reference grows one tree per channel from the whole image; alternatively we may *pre-partition* the image into regions and grow one tree per channel per region. Before the data, the theory already tells us which should win on quality. With no pre-partition the algorithm is free to place its budget anywhere, and finds the global near-best; a pre-partition freezes a set of region boundaries and never lets the algorithm move budget across them, so at any budget it can only *match* the no-partition result or fall short of it. No pre-partition should therefore be the best choice for quality — with one caveat: the algorithm is greedy, not optimal, so a sensibly placed pre-partition might occasionally help it avoid a poor greedy path.

We compare, at a consistent count of about fifty initial regions, no pre-partition against a regular 7×7 grid of blocks and against k -means and density-weighted Lloyd, each in two forms: a *shared* one, a single partition used for all three colour channels, and a *per-channel*

one, which fits a separate partition to each channel’s own content, adapting more closely but storing three partitions rather than one. All six error-versus-budget curves are *indistinguishable*: the adaptive groupings are so sensible, and the grid so mild, that the imposed boundaries cost essentially nothing, and the sub-percent reversals are greedy noise. The shared and per-channel forms coincide in quality in particular; the per-channel form pays only a little more to store — three sets of region centres instead of one, which shows up as a slightly larger starting budget ($N = 453$ against 353, Table 4.4), a negligible fraction of the final total. The region count must be read with its meaning: for a grid it counts divisions per side (k^2 blocks), for k -means and Lloyd it counts clusters, so we matched them all at ≈ 50 regions to compare like with like.

If quality is at best unchanged, why pre-partition at all? For a modest gain in *speed*. By far the most expensive operation in a step is splitting a cell and re-fitting its polynomial, and that cost grows with the number of pixels in the cell; the early splits, acting on enormous cells, are therefore far more expensive than the late ones over tiny cells — the cost is *front-loaded* onto the first splits. A pre-partition sidesteps exactly this: each tree starts from a small region, so the ruinous early splits over the whole image never happen, and every pre-partitioned run descends more steeply at the start (Figure 4.3(b)). Measured consistently — and counting the cost of building the partition itself — this makes the block-grid and k -means runs about a fifth faster than no pre-partition (about 40 against 48 seconds on the test image). The one qualification is that the partition is not always free to build: the density-weighted Lloyd clustering is itself iterative, and its per-channel form fits three separate clusterings, so that setup cost gives back part of the refinement saving — enough that per-channel Lloyd ends up no faster than no pre-partition at all at this budget. And because the regions are independent, they could be processed fully in parallel on top of this, which no pre-partition cannot. The pre-partition is thus a lever for time — a real if modest one — not for quality, as Chapter 3 anticipated. Modest, that is, on the cheap reference configuration: as the closing section of this chapter will show (Section 4.4), its value grows with the cost of the configuration it is applied to.

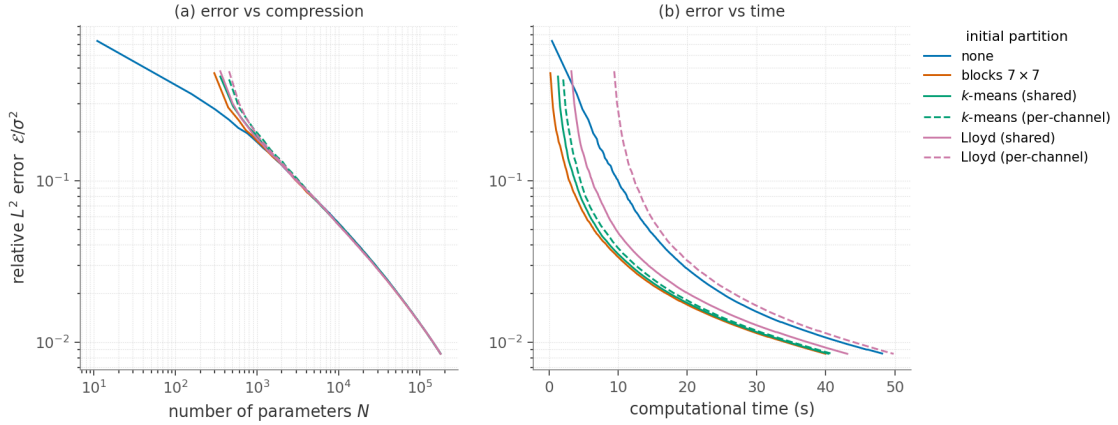


Figure 4.3: Initial partition, at ≈ 50 initial regions each; k -means and Lloyd are shown in both shared (solid) and per-channel (dashed) form. (a) error against budget: all six are indistinguishable in quality, and the per-channel forms merely start at a slightly larger N . (b) error against time (counting the cost of building the partition): the block grid and k -means reach a given error fastest, while the costlier-to-build Lloyd clustering gives back part of the gain — its per-channel form, fitting three clusterings, ending level with no pre-partition.

The two panels repay a direct reading. In panel (a) every pre-partitioned curve starts far to the right of the single-tree run — at many more parameters — and this is only in small part the cost of the region descriptors themselves: mostly it is that each initial region already carries its own starting polynomial, so the run begins with dozens of coefficients already spent. From there each curve drops *steeply* against N , because in cells that small a polynomial fits the local signal almost at once. In panel (b) the same head start reads as speed: the pre-partitioned curves are visibly steeper and reach a given error sooner. The exception is Lloyd, which is itself expensive to build, so its curve starts *later* on the time axis; and the per-channel Lloyd, building three separate clusterings before a single refinement, starts so late that it never catches up — ending, on this image, slightly worse than no pre-partition at all.

This is one of the studies announced above in which the raw step count is a misleading yardstick. A pre-partitioned run does not start from zero: its initial regions already carry their own starting polynomials, so at any step count it has simply stored more. The honest comparison is at an equal number of *coefficients* — exactly Binev’s own budget, the quantity his theory bounds, though one should remember that it is a mathematical measure: in the eventual file the centres, which it ignores, also weigh their bits. Table 4.4 therefore reads every configuration at the matched coefficient budget of the free run ($\approx 99\,600$ coefficients).

Table 4.4: Initial partition at ≈ 50 initial regions (per channel for the per-channel variants; test image). Every row is read at the *same* coefficient budget — the 99 634 coefficients of the free run, Binev’s own measure: each pre-partitioned run is stopped at the first step where its stored coefficients reach that budget, which happens a hundred-odd steps before its 100 000th, since it starts with coefficients already stored. The cells column counts the cells (and hence the centres) each run has built at that point. The time refers to the full run and includes building the partition.

Initial partition	Regions	PSNR (dB)	Cells (centres)	Time (s)
none	1	33.07	79 039	48
blocks 7×7	49	33.08	78 793	40
k -means (shared)	50	33.08	78 915	40
k -means (per-channel)	3×50	33.09	78 857	41
Lloyd (shared)	50	33.09	78 940	43
Lloyd (per-channel)	3×50	33.09	78 853	50

One aspect of the table deserves an honest word, because it seems to contradict the theory. We argued that *no* pre-partition should be best for quality — it is free to place its budget anywhere — yet even at Binev’s own budget the pre-partitioned runs end a hair *ahead* (about 0.4% lower error, a hundredth or two of a decibel). Part of the raw advantage was indeed the head start — comparing at equal coefficients narrows it — but a small, genuine residue survives, and that residue is the greedy caveat of Chapter 3 made visible. The algorithm is greedy, not optimal, so the free run does not actually attain the global near-best; a sensibly placed pre-partition can occasionally steer the greedy search onto a marginally better path than it would find on its own. The gap is well within this greedy noise — it reverses cell by cell and never exceeds a rounding error in PSNR — so it does not overturn the theoretical expectation, but rather measures the small slack between the greedy algorithm and the optimum it approximates.

4.2.4 THE REFINEMENT LOOP: WHICH TREE, WHICH LEAF

Two rules govern where the algorithm spends its next step: *which tree* to advance — one of the per-channel trees, or, when a pre-partition is used, one of the several trees of a channel — and *which leaf* of that tree to split. These are the choices in which Binev’s near-best theory is at stake, so they deserve the closest look.

The between-tree choice is decisive. Figure 4.4 compares the three tree-selection rules. Our generalization of Binev’s q -based rule (his own framework selects the leaf *within* a single tree and never has to choose *between* trees) and the modified-error rule are indistinguishable — their curves lie exactly on top of one another — and both pull steadily ahead of the naive “advance

the tree of largest error” rule as the budget grows, opening a gap of more than two decibels by the end (Table 4.5). The coincidence of the first two settles, in the negative, the concern raised in Chapter 3 that $\tilde{E}$ might over-prioritize heavily-refined trees: the gap between $\tilde{E}(\mathcal{R})$ and $q(\mathcal{R})$ is real and does widen as a tree grows, but the three channel trees grow to comparable depths, so the inflation is roughly proportional across them — and the *ranking* of the trees, the only thing the selection reads, almost never changes. In practice, then, choosing which channel to advance by how much it can still be *improved* — Binev’s rule — rather than by how large its current error simply is, is what keeps the multi-channel refinement efficient: the near-best theory of Chapter 2 paying off on real data.

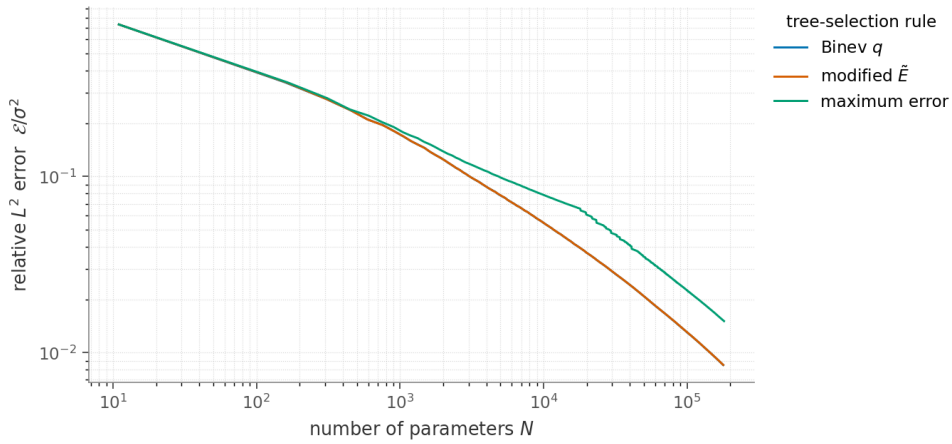


Figure 4.4: Between-tree selection. Binev’s q -rule and the modified-error rule coincide and outperform the maximum-error rule as the budget grows.

The within-tree choice — which leaf to split — is a far smaller lever. Binev’s rule and its greedy-at-birth variant are indistinguishable, and even plain maximum error trails them by only a few hundredths of a decibel (Table 4.5); the three curves would sit on top of one another, so the table suffices. The ranking, though, repeats the one seen between trees: the naive maximum-error rule is the worst of the three at both levels — decisively so between trees, marginally within one. The reason for the different margins is simple to state: on a natural image the leaf of largest error and the leaf Binev’s rule would pick very often coincide, so within a tree the modified errors have little room to make a difference; between trees, where the naive comparison goes systematically wrong, the theory pays off in full.

Table 4.5: Selection rules: end-of-run figures on the test image.

Rule	PSNR (dB)	Parameters N
tree-selection — Binev q	33.07	178 676
tree-selection — modified $\tilde{E}$	33.06	178 650
tree-selection — maximum error	30.58	180 966
leaf-selection — Binev	33.07	178 676
leaf-selection — greedy-at-birth	33.07	177 925
leaf-selection — maximum error	33.03	178 830

4.2.5 THE h -STEP: HOW A CELL IS SPLIT, AND WHAT IT COSTS

Two parameters shape the split itself: the metric that decides the boundary, and the rule that chooses the new centre. Figure 4.5 compares the centre-selection rules. The purely geometric max-distance rule, which simply takes the farthest pixel, is the weakest: at equal budget it carries about a tenth more error than the adaptive rules, which sample a few candidate centres and keep the one that best reduces the error. The random-adaptive rule and its hybrid with max-distance are, for their part, indistinguishable from each other — which is itself telling. The hybrid always includes the max-distance pixel among its candidates, so its coinciding with the purely random-adaptive rule means that that farthest pixel is almost never the centre finally chosen: another sign that the naive geometric criterion rarely lands on a good split.

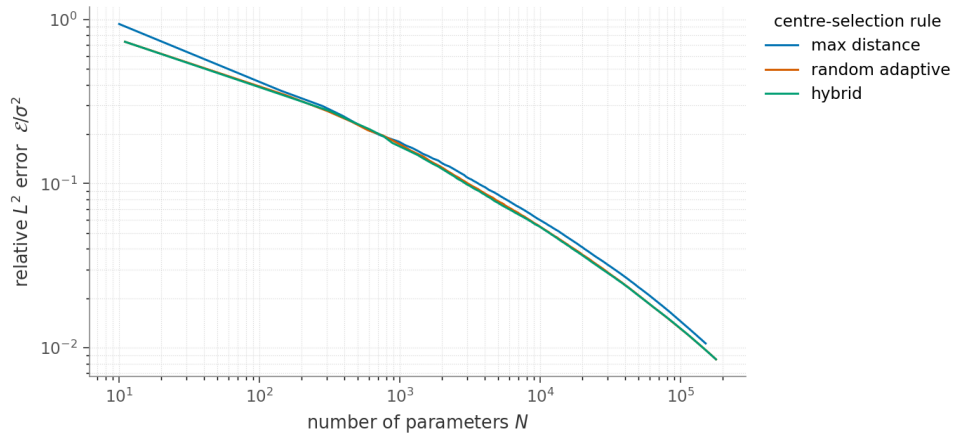


Figure 4.5: Centre-selection rule: the adaptive rules (random, hybrid) beat the purely geometric max-distance rule.

This is also the natural place for a precise look at time, because the centre-selection rule is exactly a trade of computation for quality. Choosing a centre by max-distance costs a single pass over the cell's pixels — only distances, no fitting — whereas each sampled candidate of the adaptive rules costs a least-squares fit of the two tentative children; sampling n candidates therefore costs about n fits per split. The measured times bear this out: on the test image the reference random-adaptive run takes about 1.7 times as long as the same run with the max-distance rule (roughly 48 against 28 seconds for the full refinement), with the hybrid — which samples just as many candidates — costing much the same as the random-adaptive. The rest of a split is cheap by comparison and is kept cheap by construction: the dyadic assignment only relabels the pixels that change side, each cell caches the indices of its own pixels so no split ever rescans the whole image, and the running errors and parameter counts are updated incrementally along the path back to the root rather than recomputed. The one genuinely unavoidable cost is the re-fit of each ancestor's polynomial on the way up — a handful of least-squares solves over progressively larger supports — and it is this, accumulated over the expensive early splits, that dominates the front-loaded cost noted above.

Both adaptive rules improve further as more candidates are sampled — Table 4.6 reads the trend off at equal steps: from five to fifty candidates the PSNR climbs steadily (by almost three quarters of a decibel), but the time climbs threefold with it, for exactly the reason just quantified: the number of candidates is a genuine quality-versus-time dial. At the limit sits the *fully-adaptive* rule discussed in Chapter 3, which tries *every* pixel of the cell as a candidate and keeps the best: it is the ideal, error-minimizing split, but its cost — a least-squares fit per pixel per split — is prohibitive at this scale, and random-adaptive with a handful of samples already sits so close to it that we use the sampled rule throughout and do not run the fully-adaptive one at full size.

Table 4.6: Number of sampled candidates n (test image, 100 000 steps): more candidates buy quality at a near-proportional cost in time.

Candidates n	PSNR (dB)	Parameters N	Time (s)
5	32.76	171 851	38
10	33.07	178 676	48
20	33.32	181 967	67
50	33.50	183 971	116

The splitting metric itself is only a minor lever on *quality* (Table 4.7): at equal budget the exponents $\alpha = 1, 2, 3$ are indistinguishable and the limit $\alpha = \infty$ is a touch worse — too small a difference to be worth its own plot. On *time*, though, the metric leaves a clear mark, and $\alpha = 2$ is the fastest of the four by construction. For $\alpha = 2$ the boundary between the two new centres is a straight line, the perpendicular bisector, so each pixel can be assigned by a single sign test — which side of that line it falls on — rather than by computing and comparing two distances; the implementation special-cases exactly this. The other exponents evaluate both L^α distances at every pixel, and $\alpha = 3$ pays on top of that for a fractional power, which makes it the slowest.

Table 4.7: Splitting metric d_α : end-of-run figures on the test image. At equal budget the quality differences are within noise ($\alpha = \infty$ marginally the worst); the times differ more, and $\alpha = 2$ is the fastest by construction (its boundary is a line, so assignment is a single sign test).

Metric exponent	PSNR (dB)	Parameters N	Time (s)
$\alpha = 1$	33.11	181 663	61
$\alpha = 2$	33.07	178 676	48
$\alpha = 3$	33.07	178 451	72
$\alpha = \infty$	32.90	177 615	62

4.2.6 THE p -STEP: GROWING THE POLYNOMIALS

Three parameters control how the local polynomials grow.

The *order in which monomials are added* (Table 4.8) is more consequential than one might first expect. Both adaptive orderings improve on the deterministic one, and in the direction that helps compression. The geometry-adaptive rule reaches a slightly better PSNR at fewer parameters and almost the same time — fewer, larger cells carrying richer polynomials, hence fewer centres to store — and the signal-adaptive rule is the best of the three on *both* counts at once, some 0.13 dB ahead at a visibly smaller budget, though at the longest running time. Why, then, does the reference keep the cheap deterministic ordering? First of all, certainly, for the running time, which the table shows to be clearly the lowest of the three; but also because it alone stores for free: under an adaptive ordering the file must also record *which* monomials each cell uses, a small overhead we quantify in Section 4.3, whereas for the deterministic ordering the active set is always a canonical prefix and needs no recording. The adaptive orderings are the

better compressors in the abstract parameter count N ; the deterministic one is the better *codec*, and that is the axis the reference optimizes.

Table 4.8: Monomial-ordering rule: end-of-run figures on the test image. Both adaptive orderings lower the budget and raise the quality; the deterministic one is kept only because it needs no monomial-identity bits (Section 4.3).

Monomial ordering	PSNR (dB)	Parameters N	Cells (centres)	Coefficients	Time (s)
deterministic	33.07	178 676	79 039	99 634	48
geometry-adaptive	33.09	178 042	78 445	99 594	54
signal-adaptive	33.20	173 552	73 804	99 745	62

It is worth seeing where the saving comes from. The coefficient count barely moves across the three orderings (about 99,600 each); the whole reduction in the budget N is in the *centres*. The number of final cells falls from 79,039 for the deterministic ordering to 73,804 for the signal-adaptive one — some 5,200 fewer cells, and hence 5,200 fewer centres to store. Better monomial choices, in other words, let a cell carry a richer polynomial and so cover ground that would otherwise be split into several constant cells. The final distribution of polynomial sizes bears this out (Table 4.9): the signal-adaptive ordering roughly doubles the share of two-monomial cells and raises the four-monomial share, at the expense of plain constants. Both effects — fewer centres and higher quality — push the same way, towards better compression; the only price is time.

Table 4.9: Distribution of polynomial size under the three monomial orderings — share of final cells (%) carrying 1, . . . , 6 monomials, and the resulting cell count. Adaptive orderings shift mass onto larger polynomials, lowering the cell (and centre) count.

Monomial ordering	1	2	3	4	5	6	cells
deterministic	87.6	5.6	3.2	1.5	1.0	1.1	79 039
geometry-adaptive	86.6	6.6	3.1	1.6	0.9	1.1	78 445
signal-adaptive	81.4	10.6	2.7	2.9	1.5	0.9	73 804

The *initial order* k gives one of the most instructive pictures in the study — provided it is read against the right budget. Comparing runs at a fixed number of *steps* is meaningless here, since $k = 1$ adds one parameter per step, $k = 2$ two and $k = 3$ three, so a larger k has simply spent more by any given step. Read against the stored budget N , the three trade-off

curves lie almost exactly on top of one another (so we omit the plot) — the near-best guarantee anticipated in Chapter 3, surviving on the coarser sequence of budgets $\{k, 2k, 3k, \dots\}$ — with a *slight* edge to the larger k . That small edge, taken at face value, would say that forcing every cell to a higher order is a free improvement, which cannot be right; it is worth pinning down where it comes from.

The resolution is that our budget N counts *two* things per cell — one centre and its coefficients — and the two respond to k in opposite ways (Table 4.10). A larger k starts every cell richer, so it reaches a given quality with *fewer, larger* cells; fewer cells means fewer centres, and since each cell costs exactly one centre whatever its order, a larger k spends less of N on centre overhead and more on actual coefficients. Against the pure *coefficient* budget — the quantity the near-best theory really bounds — the picture reverses: at a matched $\approx 99\,600$ coefficients, $k = 1$ is the more accurate, by about 0.7 dB, exactly as one would expect, since forcing order $\geq k$ wastes coefficients on the flat majority of cells that a single constant would have described just as well. The faint advantage of larger k against N is thus an accounting effect — richer cells amortise the one-centre-per-cell overhead — not a modelling gain. In modelling terms $k = 1$ is the most efficient, and that, with its being also the cheapest and the simplest, is why the reference keeps it.

Table 4.10: Initial order k at a matched *coefficient* budget ($\approx 99\,600$ coefficients, the quantity the near-best theory bounds). A larger k reaches it with far fewer cells — hence fewer centres, and a smaller stored N — but at a *higher* error: on the modelling budget $k = 1$ is the more efficient, as forcing order $\geq k$ wastes coefficients on the flat majority.

Initial order	Cells (centres)	Coefficients	Stored N	PSNR (dB)	Time (s)
$k = 1$ (constant)	79 039	99 634	178 676	33.07	48
$k = 2$ (partial-linear)	39 704	99 091	138 798	32.43	53
$k = 3$ (linear)	29 706	98 603	128 312	32.37	45

The *maximum order* $P_{\max}$ does impact quality, but with quickly diminishing returns above a certain level (Table 4.11): the step from a linear to a quadratic cap is clearly visible, the step to a cubic one still measurable, and beyond that the gains fade to hundredths of a decibel — consistent with Chapter 3’s observation that high-order polynomials are rarely the optimal local model on a natural image. Setting a threshold is therefore right; but *where* to set it must weigh the computational cost too, which keeps climbing steeply where the quality no longer follows: a quartic cap runs well over twice as slow as the reference, since each high-order re-fit along the climb is expensive. In theory the cap breaks the unbounded-order assumption behind the

near-best proof; in practice the break is invisible in quality, and a modest cap — the reference’s quadratic — is the sensible compromise.

Table 4.11: Maximum order $P_{\max}$: end-of-run figures on the test image. Quality barely moves; the time rises steeply.

Maximum order $P_{\max}$	PSNR (dB)	Parameters N	Time (s)
3 (linear)	32.98	185 535	37
6 (quadratic)	33.07	178 676	48
10 (cubic)	33.12	175 617	69
15 (quartic)	33.13	174 290	109

The pattern is the one just seen for the monomial ordering: as the cap rises the budget N falls and the quality rises, at a cost only in time — and again the saving is in the centres, since the coefficient count stays near 99,600 throughout while the cell count drops from 86,287 at a linear cap to 74,333 at a quartic one, as the occasional high-order cell absorbs what would otherwise be several constant ones. The steps between caps are uneven: from 3 to 6 the gain is large (nearly 7,000 fewer parameters), from 6 to 10 still clear (about 3,000), but from 10 to 15 almost imperceptible in quality and budget while the time nearly doubles — so the useful range lies somewhere between 10 and 15. One doubt is worth settling directly: are these gains really due to cells that *end* at high order, or does a high cap merely help the greedy search make better intermediate choices, leaving the final distribution unchanged? The data favours the former, if modestly: at a cap of 10 about 1.2% of the final cells actually carry more than six monomials, and at a cap of 15 about 1.4%, up to the full fifteenth-order model. They are few, but each replaces several smaller cells, which is enough to account for the drop in the number of centres.

Finally, the *coordinate normalization* is, as Chapter 3 argued, purely representational: local, global and no normalization at all produce a *bit-for-bit identical* error and partition on every run — the choice rescales the polynomial coefficients but changes no decision the algorithm makes. We record this as a consistency check rather than a comparison; its only role is the numerical stability of the high-order fits.

4.3 SECONDARY ANALYSES AND REAL COMPRESSION

The first-level study measured the method from the outside, through its error, its budget and its time. This section looks *inside* the compressed representation — what it is actually made of

— and then turns the abstract parameter count into a concrete file size.

4.3.1 THE ANATOMY OF THE REPRESENTATION

At the reference configuration the final representation of the test image consists of about 79 000 cells across the three channels. Figure 4.6 shows how many monomials their polynomials carry, and the picture is lopsided in a telling way: about 88% of the cells carry a single monomial (a *constant*), and the remaining eighth spreads thinly over two to six monomials — only about one cell in a hundred uses the full quadratic. The overwhelming majority of the image is thus described by flat patches, with polynomials spent only on the minority of cells where a gentle gradient earns its extra coefficients. This is the concrete counterpart of the qualitative picture in the progressive reconstruction, and it is the reason the representation is cheap: the method does not pay for a high-order model everywhere, only where the signal asks for it. It also explains why the maximum order $P_{\max}$ is such a weak lever — the models competing for the cap are a small tail of an already small tail.

Two consequences of this lopsidedness are worth drawing out. First, it is exactly why forcing a high *initial* order — a $k > 1$ that makes *every* new cell at least linear — probably is the wrong default: the constant majority would then carry coefficients it does not need, whereas letting the order grow adaptively spends them only on the minority of cells that repay them. Second, the cells that *do* grow past a constant earn their keep twice. A single linear or quadratic patch covers ground that would otherwise have to be split into several constant cells, and each cell it saves is also one fewer *centre* to store: the higher-order polynomials buy down not only the error but the size of the partition itself, which is why they lower the parameter count rather than raise it.

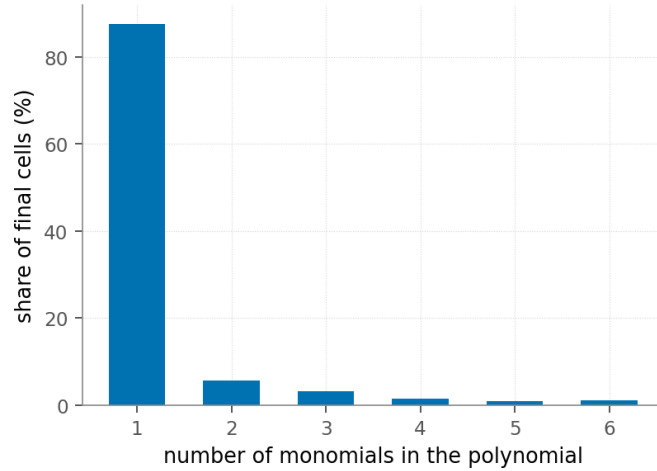


Figure 4.6: How many monomials the final cells' polynomials carry. The great majority are constants.

4.3.2 FROM PARAMETERS TO BYTES

The parameter count N is a faithful proxy for the compressed size, but the true size in bytes needs a model of how each piece is stored. As we have stressed more than once, the stored representation is not only centres and coefficients: it also carries a small header, the descriptors of the initial partition when one is used, and the handful of numbers of the colour transform. Setting those fixed or minor costs aside, the size is dominated by the *centres* that replay the partition and the *coefficients* of each final cell, with — under an adaptive ordering — the almost-free monomial identities.

What follows is a mostly theoretical account, developed for the plain case of no initial partition: one tree per channel, grown from the whole image. The aim is to understand what drives the size and to bracket it between an optimistic and a pessimistic estimate, not to specify a finished byte format. An initial partition is not a mere change of constants here: it replaces the single root of WH pixels by R region-roots of about WH/R pixels each, so every tree starts from a much smaller cell and its centres are chosen from correspondingly smaller sets. Since the centre cost is driven precisely by the size of the cell a centre is picked from, an initial partition significantly lowers the centre estimate — roughly, WH is replaced by WH/R inside the logarithm, summed over the R region-trees. The formulae below stay valid but must be re-instantiated per region.

THE CENTRES. A centre is a single pixel index, and what makes it cheap to encode is *which set the index ranges over*. When a cell is split, the new centre is chosen among the pixels of the cell being divided, so its cost is set by the size of that *parent* cell Δ — the cell it is picked *from* — and not by the child cell it ends up representing: after the split the centre lives in one of the two smaller children, but it was selected out of the whole parent, and that is what must be encoded. A centre chosen inside a parent of $|\Delta|$ pixels therefore costs $\lceil \log_2 |\Delta| \rceil$ bits, far fewer than $\log_2(WH)$ once the cells deep in the tree are small.

To write the total cleanly, let $\Delta_1, \dots, \Delta_L$ be the sequence of cells from which the $L = |\mathcal{L}(T_N)|$ centres are taken, in encoding order, with $\Delta_1 := V$ the whole image (from which the root centre is chosen) and each later Δ_j the parent cell that the j -th centre splits. Then

$$C_{\text{centres}} = \sum_{j=1}^L \lceil \log_2 |\Delta_j| \rceil, \quad \Delta_1 = V, |V| = WH. \quad (4.1)$$

The sum depends on the *shape* of the tree, and two extremes bracket it. In the pessimistic case, a degenerate “comb” peels a tiny cell off a still-huge one at every split, so each $|\Delta_j| \approx WH$ and $C_{\text{centres}}^{\text{comb}} \approx L \log_2(WH)$: every centre paid at full image resolution. In the optimistic case, a balanced tree, the cells shrink as fast as possible, so the deep splits act on small cells — this is in fact optimal, since a comb instead keeps splitting near-full-size cells and pays $\log_2(WH)$ over and over. A balanced binary tree has depth $d = \log_2 L$, and its splits fall into d levels: level i (for $i = 0, \dots, d-1$) contains 2^i cells of size $WH/2^i$, each costing $\log_2(WH/2^i)$. Summing, and evaluating the two standard geometric sums $\sum_{i=0}^{d-1} 2^i = 2^d - 1$ and $\sum_{i=0}^{d-1} i 2^i = (d-2) 2^d + 2$ at $2^d = L$ — the number of leaves, and hence of centres, of the balanced tree of depth $d = \log_2 L$ — so that they read $L - 1$ and $(d-2)L + 2$,

$$C_{\text{centres}}^{\text{bal}} = \sum_{i=0}^{d-1} 2^i \log_2 \left(\frac{WH}{2^i} \right) = L \log_2 \left(\frac{WH}{L} \right) + 2L - 2. \quad (4.2)$$

Each centre thus costs on average about $\log_2(WH/L) + 2$ bits, roughly $\log_2 L$ bits less than the naive $\log_2(WH)$; both bounds lie above the information-theoretic floor $\log_2 \binom{WH}{L}$ for placing L centres among WH pixels. The image carries three such trees, one per channel, so the totals are summed over the three, each over its own L leaves.

THE COEFFICIENTS. Each coefficient is stored at a fixed precision. Taking two bytes per coefficient, the natural half-precision size, the coefficient block costs

$$C_{\text{coeff}} = 16 \sum_{\text{cells}} n_{\text{coeff}} \text{ bits.} \quad (4.3)$$

Unlike the centres, this block does not depend on the tree's shape, only on how many coefficients there are. The *monomial identities* are almost free, and the reason is worth spelling out. Under the deterministic ordering they cost nothing: the active set is a canonical prefix, so the count s alone identifies it. Under an adaptive ordering the file must record which monomials a cell uses — but far less information is needed than one might fear, because the progression works degree by degree: every degree is completed before the next is opened (Chapter 3). For a cell with s coefficients, then, the filled degrees are known from s itself, and all that must be identified is *which* r of the $m = d+1$ monomials of the top degree d are active: $\lceil \log_2 \binom{m}{r} \rceil$ bits, which is zero whenever the top degree is complete — in particular for every constant, full-linear and full-quadratic cell — and at most a few bits otherwise (e.g. a cell with 8 coefficients surely holds the full quadratic and needs only $\lceil \log_2 \binom{4}{2} \rceil = 3$ bits to name its two cubic monomials). Averaged over a run, this comes to a fraction of a bit per cell.

FROM STEPS TO BITS. Two facts, measured on the reference run, put everything in terms of the number of refinement steps s . First, the coefficients track the steps closely but not exactly: each step adds a split and, ascending the tree, normally one new coefficient, so $b \approx s$; the count falls just short of s because on some ascents a region has already reached the maximum order P_{max} , and there the ascent adds no coefficient, only updates the errors. Second, the final cells are a fixed fraction of the coefficients, $c \approx 0.79 b$, because the polynomial-size distribution stays roughly stable along the whole growth, a fact we verified empirically on the reference run; hence $c \approx 0.79 s$. Substituting $c \approx 0.79 s$ into the balanced centre estimate for three trees, $C_{\text{centres}} \approx c \log_2(3WH/c) + 2c$, and adding the coefficient block $16 b \approx 16 s$,

$$\text{bits}(s) \approx \underbrace{0.79 s \log_2 \left(\frac{3.8 WH}{s} \right)}_{\text{centres (optimistic)}} + 1.6 s + \underbrace{16 s}_{\text{coefficients}}. \quad (4.4)$$

The coefficient block is strictly linear in s ; the centre block adds only a slowly-varying $\log_2(WH/s)$ factor. The file therefore grows essentially linearly in the number of steps, $O(s \log(WH/s))$.

TWO REFINEMENTS. The account above needs two further touches to be complete — a term it leaves implicit, and a saving it has not yet claimed. The first concerns the centres: the prices of Equation (4.1) presuppose that the decoder knows, split after split, *which* cell the next centre is drawn from. It is enough to attach two extra bits to every split, stating which of its two children will be split in turn — neither, the left one, the right one, or both; together with a fixed, deterministic order of visit, these flags let the decoder replay the entire hierarchy of cells (the chronological order of the growth, on which the partition does not depend, is never stored). The selection therefore costs two bits per cell: a marginal surcharge on the centre bill. The second concerns the coefficients. As Table 4.9 showed a few pages back, the vast majority of the final cells — 87.6% on the reference run — carry a *constant* polynomial, and the only coefficient of a constant cell is nothing more than the average channel value of its pixels: precisely the information that occupies *one* byte in the decoded image. Spending two bytes on a number that the decoded image will render in one is plainly wasteful, so the refined account stores the coefficient of every constant cell in a single byte and keeps two bytes for all the others. How coarsely the coefficients can in general be quantized before the quantization error competes with the approximation error is a question that deserves a study of its own, and lies beyond our scope; this particular case, though, needs no study: the correction halves the cost of some seventy per cent of the coefficients, which are themselves the dominant block of the file, and we verified on the reference run that the loss it induces is about five hundredths of a decibel. Together the two refinements replace the $16s$ coefficient term of Equation (4.4) by roughly $10.4s + 1.6s$ and lower the reference estimate from about 253 to about 205 KB; all sizes and rates quoted from here on use this refined account.

A WORKED EXAMPLE. Only here do we put numbers on it, for the reference reconstruction of the test image (the polynomial-size shares behind c are those of Figure 4.6). One clarification on how these numbers are obtained: this is still an *estimate* — no actual file is written — but not the coarse closed-form one of Equation (4.4). It is computed exactly, term by term, on the final tree: for each split of the optimal subtree we charge the true $\lceil \log_2 |\Delta| \rceil$ bits of its actual parent cell, and for the coefficients the exact count the run produced. Table 4.12 collects the result. The coefficients dominate in either accounting — about three quarters of the two-byte total, and still over sixty per cent of the refined one — while the centres are the smaller, shape-dependent part; the whole compressed object comes out on the order of a couple of hundred kilobytes, a healthy saving over the raw three bytes per pixel of a 786 432-pixel image.

Table 4.12: Estimated byte budget for the reference reconstruction of the test image. The centres are bracketed by the balanced and comb estimates; the coefficients are shape-independent, and are shown both in the plain two-byte accounting and in the refined one (one byte for the constant cells).

Piece	Estimate	Size
Centres — balanced (optimistic)	$L \log_2(WH/L) + 2L$	≈ 67 KB
Centres — comb (pessimistic)	$L \log_2(WH)$	≈ 190 KB
Cell selection	2 bits per cell	≈ 19 KB
Coefficients (two bytes each)	$16b$	≈ 195 KB
Coefficients (one byte for constants)	$16b - 8b_{\text{const}}$	≈ 127 KB
Total (optimistic, two-byte coefficients)	balanced + coefficients	≈ 261 KB
Total (refined)	balanced + selection + refined	≈ 213 KB

AN ESTIMATED SIZE, AND A COMPARISON WITH JPEG. The model above turns each run into an estimated file size, so we can place the method on a rate–distortion curve without committing to any particular serializer. Evaluating the refined estimate on the real tree at a range of budgets — the true centre cost of each split, the two-bit cell selection, one byte per constant coefficient and two for the others, and the small header — gives, at the reference quality of 33.0 dB, about 205 KB, or 2.1 bits per pixel: an elevenfold saving over the raw 24. Stopped earlier the estimate is smaller still, and Figure 4.7 plots the whole curve against JPEG on the same image.

The comparison is closer than a naive byte count would suggest. Across the range where the two overlap, JPEG stays above our estimate — by about one decibel at the low end of the shared range, growing to some three decibels at the highest rates. This is expected: JPEG is a mature codec, with a quantization table and an entropy stage tuned over decades, whereas our estimate stores each coefficient at a plain fixed precision, with no entropy coding at all. At the very lowest rates the picture even reverses: below roughly a quarter of a bit per pixel, where JPEG can no longer produce a valid file, our method keeps degrading gracefully and matches or slightly edges it.

We make no claim to a competitive codec. The contribution of the thesis is the adaptive *hp* construction and its near-best behaviour; this estimate and this curve are simply its faithful, end-to-end reading. The gap that remains to JPEG is just what a dedicated entropy stage would close — quantizing the coefficients more tightly and coding them, and choosing the stopping point for rate rather than for steps — a natural and promising next step, but a separate one.

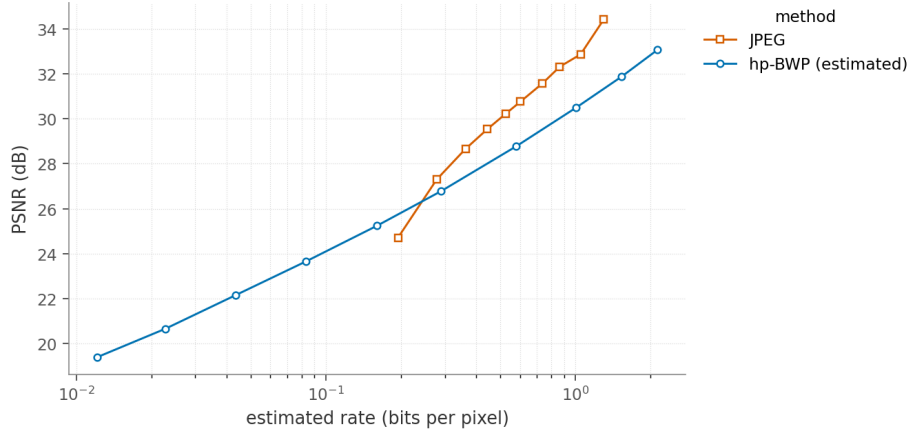


Figure 4.7: Rate–distortion (PSNR against bits per pixel) on the test image: our *hp*-BWP method against JPEG. Our rate is the *estimated* byte size from the refined model above (the true centre cost, the two-bit cell selection, one byte per constant coefficient and two for the others, and a small header), not a naive file count. JPEG stays one to three decibels above across the shared range; below about 0.25 bits per pixel, where JPEG stops, our method matches or edges it and continues to lower rates.

4.3.3 DECODING: THE TIME TO OPEN THE FILE

Everything so far has measured *encoding* time — the cost of building the compressed representation. The complementary quantity is *decoding* time: the cost of opening a compressed file and rebuilding the image. Decoding is a far lighter task, because it involves no search and no fitting: the coefficients are already known, so one only has to replay the stored centre sequence to rebuild the partition and then evaluate each cell’s polynomial over its pixels. Measured on the test image, decoding is indeed twenty to thirty times cheaper than encoding (Table 4.13): a file that took about fifty seconds to produce reopens in roughly two.

Table 4.13: Decoding versus encoding on the test image (single thread).

Steps	Final cells	Encode (s)	Decode (s)	Encode/decode
5 000	3 732	13.0	0.41	32×
25 000	18 832	22.9	0.85	27×
100 000	79 039	49.0	2.40	20×

The decode splits neatly into two parts of a very different nature. The *partition replay* — reading the ordered centres and re-performing each split — is inherently *sequential*: the cell a centre lands in exists only after all the earlier splits have been applied, so the order cannot be

changed. The *polynomial evaluation* — computing each cell’s polynomial over its pixels — is, by contrast, *fully parallelizable*: the cells are disjoint and independent, and the three colour channels are independent as well. At the smallest budgets the sequential replay dominates; by the largest it is about half the work, the evaluation the other half.

These tests use no parallelization at all: the decoder runs on a single thread. Since the polynomial evaluation is independent across cells and channels, a parallel implementation could speed the decoding up further and significantly; at a couple of seconds for a full-quality image, however, it is already fast enough in practice.

4.4 THE OPTIMAL CONFIGURATION

The first-level study measured each design choice in isolation, and three emerged as genuine quality levers: the signal-adaptive monomial ordering, a higher polynomial cap $P_{\max}$, and a larger candidate sample n . Each buys its decibels at a computational price. The natural closing question is what happens when they are combined — do the gains add up, or does one lever absorb the others? — and what the combination costs. We therefore ran a dedicated experiment comparing, beyond those isolated parameters, a configuration that switches them all on at once; Table 4.14 climbs the ladder, from the reference through each lever alone at its strongest useful level up to the full combination, all at an equal 100 000 steps.

Table 4.14: The ladder of levers: the reference, each quality lever alone, and their combinations, at an equal 100 000 steps.

Configuration	PSNR (dB)	N	Cells	Coefficients	Size (KB)	Time (s)
reference	33.07	178 676	79 039	99 634	205.0	51
signal-adaptive	33.20	173 552	73 804	99 745	211.6	75
$n = 50$	33.50	183 971	84 247	99 721	203.7	128
$P_{\max} = 15$	33.13	174 290	74 333	99 954	205.2	114
signal + $P_{\max}15$	33.25	168 673	68 694	99 976	211.9	226
optimal (signal + $P_{\max}15$ + $n50$)	33.63	177 129	77 147	99 979	209.3	305

On quality, the gains compound almost exactly. Over the reference the three levers alone buy +0.13, +0.43 and +0.06 dB; switched on together they deliver +0.56 dB, within a few hundredths of the plain sum. The combined run, which we call the *optimal configuration*, reaches 33.63 dB — the best figure measured in this thesis — and the cell counts in the table show *how*.

The cap alone brings no enormous contribution, but combined with the signal-adaptive ordering the two act on the same front, building complex and efficient local polynomials: together they cut the cell count to 68 694, far below what either achieves alone, because every well-built polynomial absorbs cells that would otherwise have to be split. The candidate sample works in the opposite direction: it is the strongest lever in isolation, yet it *raises* the cell count (to 84 247 alone), since candidates chosen that well identify good bipartitions from the very start, and the algorithm then often prefers opening new cells to enriching the polynomials of existing ones. In the optimal configuration the two tendencies partly offset, and the coefficient count stays pinned near 100 000 throughout: what the levers really redistribute is *where* those coefficients live.

The costs compound too, and not all in the same way. The optimal run takes 305 seconds against the 51 of the reference. The signal-adaptive ordering raises the baseline cost of fitting a polynomial in a cell, since every candidate monomial must be tried before the best is kept; the cap $P_{\max}$ lengthens the climb back up the tree, and hence the number of fits per step — and the two couple exactly, each of the extra fits demanded by the cap now paying the audition demanded by the ordering. The deltas make the coupling visible: alone, the two levers add 24 and 63 seconds to the reference, but combined they add 175, twice their sum. The candidate sample, instead, acts on the split and does not interact with either: its surcharge is essentially the same whether it lands on the reference ($128 - 51 = 77$ seconds) or on top of the other two levers ($305 - 226 = 79$).

On size, at equal steps the optimal file is marginally *larger* (209.3 against 205.0 KB): the identity of the adaptively-chosen monomials must be stored, and the refined accounting adds a second, subtler reason — the very success of the adaptive machinery replaces constant cells with richer ones, and every constant cell lost is a coefficient that no longer fits in a single byte. The honest comparison, as always, is the rate–distortion curve — panel (b) of Figure 4.8 — and there the ranking is unambiguous: at every file size, the optimal configuration sits about half a decibel above the reference, from the lowest rates to the highest.

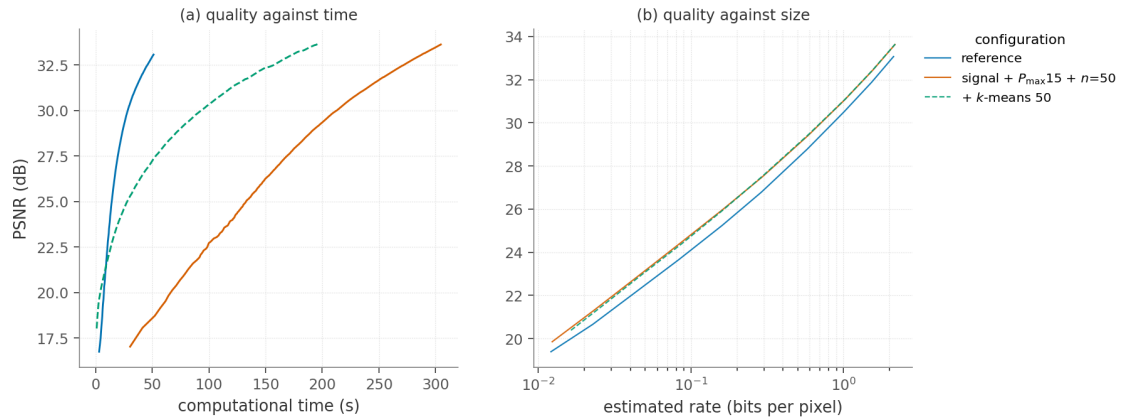


Figure 4.8: The reference, the optimal configuration and its pre-partitioned variant. (a) quality against computational time: the pre-partitioned run climbs towards the same final quality along a visibly faster path. (b) quality against estimated size: the optimal configuration and its pre-partitioned variant coincide, about half a decibel above the reference at every rate.

BUYING THE TIME BACK: THE INITIAL PARTITION. Six times the computation — the factor measured in this simulation, though it does not scale linearly with the step count — is a heavy price, and this is exactly the place where the initial partition, so far a lever of modest value, earns its keep. The reason was established early in the chapter: the cost of a run is front-loaded onto the first splits, which act on enormous cells — and each of the three levers multiplies precisely that early cost: fifty candidate fits per split, high-order re-fits, signal-driven auditions, all over cells of hundreds of thousands of pixels. An initial partition makes the ruinous early splits simply never happen. Table 4.15 runs the optimal configuration with a shared k -means partition at 50 and at 100 regions.

Table 4.15: The optimal configuration with and without an initial partition (100 000 steps; the time includes building the partition).

Configuration	PSNR (dB)	N	Cells	Coefficients	Size (KB)	Time (s)
optimal, no pre-partition	33.63	177 129	77 147	99 979	209.3	305
optimal + k -means 50	33.64	176 669	76 487	100 129	209.9	196
optimal + k -means 100	33.64	177 203	76 832	100 268	209.8	172

The result is as clean as one could hope. The quality does not move — if anything it rises by a hundredth of a decibel, and it is worth being precise about why: not because the pre-partition helps the search (if anything, freezing boundaries breaks the greedy approach), but

because the run starts from an already-advanced refinement, with fifty or a hundred regions and their parameters in place before the first split. The comparison that matters is the pair of curves in Figure 4.8, and there the quality is not significantly impacted. The size does not move either. But the time falls by a third at fifty regions and by almost half at a hundred (196 and 172 seconds against 305), and panel (a) of the figure shows the whole trajectory: the pre-partitioned run climbs towards the same final quality along a visibly faster path.

It is worth appreciating how the two findings of this section fit together. On the reference configuration the pre-partition saved a modest fifth of a fifty-second run — a curiosity more than a feature. On the optimal configuration it saves over a hundred seconds, because the cost it removes (the early splits) is exactly the cost the quality levers inflate. The heavier the configuration, the more the pre-partition gives back: the levers buy quality with time, and the initial partition buys much of that time back for free. This combined recipe — the adaptive ordering, a generous cap and candidate sample, over a cheap initial partition — is the configuration we would recommend whenever computation allows; paired with a more advanced, parallelized implementation, it is also the natural starting point for a production codec.

5

Conclusions

This thesis set out to turn an abstract approximation framework into a working compression method, and to measure, honestly and one choice at a time, what every part of it contributes. We started from the near-best theory of Binev, which guarantees that a greedy, incrementally grown tree can stay provably close to the best achievable *hp*-adaptive approximation, but which leaves many aspects open, and some entirely untreated: how a cell is split, which monomial enriches a local polynomial, how several signals share one budget. The work of the thesis has been to fill, modify and extend that framework into a complete algorithm (Chapter 3), to implement it as a usable Python library, and to submit every design choice to the same experimental discipline (Chapter 4): one reference configuration, one parameter varied at a time, and the whole trade-off curve of error against budget, time and size as the unit of judgement. The geometric backbone throughout is the binary wedge partitioning tree, whose one-new-centre-per-split structure makes the partition itself remarkably cheap to store; the theoretical backbone is Binev’s pair of modified errors, which decide at every step where the next parameter is best spent.

The experiments returned a picture that is coherent, and in places genuinely instructive. The single strongest lever is not part of the adaptive machinery at all: decorrelating the colour channels before approximating them, ideally with the principal-component transform, is worth about two and a half decibels at any budget, essentially for free. Binev’s selection, generalized to choose among the channel trees, proved decisive as well: advancing the channel by how much it can still be improved keeps more than two decibels of advantage over the naive largest-error

rule.

Looking inside the representation explained most of the remaining behaviour. The final partition is dominated by constant cells, close to ninety per cent of the total, with the richer polynomials concentrated on the minority of cells that repay them; and since every cell costs exactly one centre, whatever raises the quality of the local models, a better monomial ordering, a higher polynomial cap, tends to *merge* cells and so to save centres, the cheap kind of parameter, rather than coefficients, the expensive kind. This asymmetry between centres and coefficients ran through the whole second half of the study. It resolved the apparent paradox of the initial order, whose advantage at equal parameter count turned out to be an accounting effect of the one-centre-per-cell overhead rather than a modelling gain; it explained why the adaptive monomial orderings compress the abstract parameter count better than they compress the file, once the identity of their monomials is billed; and it kept the byte model of Section 4.3 honest, with the true cost of a centre derived from the cell it is chosen in, and the file growing essentially linearly in the number of refinement steps.

The byte model also allowed a fair external comparison. Measured on its estimated size, the method stands between one and three decibels below JPEG across the range the two share, which is the expected distance from a mature codec with a tuned quantization and entropy stage, given that we store plain fixed-precision coefficients with no entropy coding at all; and at very low rates, below the point where JPEG stops producing valid files, the method keeps degrading gracefully and edges ahead. We make no claim to a competitive codec: the point of the comparison is that the adaptive *hp* construction survives an end-to-end reading, from theory to bits, with no step of the account left to the imagination.

The closing experiment brought the pieces together. The three genuine quality levers, the signal-adaptive ordering, a generous polynomial cap and a generous candidate sample, compound almost exactly when combined, reaching the best figure measured in the thesis at about half a decibel above the reference; but their costs compound too, and the combined run is six times slower. This is precisely where the initial partition, elsewhere a lever of modest value, earns its keep: because the cost of a run is front-loaded onto the first splits over enormous cells, and because the quality levers inflate exactly that early cost, a cheap *k*-means partition buys back a third to a half of the running time at no measurable cost in quality or size. The practical recipe of this thesis is therefore a combined one: adaptive orderings and generous search over an inexpensive initial partition, and, whenever computation allows, all of it together.

Several limitations bound these conclusions, and we prefer to state them plainly. The quantitative study lives on a single, deliberately mixed test image, and a systematic multi-image study

remains to be done. The implementation is a research prototype in Python: the times we report are internally consistent and support every relative claim, but they are not a benchmark, and both encoder and decoder are single-threaded even where the work, local fits on one side, polynomial evaluation on the other, is naturally parallel. The polynomial cap, however useful in practice, formally breaks the unbounded-order assumption behind the near-best proof; and our multi-channel selection rule, the generalization of Binev’s within-tree criterion to a forest of trees, is adopted as the natural well-motivated choice, without a formal extension of the guarantee. None of these gaps undermines the measurements; each of them marks an edge of what the thesis can claim.

The same edges trace the directions we consider most promising. The nearest and most concrete is the byte side: the non-constant coefficients are still stored at more precision than the approximation actually needs, so a careful quantization, followed by an entropy stage over coefficients and centres alike, would close much of the distance to JPEG without touching the underlying construction; choosing the stopping point for rate rather than for step count belongs to the same effort. On the experimental side, the one-parameter-at-a-time design should grow into a study of interactions, a grid over the few parameters that matter, including the candidate sample, and the cost model of the centres should be re-instantiated for pre-partitioned runs, whose region roots change the sizes every centre is drawn from. On the algorithmic side, several of the design choices we fixed could be made more supple, and each would repay investigation: more advanced programming techniques would attack the running times directly; the candidate sample need not be a fixed count, and could scale with the size of the cell being split, or trade randomness for a fixed but well-distributed design; and the growth of the polynomials could shed the constraint of completing one degree before opening the next, or adopt adaptive orderings cheap enough not to require fitting every candidate polynomial in full. On the theoretical side, the formal extension of the near-best result to the multi-channel forest is the natural next step. And beyond images, the entire construction was built on a metric space rather than on a pixel grid: nothing in the algorithm requires the domain to be an image, and applying it to genuine graph signals, with coordinate signals chosen for the domain at hand, is the generalization the framework was designed for from the start.

References

- [1] W. Erb, “Graph wedgelets: Adaptive data compression on graphs based on binary wedge partitioning trees and geometric wavelets,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 9, pp. 24–34, 2023.
- [2] D. L. Donoho, “Wedgelets: Nearly-minimax estimation of edges,” *The Annals of Statistics*, vol. 27, no. 3, pp. 859–897, 1999.
- [3] P. Binev, “Tree approximation for hp-adaptivity,” *SIAM Journal on Numerical Analysis*, vol. 56, no. 6, pp. 3346–3357, 2018.
- [4] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,” *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.
- [5] R. A. DeVore, “Nonlinear approximation,” *Acta Numerica*, vol. 7, pp. 51–150, 1998.
- [6] H. Hotelling, “Analysis of a complex of statistical variables into principal components,” *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933.
- [7] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed., ser. Springer Series in Statistics. New York: Springer, 2002.
- [8] International Telecommunication Union, “Recommendation itu-r bt.601-7: Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios,” ITU-R, Tech. Rep., 2011.
- [9] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1. University of California Press, 1967, pp. 281–297.
- [10] S. P. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

- [11] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [12] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," in *Electronics Letters*, vol. 44, no. 13. IET, 2008, pp. 800–801.