Minimum Description Length Model Selection

de Rooij, S.

Publication date
2008

Chapter 6

Individual Sequence Rate Distortion

Rate-distortion theory analyses communication over a channel under a constraint on the number of transmitted bits, the "rate". It currently serves as the theoretical underpinning for many important applications such as lossy compression and denoising, or more generally, applications that require a separation of structure and noise in the input data.

Classical rate-distortion theory evolved from Shannon's theory of communication [79]. It studies the trade-off between the rate and the achievable fidelity of the transmitted representation under some distortion function, where the analysis is carried out *in expectation* under some source distribution. Therefore the theory can only be meaningfully applied if we have some reasonable idea as to the distribution on objects that we want to compress lossily. While lossy compression is ubiquitous, propositions with regard to the underlying distribution tend to be ad-hoc, and necessarily so, because (1) it is a questionable assumption that the objects that we submit to lossy compression are all drawn from the same probability distribution, or indeed that they are drawn from a distribution at all, and (2) even if a true source distribution is known to exist, in most applications the sample space is so large that it is extremely hard to determine what it is like: objects that occur in practice very often exhibit more structure than predicted by the used source model.

For large outcome spaces then, it becomes important to consider structural properties of *individual objects*. For example, if the rate is low, then we may still be able to transmit objects that have a very regular structure without introducing any distortion, but this becomes impossible for objects with high information density. This point of view underlies some recent research in the lossy compression community [77]. At about the same time, a rate-distortion theory which allows analysis of individual objects has been developed within the framework of Kolmogorov complexity [60]. It defines a rate-distortion function not with respect to a source distribution, but with respect to an individual source word. Every source word thus obtains its own associated rate-distortion function.

We will first give a brief introduction to algorithmic rate-distortion theory in Section 6.1. We also describe a novel generalisation of the theory to settings with side information, and we describe two distinct applications of the theory, namely lossy compression and denoising.

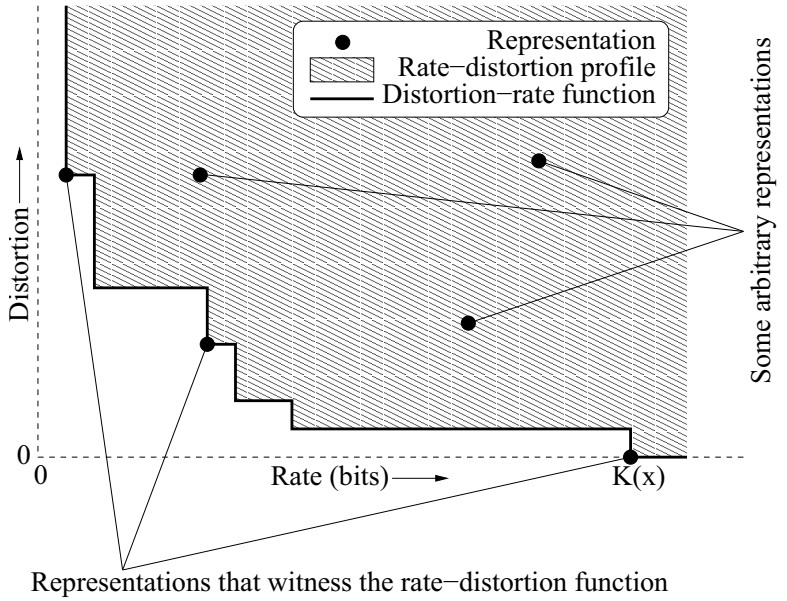
Algorithmic rate-distortion theory is based on Kolmogorov complexity, which is not computable. We nevertheless cross the bridge between theory and practice in Section 6.2, by approximating Kolmogorov complexity by the compressed size of the object by a general purpose data compression algorithm. Even so, approximating the rate-distortion function is a difficult search problem. We motivate and outline the genetic algorithm we used to approximate the rate-distortion function.

In Section 6.3 we describe four experiments in lossy compression and denoising. The results are presented and discussed in Section 6.4. Then, in Section 6.5 we take a step back and discuss to what extent our practical approach yields a faithful approximation of the theoretical algorithmic rate-distortion function, continued by a discussion of how such a practical approach fits within the framework of MDL model selection (Section 6.6). We end with a conclusion in Section 6.7.

6.1 Algorithmic Rate-Distortion

Suppose we want to communicate objects x from a set of source words $\mathcal{X}$ using at most r bits per object. We call r the *rate*. We locate a good *representation* of x within a finite set $\mathcal{Y}$, which may be different from $\mathcal{X}$ in general (but we usually have $\mathcal{X} = \mathcal{Y}$ in this text). The lack of fidelity of a representation y is quantified by a distortion function $d : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.

Figure 6.1 Rate-distortion profile and distortion-rate function



The Kolmogorov complexity of y, denoted $K(y)$, is the length of the shortest program that constructs y. More precisely, it is the length of the shortest input to a fixed universal binary prefix machine that will output y and then halt; also see the textbook [60]. We can transmit any representation y that has $K(y) \leq r$, the receiver can then run the program to obtain y and is thus able to reconstruct x up to distortion $d(x, y)$. Define the *rate-distortion profile* P_x *of the source word* x as the set of pairs $\langle r, a \rangle$ such that there is a representation $y \in \mathcal{Y}$ with $d(x, y) \leq a$ and $K(y) \leq r$. The possible combinations of r and a can also be characterised by the *rate-distortion function of the source word* x, which is defined as $r_x(a) = \min\{r : \langle r, a \rangle \in P_x\}$, or by the *distortion-rate function of the source word* x, which is defined as $d_x(r) = \min\{a : \langle r, a \rangle \in P_x\}$. These two functions are somewhat like inverses of each other; although strictly speaking they are not since they are monotonic but not strictly monotonic. A representation y is said to *witness* the rate-distortion function of x if $r_x(d(x, y)) = K(y)$. These definitions are illustrated in Figure 6.1.

Algorithmic rate-distortion theory is developed and treated in much more detail in [92]. It is a generalisation of Kolmogorov's structure function theory, see [93]. We generalise the algorithmic rate-distortion framework, so that it can accommodate side information. Suppose that we want to transmit a source word $x \in \mathcal{X}$ and we have chosen a representation $y \in \mathcal{Y}$ as before. The encoder and decoder often share a lot of information: both might know that grass is green and the sky is blue, they might share a common language, and so on. They would not need to transmit such information. If encoder and decoder share some information z, then the programs they transmit to compute the representation y may use this side information z. Such programs can be much shorter, and are never much longer, than their counterparts that do not use side information. This can be formalised by switching to the *conditional* Kolmogorov complexity $K(y|z)$, which is the length of the shortest Turing machine program that constructs y on input z. We redefine $K(y) = K(y|\epsilon)$, where ϵ is the empty sequence, so that $K(y|z) \leq K(y) + O(1)$: the length of the shortest program for y can never significantly increase when side information is provided, but it might certainly decrease when y and z share a lot of information [60]. We change the definitions as follows: The *rate-distortion profile of the source word* x *with side information* z is the set of pairs $\langle r, a \rangle$ such that there is a representation $y \in \mathcal{Y}$ with $d(x, y) \leq a$ and $K(y|z) \leq r$. The definitions of the rate-distortion function and the distortion-rate function are similarly changed. Henceforth we will omit mention of the side information z unless it is relevant to the discussion.

While this generalisation seems very natural the authors are not aware of earlier proposals along these lines. In Section 6.4 we will demonstrate one use for this generalised rate-distortion theory: removal of spelling errors in written text, an example where denoising is not practical without use of side information.

6.1.1 Distortion Spheres, the Minimal Sufficient Statistic

A representation y that witnesses the rate-distortion function is the best possible rendering of the source object x at the given rate because it minimises the distortion, but if the rate is lower than $K(x)$, then some information is necessarily lost. Since one of our goals is to find the best possible separation between structure and noise in the data, it is important to determine to what extent the discarded information is noise.

Given a representation y and the distortion $a = d(x, y)$, we can find the source object x somewhere on the list of all $x' \in \mathcal{X}$ that satisfy $d(x', y) = a$. The information conveyed about x by y and a is precisely, that x can be found on this list. We call such a list a *distortion sphere*. A distortion sphere of radius a, centred around y is defined as follows:

$$S_y(a) := \{x' \in \mathcal{X} : d(x', y) = a\}. \tag{6.1}$$

If x is a completely random element of this list, then the discarded information is pure "white" noise. Moreover, all random elements in the list share all "simply described" (in the sense of having low Kolmogorov complexity) properties that x satisfies. Hence, with respect to the "simply described" properties, every such random element is as good as x, see [92] for more details. In such cases a literal specification of the index of any object x' in the list (in particular the original object x) is the most efficient code for that x', given only that it is in $S_y(a)$. A fixed-length, literal code requires $\log |S_y(a)|$ bits. (Here and in the following, all logarithms are taken to base 2 unless otherwise indicated.) On the other hand, if the discarded information is structured, then the Kolmogorov complexity of the index of x in $S_y(a)$ will be significantly lower than the logarithm of the size of the sphere. The difference between these two code lengths can be used as an indicator of the amount of structural information that is discarded by the representation y. Vereshchagin and Vitányi [92] call this quantity the *randomness deficiency* of the source object x in the set $S_y(a)$, and they show that if y witnesses the rate-distortion function of x, then it *minimises* the randomness deficiency at rate $K(y)$; thus the rate-distortion function identifies those representations that account for as much structure as possible at the given rate.

To assess how much structure is being discarded at a given rate, consider a code for the source object x in which we first transmit the shortest possible program that constructs both a representation y and the distortion $d(x, y)$, followed by a literal, fixed-length index of x in the distortion sphere $S_y(a)$. Such a code has length function

$$K(y, d(x, y)) + L_y(x), \text{ where } \quad L_y(x) := \log |S_y(d(x, y))|. \tag{6.2}$$

If the rate is very low then the representation y models only very basic structure and the randomness deficiency in the distortion sphere around y is high. Borrowing terminology from statistics, we may say that y is a representation that "underfits" the data. In such cases we should find that $K(y, d(x, y)) + L_y(x) > K(x)$,

because the fixed-length code for the index of x within the distortion sphere is suboptimal in this case. But suppose that y is complex enough that it satisfies $K(y, d(x, y)) + L_y(x) \approx K(x)$. In [92], such representations are called *(algorithmic) sufficient statistics* for the data x. A sufficient statistic has close to zero randomness deficiency, which means that it represents all structure that can be detected in the data. However, sufficient statistics might contain not only structure, but noise as well. Such a representation would be overly complex, an example of overfitting. A *minimal* sufficient statistic balances between underfitting and overfitting. It is defined as the lowest complexity sufficient statistic, in other words the lowest complexity representation y that minimises the total code length. As such it can also be regarded as the "model" that should be selected on the basis of the Minimum Description Length (MDL) principle [4]. For a further discussion of this relationship see Section 6.6. To be able to relate the distortion-rate function to this code length we define the *code length function* $\lambda_x(r) = K(y, d(x, y)) + L_y(x)$ where y is the representation that minimises the distortion at rate r.[1]

6.1.2 Applications: Denoising and Lossy Compression

Representations that witness the rate-distortion function provide optimal separation between structure that can be expressed at the given rate and residual information that is perceived as noise. Therefore, these representations can be interpreted as denoised versions of the original. In denoising, the goal is of course to discard as much noise as possible, without losing any structure. Therefore the minimal sufficient statistic, which was described in the previous section, is the best candidate for applications of denoising.

While the minimal sufficient statistic is a denoised representation of the original signal, it is not necessarily given in a directly usable form. For instance, $\mathcal{Y}$ could consist of subsets of $\mathcal{X}$, but a *set* of source-words is not always acceptable as a denoising result. So in general one may need to apply some function $f : \mathcal{Y} \to \mathcal{X}$ to the sufficient statistic to construct a usable object. But if $\mathcal{X} = \mathcal{Y}$ and the distortion function is a metric, as in our case, then the representations are already in an acceptable format, so here we use the identity function for the transformation f.

In applications of lossy compression, one may be willing to accept a rate which is lower than the minimal sufficient statistic complexity, thereby losing some structural information. However, for a minimal sufficient statistic y, theory does tell us that it is not worthwhile to set the rate to a higher value than the complexity of y. The original object x is a random element of $S_y(d(x, y))$, and it cannot be distinguished from any other random $z \in S_y(d(x, y))$ using only

[1] This is superficially similar to the MDL function defined in [93], but it is *not* exactly the same since it involves optimisation of the distortion at a given rate rather than direct optimisation of the code length.

"simply described" properties. So we have no "simply described" test to discredit the hypothesis that x (or any such z) is the original object, given y and $d(x,y)$. If we increase the rate and find a model y' with $d(x,y') < d(x,y)$, then commonly the cardinality of $S_{y'}$ is smaller than that of S_y, such that some elements of S_y are not included in $S_{y'}$. These excluded elements, however, were perfectly good candidates of being the original object. That is, at rate higher than that of the minimal sufficient statistic, the resulting representation y' models irrelevant features that are specific to x, that is, noise and no structure, that exclude viable candidates for the original object: the representation starts to "overfit".

In lossy compression, as in denoising, the representations themselves may be unsuitable for presentation to the user. For example, when decompressing a lossily compressed image, in most applications a *set* of images would not be an acceptable result. So again a transformation from representations to objects of a usable form has to be specified. There are two obvious ways of doing this:

1. If a representation y witnesses the rate-distortion function for a source word $x \in \mathcal{X}$, then this means that x cannot be distinguished from any other object $x' \in S_y(d(x,y))$ at rate $K(y)$. Therefore we should not use a deterministic transformation, but rather report the uniform distribution on $S_y(d(x,y))$ as the lossily compressed version of x. This method has the advantage that it is applicable whether or not $\mathcal{X} = \mathcal{Y}$.

2. On the other hand, if $\mathcal{X} = \mathcal{Y}$ and the distortion function is a metric, then it makes sense to use the identity transformation again, although here the motivation is different. Suppose we select some $x' \in S_y(d(x,y))$ instead of y. Then the best upper bound we can give on the distortion is $d(x,x') \leq d(x,y) + d(y,x') = 2d(x,y)$ (by the triangle inequality and symmetry). On the other hand if we select y, then the distortion is exactly $d(x,y)$, which is only half of the upper bound we obtained for x'. Therefore it is more suitable if one adopts a worst-case approach. This method has as an additional advantage that the decoder does not need to *know* the distortion $d(x,y)$ which often cannot be computed from y without knowledge of x.

To illustrate the difference one may expect from these approaches, consider the situation where the rate is lower than the rate that would be required to specify a sufficient statistic. Then intuitively, all the noise in the source word x as well as some of the structure are lost by compressing it to a representation y. The second method immediately reports y, which contains a lot less noise than the source object x; thus x and y are qualitatively different, which may be undesirable. On the other hand, the compression result will be qualitatively different from x anyway, because the rate simply is too low to retain all structure. If one would apply the first approach, then a result x' would likely contain *more* noise than the original, because it contains less structure at the same level of distortion (meaning that $K(x') > K(x)$ while $d(x',y) = d(x,y)$).

If the rate is high enough to transmit a sufficient statistic, then the first approach seems preferable. We have nevertheless chosen to always report y directly in our analysis, which has the advantage that this way, all reported results are of the same type.

6.2 Computing Individual Object Rate-Distortion

The rate-distortion function for an object x with side information z and a distortion function d is found by simultaneously minimising two objective functions

$$\begin{aligned} g_1(y) &= K(y|z), \\ g_2(y) &= d(x,y), \\ g(y) &= \left\langle g_1(y), g_2(y) \right\rangle. \end{aligned} \tag{6.3}$$

We call the tuple $g(y)$ the *trade-off* of y. We impose a partial order on representations:

$$y \preccurlyeq y' \qquad \text{if and only if} \qquad g_1(y) \leq g_1(y') \text{ and } g_2(y) \leq g_2(y'). \tag{6.4}$$

Our goal is to find the set of representations that are minimal under $\preccurlyeq$.

Such an optimisation problem cannot be implemented because of the uncomputability of $K(\cdot)$. To make the idea practical, we need to approximate the conditional Kolmogorov complexity. As observed in [20], it follows directly from symmetry of information for Kolmogorov complexity (see [60, p.233]) that:

$$K(y|z) = K(zy) - K(z) + O(\log n), \tag{6.5}$$

where n is the length of zy. Ignoring the logarithmic term, this quantity can be approximated by replacing $K(\cdot)$ by $\tilde{K}(\cdot)$, the length of the compressed representation under a general purpose compression algorithm. The approximate conditional complexity then becomes

$$\begin{aligned} \tilde{K}(y|z) &:= \tilde{K}(zy) - \tilde{K}(z) \\ &\approx K(zy) - K(z) \quad = \quad K(y|z) + O(\log n). \end{aligned} \tag{6.6}$$

This may be a poor approximation: it is an upper bound that may be quite high even for objects that have conditional Kolmogorov complexity close to zero. Our results show evidence that some of the theoretical properties of the distortion-rate function nevertheless carry over to the practical setting; we also explain how some observations that are not predicted by theory are in fact related to the (unavoidable) inefficiencies of the used compressor.

6.2.1 Compressor (rate function)

We could have used any general-purpose compressor in (6.6), but we chose to implement our own for three reasons:

- It should be both fast and efficient. We can gain some advantage over other available compressors because there is no need to actually construct a code. It suffices to compute code *lengths*, which is much easier. As a secondary advantage, the code lengths we compute are not necessarily multiples of eight bits: we allow rational idealised code lengths, which may improve precision.
- It should not have any arbitrary restrictions or optimisations. Most general purpose compressors have limited window sizes or optimisations to improve compression of common file types; such features could make the results harder to interpret.

In our experiments we used a block sorting compression algorithm with a move-to-front scheme as described in [17]. In the encoding stage M2 we employ a simple statistical model and omit the actual encoding as it suffices to accumulate code lengths. The source code of our implementation (in C) is available from the authors upon request. The resulting algorithm is very similar to a number of common general purpose compressors, such the freely available bzip2 and zzip (see [82]), but it is simpler and faster for small inputs.

Of course, domain specific compressors might yield better compression for some object types (such as sound wave files), and therefore a better approximation of the Kolmogorov complexity. However, the compressor that we implemented is quite efficient for objects of many of the types that occur in practice; in particular it compressed the objects of our experiments (text and small images) quite well. We have tried to improve compression performance by applying standard image preprocessing algorithms to the images, but this turned out not to improve compression at all. Figure 6.1 lists the compressed size of an image of a mouse under various different compression and filtering regimes. Compared to other compressors, ours is quite efficient; this is probably because other compressors are optimised for larger files and because we avoid all overhead inherent in the encoding process. Most compressors have optimisations for text files which might explain why our compressor compares less favourably on the Oscar Wilde fragment.

6.2.2 Code Length Function

In Section 6.1.1 we introduced the code length function $\lambda_x(r)$. Its definition makes use of (6.2), for which we have not yet provided a computable alternative. We use the following approximation:

$$K(y, d(x, y)) \approx \tilde{K}(y) + L_D(d(x, y)|y), \tag{6.7}$$

Table 6.1 Compressed sizes of three objects that we experiment upon. See Figure 6.4(h) for the mouse, Figure 6.6 for the cross with added noise and Figure 6.10 for the corrupted Oscar Wilde fragment (the middle version). In the latter we give the compressed size *conditional* on a training text, like in the experiments. "A" is our own algorithm, described in Section 6.2.1. For a description of the filters see [75].

Compression	mouse	cross	Wilde	description
A	7995.11	3178.63	3234.45	Our compressor, described in §6.2.1
`zzip`	8128.00	3344.00	3184.00	An efficient block sorting compressor
`PPMd`	8232.00	2896.00	2744.00	High end statistical compressor
RLE → A	8341.68	3409.22	–	A with run length encoding filter
`bzip2`	9296.00	3912.00	3488.00	Widespread block sorting compressor
`gzip`	9944.00	4008.00	3016.00	LZ77 compressor
sub → A	10796.29	4024.26	–	A with Sub filter
paeth → A	13289.34	5672.70	–	A with Paeth filter
None	20480.00	4096.00	5864.00	Literal description

where L_D is yet another code which is necessary to specify the radius of the distortion sphere around y in which x can be found. It is possible that this distortion is uniquely determined by y, for example if $\mathcal{Y}$ is the set of all finite subsets of $\mathcal{X}$ and list decoding distortion is used, as described in [93]. If $d(x, y)$ is a function of y then $L_D(d(x, y)|y) = 0$. In other cases, the representations do not hold sufficient information to determine the distortion. This is typically the case when $\mathcal{X} = \mathcal{Y}$ as in the examples in this text. In that case we actually need to encode $d(x, y)$ separately. It turns out that the number of bits that are required to specify the distortion are negligible in proportion to the total three part code length. In the remainder of the chapter we use for L_D a universal code on the integers similar to the one described in [60]; it has code length $L_D(d) = \log(d + 1) + O(\log \log d)$.

We also need to calculate the size of the distortion sphere, which we therefore calculate for each of the distortion functions that we describe below.

6.2.3 Distortion Functions

We use three common distortion functions. All distortion functions used in this text are metrics, so they can only be used when $\mathcal{X} = \mathcal{Y}$.

Hamming distortion Hamming distortion is perhaps the simplest possible distortion function. It can be defined when $\mathcal{X} = \mathcal{Y} = \Sigma^n$, sequences of n symbols from a finite alphabet Σ. Let x and y be two objects of equal length n. The Hamming distortion $d(x, y)$ is equal to the number of symbols in x that do not match those in the corresponding positions in y.

A Hamming-distortion sphere $S_y(a)$ contains all objects of length n that can be constructed by replacing a symbols in y with different symbols from the alphabet Σ. Thus the size of the sphere is $\binom{n}{a}(|\Sigma|-1)^a$.

Euclidean distortion As before, let $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ be two objects of equal length, but the symbols now have a numerical interpretation. Euclidean distortion is $d(x,y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$: the distance between x and y when they are interpreted as vectors in an n-dimensional Euclidean space. Note that this definition of Euclidean distortion differs from the one in [92].

Our variety of Euclidean distortion requires that $\mathcal{X} = \mathcal{Y} = \mathbb{Z}^n$, the set of n-dimensional vectors of integers. The size of a Euclidean distortion sphere around some $y \in \mathcal{Y}$ of length n is hard to compute analytically. We use an upper bound that is reasonably tight and can be computed efficiently. One may want to skip this calculation on first reading.

First we define $d(v) := d(v, \mathbf{0}) = \sqrt{\sum_{i=1}^n v_i^2}$ and $S(n,a)$ as the set $\{v : |v| = n, d(v) = a\}$. We have $x \in S_y(a) \Leftrightarrow x - y \in S(n,a)$, so it suffices to bound the size of $S(n,a)$. We define:

$$p(\delta|n,a) := ce^{-\delta^2 n/2a^2} \text{ where } c = 1/\sum_{\delta \in \mathbb{Z}} e^{-\delta^2 n/2a^2}$$

$$P(v) := \prod_{i=1}^n p(v_i|n, d(v));$$

$p(\cdot|n,d(v))$ can be interpreted as a probability mass function on the individual entries of v (which in our application always lie between -255 and 255, so in practice we used a reduced range in the definition of the normalising constant c). Therefore $P(v)$ defines a valid probability mass function on outcomes v in $S(n,a)$. Thus,

$$1 > \sum_{v \in S(n,a)} P(v) = \sum_{v \in S(n,a)} c^n e^{-(\sum \delta_i^2) n/2a^2} = \sum_{v \in S(n,a)} c^n e^{-n/2} = c^n e^{-n/2} |S(n,a)|.$$

This yields a bound on the size of $S(n,a)$, which is reasonably tight unless the distortion a is very low. In that case, we can improve the bound by observing that v must have at least $z = n - d(v)^2$ zero entries. Let v' be a vector of length $n - z$ that is obtained by removing z zero entries from v. Every v in $S(n,a)$ can be constructed by inserting z zeroes into v', so we have $|S_y(a)| = |S(n,a)| \leq \binom{n}{z}|S(n-z,a)|$. The size of $S(n-z,a)$ can be bounded by using the method described before recursively.

Edit distortion The edit distortion of two strings x and y, of possibly different lengths, is the minimum number of symbols that have to be deleted from,

inserted into, or changed in x in order to obtain y (or vice versa) [54]. It is also known as *Levenshtein distortion*. It is a well-known measure that is often used in applications that require approximate string matching.

Edit distortion can be defined for spaces $\mathcal{X} = \mathcal{Y} = \Sigma^*$ for a finite alphabet Σ. We develop an upper bound on the size of the edit distortion sphere, again the calculation may be skipped on first reading.

We can identify any object in $S_y(a)$ by a program p that operates on y, and which is defined by a list of instructions to copy, replace or delete the next symbol from y, or to insert a new symbol. We interpret a deletion as a replacement with an empty symbol; so the replacement operations henceforth include deletions. Let $d(p)$ denote the number of insertions and replacements in p, in other words $d(p)$ is the length of the program minus the number of copies. Clearly for all $x \in S_y(a)$, there must be a p such that $p(y) = x$ and $d(p) = d(x, y) = a$. Therefore the size of $S_y(a)$ can be upper bounded by counting the number of programs with $d(p) = a$. Let n be the length of y. Any program that contains i insertions and that processes y completely, must be of length $n + i$. The i insertions, $a - i$ replacements and $n - a + i$ copies can be distributed over the program in $\binom{n+i}{i,a-i,n+a-i}$ different ways. For each insertion and replacement, the number of possibilities is equal to the alphabet size. Therefore,

$$|S_y(a)| \quad \leq \quad |\{p : d(p) = a\}| \quad \leq \quad |\Sigma|^a \sum_{i=\max\{0,a-n\}}^{a} \binom{n+i}{i, n-a+i, a-i}.$$

The sphere can be extremely large, so to facilitate calculation of the log of the sphere size, as is required in our application, it is convenient to relax the bound some more and replace every term in the sum by the largest one. Calculation reveals that the largest term has

$$i = \left\lfloor \frac{1}{4} \left(2(a-n) + 1 + \sqrt{4(n^2 + n + a^2 + a) + 1} \right) \right\rfloor.$$

6.2.4 Searching for the Rate-Distortion Function

The search problem that we propose to address has two properties that make it very hard. Firstly, the search space is enormous: at rate r there are 2^r candidate representations to consider, and for the kinds of objects that are typically subjected to lossy compression useful representations are often millions or billions of bits long. Secondly, we want to avoid making too many assumptions about the two objective functions, so that we can later freely change the compression algorithm and the distortion function. Under such circumstances the two most obvious search methods are not practical:

- An exhaustive search is infeasible for search spaces of such large size, unless more specific properties of the objective functions are used in the design

of the algorithm. To investigate how far we could take such an approach, we have implemented an exhaustive algorithm under the requirement that, given a prefix of a representation y, we can compute reasonable lower bounds on the values of both objective functions g_1 and g_2. This allows for relatively efficient enumeration of all representations of which the objective functions do not exceed specific maxima: it is never necessary to consider objects which have a prefix for which the lower bounds exceed the constraints, which allows for significant pruning. In this fashion we were able to find the rate-distortion function under Hamming distortion for objects of which the compressed size is about 25 bits or less within a few hours on a desk-top computer.

- A greedy search starts with a poor solution and iteratively makes modifications that constitute strict improvements. We found that this procedure tends to terminate quickly in some local optimum that is very bad globally.

Since the structure of the search landscape is at present poorly understood and we do not want to make any unjustifiable assumptions, we use a genetic search algorithm which performs well enough that interesting results can be obtained.

6.2.5 Genetic Algorithm

The used algorithm is an almost completely generic procedure to simultaneously optimise two separate objective functions for objects that are represented as byte sequences. To emphasise this we will consider the abstract objective function g wherever possible, rather than the more concrete rate and distortion functions.

A finite subset of $\mathcal{Y}$ is called a *pool*. The search algorithm initialises a pool $\mathcal{P}_0$ with the representation y that has $d(x, y) = 0$, which means $y = x$ in our setup, and possibly some "blank" representation that has minimal compressed code length but high distortion. The pool is then subjected to a process of selection through survival of the fittest. The *weakness* $w_{\mathcal{P}}(y)$ of an object $y \in \mathcal{P}$ is the number of elements of the pool that are smaller according to $\preccurlyeq$. The *(transitive) reduction* $\mathrm{trd}(\mathcal{P})$ of a pool $\mathcal{P}$ is the subset of all elements with zero weakness. The elements of the reduction of a pool $\mathcal{P}$ are called *models*.

The pool is iteratively updated by replacing elements with high weakness (the fitness function is specified below) by new ones, which are created through either *mutation* (random modifications of elements) or *crossover* ("genetic" recombination of pairs of other candidates). We write $\mathcal{P}_i$ to denote the pool after i iterations. When the algorithm terminates after n iterations it outputs the reduction of $\mathcal{P}_n$.

In the next sections we describe our choices for the important components of the algorithm: the mechanics of crossover and mutation, the fitness function and the selection function which specifies the probability that a candidate is removed from the pool. In the interest of reproducibility we faithfully describe all our

important design choices, even though some of them are somewhat arbitrary. A casual reader may want to skip such details and move on to Section 6.3.

Crossover

Crossover (also called recombination) is effected by the following algorithm. Given two objects x and y we first split them both in three parts: $x = x_1x_2x_3$ and $y = y_1y_2y_3$, such that the length of x_1 is chosen uniformly at random between 0 and the length of x and the length of x_2 is chosen from a geometric distribution with mean 5; the lengths of the y_i are proportional to the lengths of the x_i. We then construct a new object by concatenating $x_1y_2x_3$.

Mutation

The introduction of a mutation operation is necessary to ensure that the search space is connected, since the closure of the gene pool under crossover alone might not cover the entire search space. While we could have used any generic mutation function that meets this requirement, for reasons of efficiency we have decided to design a different mutation function for every objective function that we implemented. This is helpful because some distortion functions (here, the edit distortion) can compare objects of different sizes while others cannot: mutation is the means by which introduction of objects of different size to the pool can be brought about when desirable, or avoided when undesirable.

The mutation algorithm we use can make two kinds of change. With probability 1/4 we make a small random modification using an algorithm that depends on the distortion function. Below is a table of the distortion functions and a short description of the associated mutation algorithms:

Distortion	Mutation algorithm
Hamming	Sets a random byte to a uniformly random value
Euclidean	Adds an $\mathcal{N}[0; \sigma = 10]$ value to a random byte
Edit	A random byte is changed, inserted or deleted

With probability 3/4 we use the following mutation algorithm instead. It splits the object x into three parts $x = x_1x_2x_3$ where the length of x_1 is chosen uniformly at random between 0 and the length of x and the length of x_2 is chosen from a geometric distribution with mean 5. The mutation is effected by training a (simplified version of) a third order PPM model [24] on x_1 and then replacing x_2 with an equally long sequence that is sampled from the model. The advantage of this scheme is that every replacement for x_2 gets positive probability, but replacements which have low code length and distortion tend to be much more likely than under a uniform distribution.

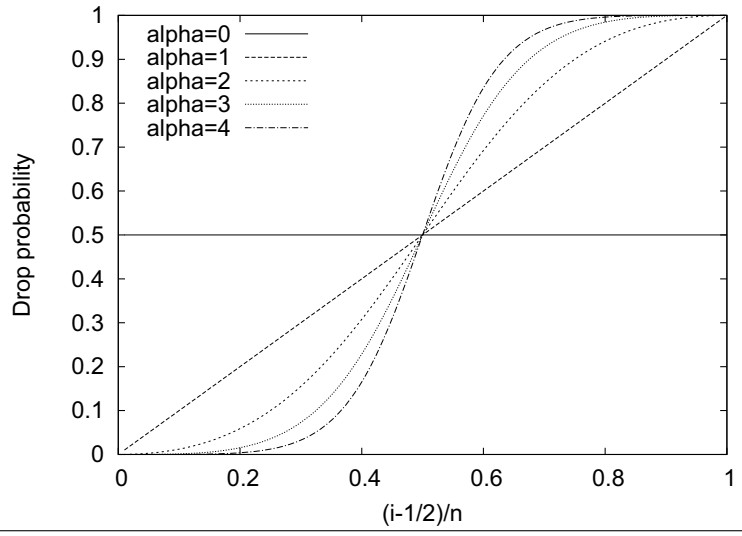
Fitness Function

In theory, the set of representations that witness the rate-distortion function does not change under monotonic transformation of either objective function g_1 or g_2. We have tried to maintain this property throughout the search algorithm including the fitness function, by never using either objective function directly but only the ordering relation $\preccurlyeq$. Under such a regime, a very natural definition of fitness is minus the weakness of the objects with respect to pool $\mathcal{P}$.

It has been an interesting mini-puzzle to come up with an efficient algorithm to compute the weakness of all objects in the pool efficiently. Our solution is the following very simple algorithm, which has an $O(n \log n)$ average case running time. It first sorts all elements of the pool by their value under g_1 and then inserts them in order into a binary search tree in which the elements are ordered by their value under g_2. As an object is inserted into the tree, we can efficiently count how many elements with lower values for g_2 the tree already contained. These elements are precisely the objects that have both lower values on g_1 (otherwise they would not appear in the tree yet) and on g_2; as such their number is the desired weakness.

Selection Function

Figure 6.2 Drop probability



A pool $\mathcal{P}$ induces a *tradeoff profile* $p(\mathcal{P}) := \{g(y) : y' \preccurlyeq y \text{ for some } y' \text{ in } \mathcal{P}\}$. It is not hard to see that we have $p(\mathcal{P}) = p(\mathrm{trd}(\mathcal{P}))$ and $p(\mathcal{P}) \subseteq p(\mathcal{P} \cup \mathcal{P}')$ for all $\mathcal{P}' \subseteq \mathcal{Y}$. Therefore monotonic improvement of the pool under modification is ensured as long as candidates with weakness 0 are never dropped.

We drop other candidates with positive probability as follows. Let $y_1, \ldots, y_n$ be the elements of $\mathcal{P}$ with nonzero weakness, ordered such that for $1 \leq i < j \leq n$

we have $w_{\mathcal{P}}(y_i) < w_{\mathcal{P}}(y_j)$ or $g_1(y_i) < g_1(y_j)$ if y_i and y_j have the same weakness. We drop candidate y_i from the pool with probability $1/(1 + (\frac{n}{i-1/2} - 1)^{\alpha})$, which is a modified sigmoid function where $\alpha \in (1, \infty)$ specifies the sharpness of the transition from probability zero to one. This function is plotted for different values of α in Figure 6.2. We used $\alpha = 4$ in our experiments.

6.3 Experiments

We have subjected four objects to our program. The following considerations have influenced our choice of objects:

- Objects should not be too complex, allowing our program to find a good approximation of the distortion-rate curve. We found that the running time of the program seems to depend mostly on the complexity of the input object; a compressed size of 20,000 bits seemed to be about the maximum our program could handle within a reasonable amount of time, requiring a running time of the order of weeks on a desk-top computer.

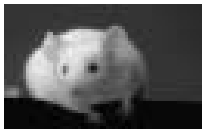
- To check that our method really is general, objects should be quite different from each other: they should come from different object domains, for which different distortion functions are appropriate, and they should contain structure at different levels of complexity.

- Objects should contain primary structure and regularities that are distinguishable and compressible by a block sorting compressor such as the one we use. Otherwise, we may no longer hope that the compressor implements a reasonable approximation of the Kolmogorov complexity. For instance, we would not expect our program to do well on a sequence of digits from the binary expansion of the number π.

With this in mind, we have selected the objects listed in Figure 6.3.

In each experiment, as time progressed the program found less and less improvements per iteration, but the pool never stabilized completely. Therefore we interrupted each experiment when (a) after at least one night of computation, the pool did not improve a lot, and (b) for all intuitively good models $y \in \mathcal{Y}$ that we could conceive of a priori, the algorithm had found an y' in the pool with $y' \preccurlyeq y$ according to (6.4). For example, in each denoising experiment, this test included the original, noiseless object. In the experiment on the mouse without added noise, we also included the images that can be obtained by reducing the number of grey levels in the original with an image manipulation program. Finally for the greyscale images we included a number of objects that can be obtained by subjecting the original object to JPEG2000 compression at various quality levels.

Figure 6.3 The four objects that are subjected to rate-distortion analysis.



A picture of a mouse of 64 × 40 pixels. The picture is analysed with respect to Euclidean distortion.

A noisy monochrome image of 64 × 64 pixels that depicts a cross. 377 pixels have been inverted. Hamming distortion is used.

The same picture of a mouse, but now zero mean Gaussian noise with $\sigma = 8$ has been added to each pixel. Euclidean distortion is used; the distortion to the original mouse is 391.1.

Beauty, real beauty, ends2wheresan intellectual expressoon begins. IntellHct isg in itself a mMde ofSexggeration, an\ destroys theLharmony of n face. [...]

(See Figure 6.10)

A corrupted quotation from Chapter 1 of *The Picture of Dorian Gray*, by Oscar Wilde. The 733 byte long fragment was created by performing 68 random insertions, deletions and replacements of characters in the original text. Edit distortion is used. The rest of chapters one and two of the novel are given to the program as side information.

The first experiment illustrates how algorithmic rate-distortion theory may be applied to lossy compression problems, and it illustrates how for a given rate, some features of the image are preserved while others can no longer be retained. We compare the performance of our method to the performance of JPEG and JPEG2000 at various quality levels. Standard JPEG images were encoded using the ImageMagick version 6.2.2; profile information was stripped. JPEG2000 images were encoded to jpc format with three quality levels using NetPBM version 10.33.0; all other options are default. For more information about these software packages refer to [83].

The other three experiments are concerned with denoising. Any model that is output by the program can be interpreted as a denoised version of the input object. We measure the denoising success of a model y as $d(x', y)$, where x' is the original version of the input object x, before noise was added. We also compare the denoising results to those of other denoising algorithms:

1. BayesShrink denoising [18]. BayesShrink is a popular wavelet-based denoising method that is considered to work well for images.

2. Blurring (convolution with a Gaussian kernel). Blurring works like a low-pass filter, eliminating high frequency information such as noise. Unfortunately other high frequency features of the image, such as sharp contours, are also discarded.

3. Naive denoising. We applied a naive denoising algorithm to the noisy cross, in which each pixel was inverted if five or more out of the eight neighbouring

pixels were of different colour.

4. Denoising based on JPEG2000. Here we subjected the noisy input image to JPEG2000 compression at different quality levels. We then selected the result for which the distortion to the original image was lowest.

6.3.1 Names of Objects

To facilitate description and discussion of the experiments we will adopt the following naming convention. Objects related to the experiments with the mouse, the noisy cross, the noisy mouse and the Wilde fragment, are denoted by the symbols $\mathbb{M}$, $\mathbb{C}$, $\mathbb{N}$ and $\mathbb{W}$ respectively. A number of important objects in each experiment are identified by a subscript as follows. For $\mathbb{O} \in \{\mathbb{M}, \mathbb{C}, \mathbb{N}, \mathbb{W}\}$, the input object, for which the rate-distortion function is approximated by the program, is called $\mathbb{O}_{\text{IN}}$. In the denoising experiments, the input object is always constructed by adding noise to an original object. The original objects and the noise are called $\mathbb{O}_{\text{ORIG}}$ and $\mathbb{O}_{\text{NOISE}}$ respectively. If Hamming distortion is used, addition is carried out modulo 2, so that the input object is in effect a pixelwise exclusive OR of the original and the noise. In particular, $\mathbb{C}_{\text{IN}}$ equals $\mathbb{C}_{\text{ORIG}}$ XOR $\mathbb{C}_{\text{NOISE}}$. The program outputs the reduction of the gene pool, which is the set of considered models. Two important models are also given special names: the model within the gene pool that minimises the distortion to $\mathbb{O}_{\text{ORIG}}$ constitutes the best denoising of the input object and is therefore called $\mathbb{O}_{\text{BEST}}$, and the minimal sufficient statistic as described in Section 6.1.1 is called $\mathbb{O}_{\text{MSS}}$. Finally, in the denoising experiments we also give names to the results of the alternative denoising algorithms. Namely, $\mathbb{C}_{\text{NAIVE}}$ is the result of the naive denoising algorithm applied to the noisy cross, $\mathbb{N}_{\text{BLUR}}$ is the convolution of $\mathbb{N}$ with a Gaussian kernel with $\sigma = 0.458$, $\mathbb{N}_{\text{BS}}$ is the denoising result of the BayesShrink algorithm, and $\mathbb{N}_{\text{JPEG2000}}$ is the image produced by subjecting $\mathbb{N}$ to JPEG2000 compression at the quality level for which the distortion to $\mathbb{N}_{\text{ORIG}}$ is minimised.

6.4 Results and Discussion

After running for some time on each input object, our program outputs the reduction of a pool $\mathcal{P}$, which is interpreted as a set of models. For each experiment, we report a number of different properties of these sets. Since we are interested in the rate-distortion properties of the input object $x = \mathbb{O}_{\text{IN}}$, we plot the approximation of the distortion-rate function of each input object: $d_x(r) = \min\{d(x, y) : y \in \mathcal{Y}, K(y) \leq r\} \approx \min\{d(x, y) : y \in \text{trd}(\mathcal{P}), \tilde{K}(y) \leq r\}$. Such approximations of the distortion-rate function are provided for all four experiments. For the greyscale images we also plot the distortion-rate approximation that is achieved by JPEG2000 (and in Figure 6.5 also ordinary JPEG) at different quality levels. Here, the rate is the code length achieved by JPEG(2000), and the

distortion is the Euclidean distortion to $\mathbb{O}_{\text{IN}}$. We also plot the code length function as described in Section 6.1.1. Minimal sufficient statistics can be identified by locating the minimum of this graph.

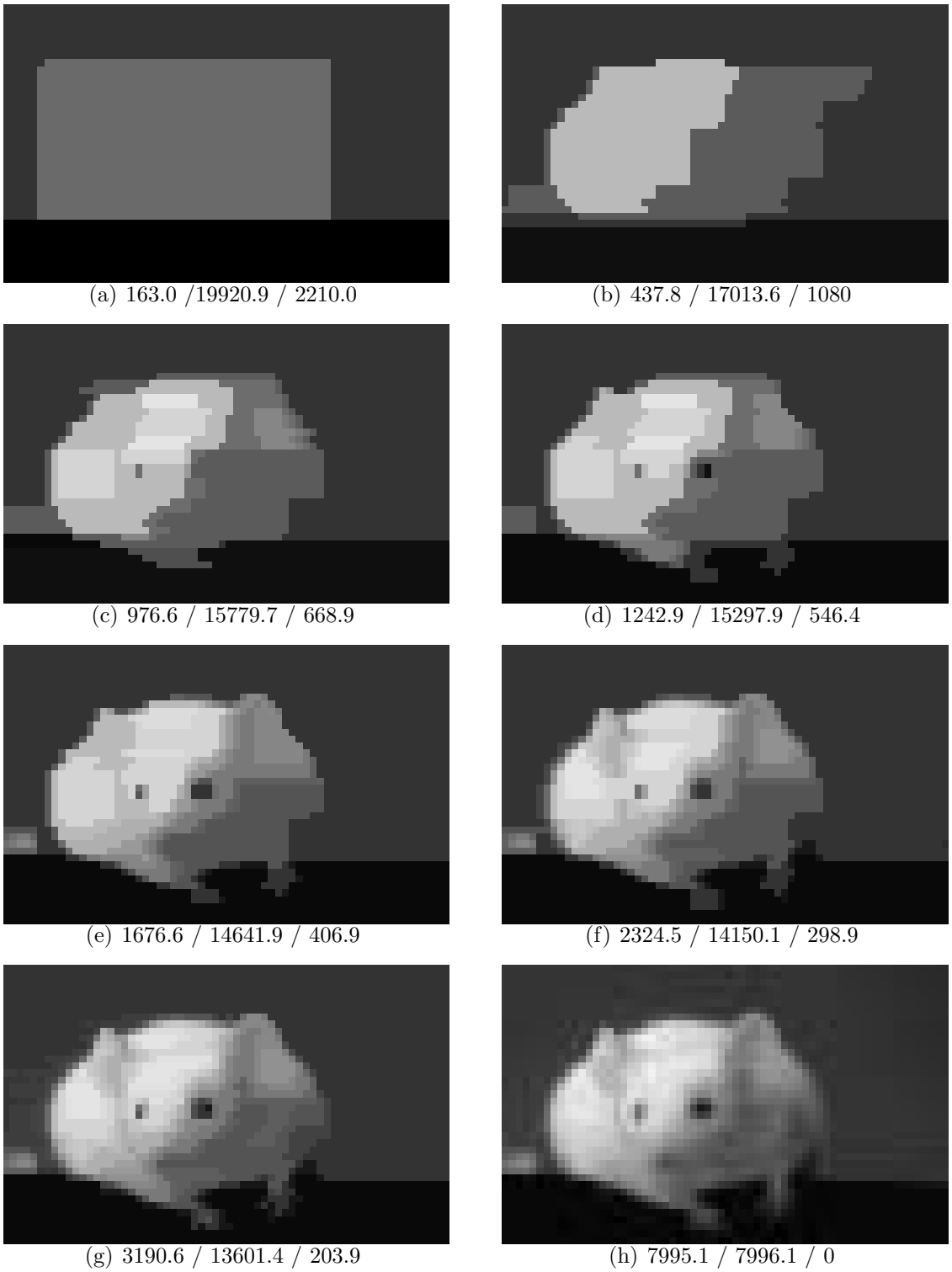
6.4.1 Lossy Compression

Experiment 1: $\mathbb{M}$ouse (Euclidean distortion)

Our first experiment involved the lossy compression of $\mathbb{M}$, a greyscale image of a mouse. A number of elements of the gene pool are shown in Figure 6.4. The pictures show how at low rates, the models capture the most important global structure of the image; at higher rates more subtle properties of the image can be represented. Image (a) shows a rough rendering of the distribution of bright and dark areas in $\mathbb{M}_{\text{IN}}$. These shapes are rectangular, which is probably an artifact of the compression algorithm we used: it is better able to compress images with rectangular structure than with oval structure. There is no real reason why an oval should be in any way more complex than a rectangle, but most general purpose data compression software is similarly biased. In (b), the rate is high enough that the oval shape of the mouse can be accommodated, and two areas of different overall brightness are identified. After the number of grey shades has been increased a little further in (c), the first hint of the mouse's eyes becomes visible. The eyes are improved and the mouse is given paws in (d). At higher rates, the image becomes more and more refined, but the improvements are subtle and seem of a less qualitative nature.

The code length function (Figure 6.5) shows that the only sufficient statistic in the set of models is $\mathbb{M}_{\text{IN}}$ itself, indicating that the image hardly contains any noise. It also shows the rates that correspond to the models that are shown in Figure 6.4. By comparing these figures it can be clearly seen that the image quality only deteriorates significantly if more than half of the information in $\mathbb{M}_{\text{IN}}$ is discarded. Note that this is not a statement about the compression ratio, where the lossily compressed size is related to the size of the *uncompressed* object rather than its complexity. For example, $\mathbb{M}_{\text{IN}}$ has an uncompressed size of $64 \cdot 40 \cdot 8 = 20480$ bits, and the representation in Figure 6.4(g) has a compressed size of 3190.6 bits. This representation therefore constitutes compression by a factor of $20480/3190.6 = 6.42$, which is substantial for an image of such small size. At the same time the amount of *information* is reduced by a factor of $7995.0/3190.6 = 2.51$.

Figure 6.4 Lossy image compression results for the mouse (h). The numbers below each image denote its compressed size $\tilde{K}(\cdot)$, total code length $\tilde{K}(\cdot)+L_{(.)}(\mathbb{M}_{\mathrm{IN}})$ and Euclidean distortion $d(\cdot, \mathbb{M}_{\mathrm{IN}})$, respectively.



(a) 163.0 /19920.9 / 2210.0

(b) 437.8 / 17013.6 / 1080

(c) 976.6 / 15779.7 / 668.9

(d) 1242.9 / 15297.9 / 546.4

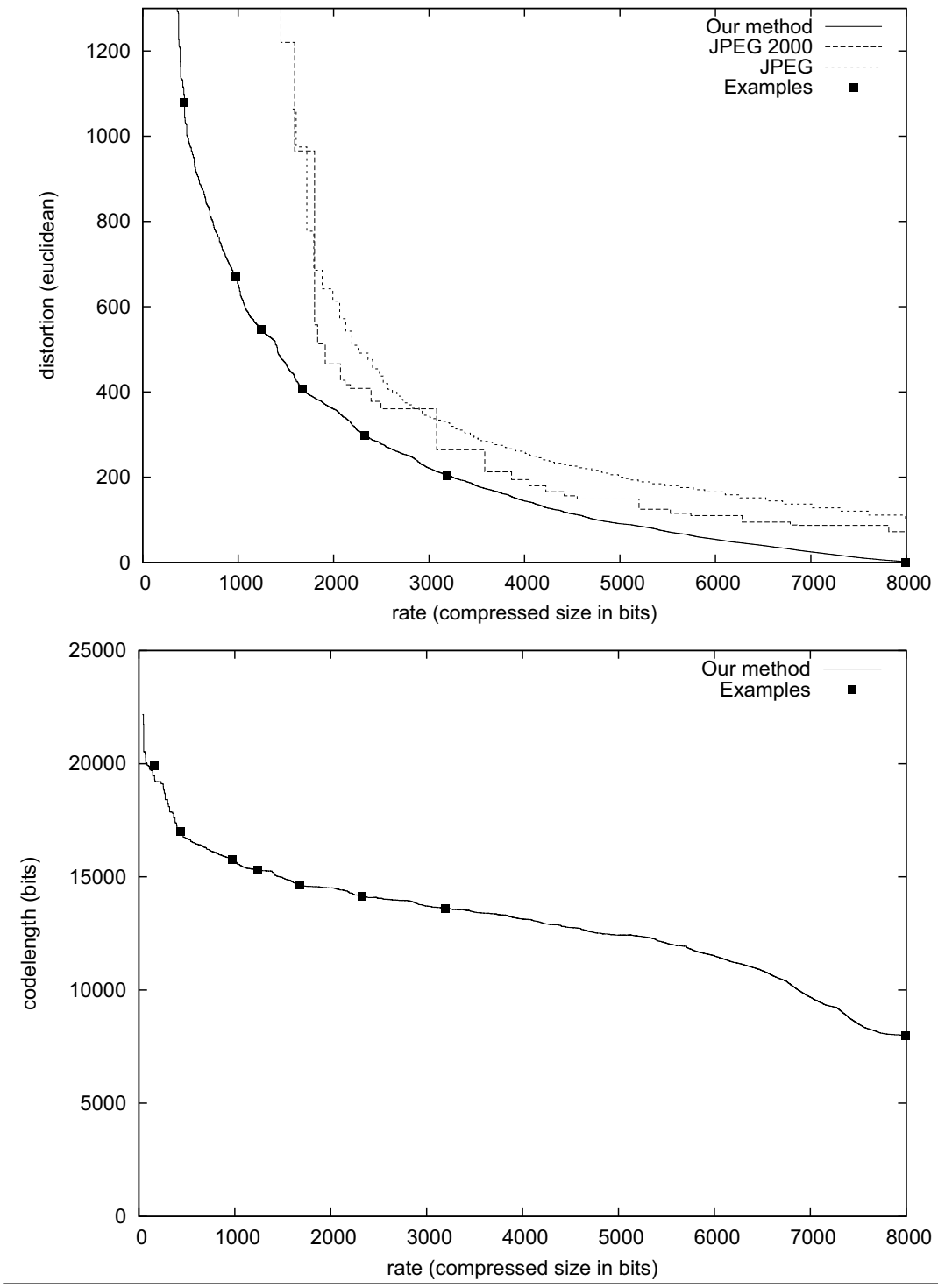
(e) 1676.6 / 14641.9 / 406.9

(f) 2324.5 / 14150.1 / 298.9

(g) 3190.6 / 13601.4 / 203.9

(h) 7995.1 / 7996.1 / 0

Figure 6.5 Approximate distortion-rate and code length functions for the mouse.



6.4.2 Denoising

For each denoising experiment, we report a number of important objects, a graph that shows the approximate distortion-rate function and a graph that shows the approximate code length function. In the distortion-rate graph we plot not only the distortion to $\mathbb{O}_{\text{IN}}$ but also the distortion to $\mathbb{O}_{\text{ORIG}}$, to visualise the denoising success at each rate.

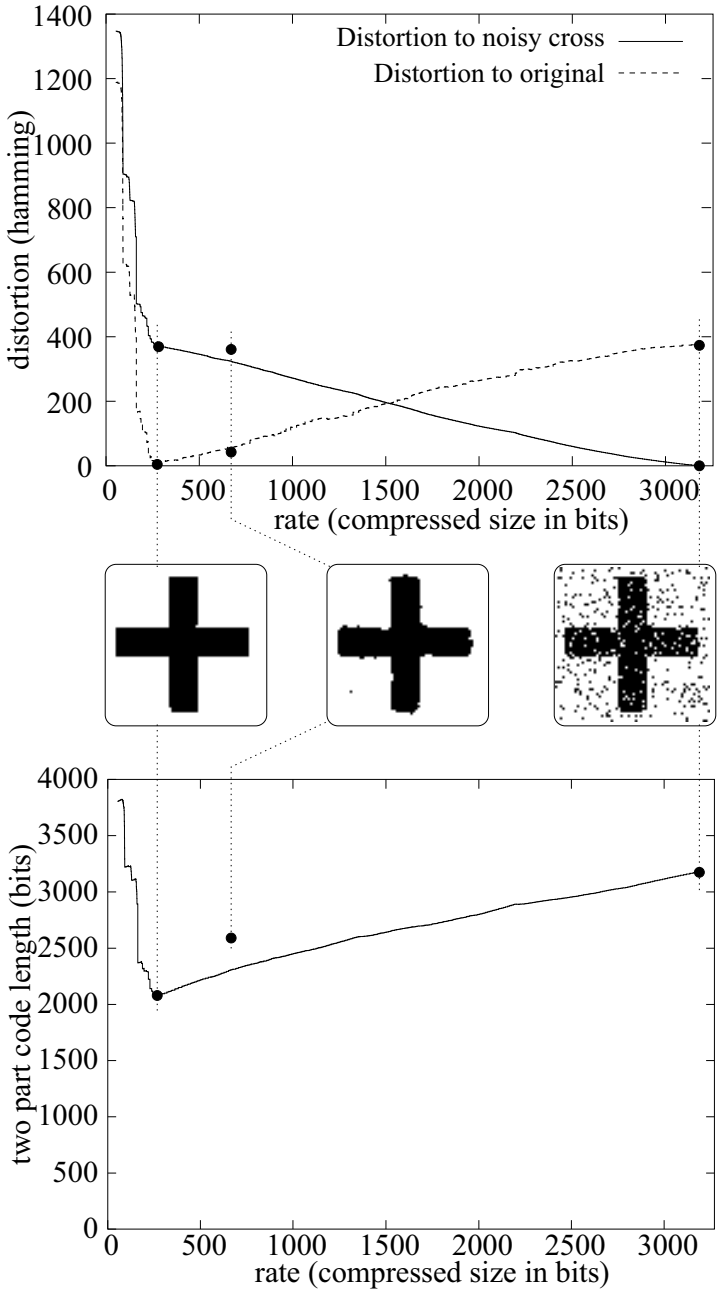
In interpreting these results, it is important to realise that only the reported minimal sufficient statistic and the results of the BayesShrink and naive denoising methods can be obtained without knowledge of the original object – the other objects $\mathbb{O}_{\text{BEST}}$, $\mathbb{O}_{\text{JPEG2000}}$ and $\mathbb{O}_{\text{BLUR}}$ require selecting between a number of alternatives in order to optimise the distortion to $\mathbb{O}_{\text{ORIG}}$, which can only be done in a controlled experiment. Their performance may be better than what can be achieved in practical situations where $\mathbb{O}_{\text{ORIG}}$ is not known.

Experiment 2: Noisy $\mathbb{C}$ross (Hamming distortion)

In the first denoising experiment we approximated the distortion-rate function of a monochrome cross $\mathbb{C}_{\text{ORIG}}$ of very low complexity, to which artificial noise was added to obtain $\mathbb{C}_{\text{IN}}$ (the rightmost image in Figure 6.6); the distortion to the noiseless cross is displayed in the same graph. The best denoising $\mathbb{C}_{\text{BEST}}$ (leftmost image) has a distortion of only 3 to the original $\mathbb{C}_{\text{ORIG}}$, which shows that the distortion-rate function indeed separates structure and noise extremely well in this example. The bottom graph shows the code length function for the noisy cross; the minimum on this graph is the minimal sufficient statistic $\mathbb{C}_{\text{MSS}}$. In this low complexity example, we have $\mathbb{C}_{\text{MSS}} = \mathbb{C}_{\text{BEST}}$, so the best denoising is not only very good in this simple example, but it can also be identified.

We did not subject $\mathbb{C}_{\text{IN}}$ to BayesShrink or blurring because those methods are not suitable for monochrome images. Therefore we used the extremely simple, "naive" denoising method that is described in Section 6.3 on this specific image instead. The middle image shows the result $\mathbb{C}_{\text{NAIVE}}$; while it does remove most of the noise, 40 errors remain, a lot more than the number of errors incurred by the minimal sufficient statistic. All errors except one are close to the contours of the cross. This illustrates how the naive algorithm is limited by its property that it takes only the local neighbourhood of each pixel into account, it cannot represent larger structures such as straight lines.

Figure 6.6 Denoising a noisy cross. Highlighted objects, from left to right: $\mathbb{C}_{\text{BEST}}$, $\mathbb{C}_{\text{NAIVE}}$ and $\mathbb{C}_{\text{IN}}$. Exact values are in the bottom table.



	$\mathbb{C}_{\text{BEST}}$=$\mathbb{C}_{\text{MSS}}$	$\mathbb{C}_{\text{NAIVE}}$	$\mathbb{C}_{\text{IN}}$
$\tilde{K}(\cdot)$	260.4	669.2	3178.6
$\tilde{K}(\cdot) + L_{(\cdot)}(\mathbb{C}_{\text{IN}})$	2081.9	2533.3	3179.6
$d(\cdot, \mathbb{C}_{\text{IN}})$	376	389	0
$d(\cdot, \mathbb{C}_{\text{ORIG}})$	3	40	377

Experiment 3: $\mathbb{N}$oisy mouse (Euclidean distortion)

The noisy mouse poses a significantly harder denoising problem, where the total complexity of the input $\mathbb{N}_{\text{IN}}$ is more than five times that of the noisy cross. Figure 6.7 shows the denoising results for various denoising methods, and Figure 6.8 shows the rate-distortion curve and, as for the noisy cross, the distortion to the original object $\mathbb{N}_{\text{ORIG}}$.

Figure 6.7(a) shows the input object $\mathbb{N}_{\text{IN}}$; it was constructed by adding noise (centre image) to the original noiseless image $\mathbb{N}_{\text{ORIG}}$ (top-right). We display three different denoising results. Image (h) shows $\mathbb{N}_{\text{BEST}}$, the best denoised object from the gene pool. Visually it appears to resemble $\mathbb{N}_{\text{ORIG}}$ quite well, but there might be structure in $\mathbb{N}_{\text{ORIG}}$ that was lost in the denoising process. Because human perception is perhaps the most sensitive detector of structure in image data, we show the difference between $\mathbb{N}_{\text{BEST}}$ and $\mathbb{N}_{\text{ORIG}}$ in (i). We would expect any significant structure in the original image that is lost in the denoising process, as well as structure that is not present in the original image, but is somehow introduced as an artifact of the denoising procedure, to become visible in this residual. In the case of $\mathbb{N}_{\text{BEST}}$ we cannot make out any particular features.

The minimal sufficient statistic, image (d), also appears to be a reasonably successful denoising, albeit clearly of lower complexity than the best one. In the residual, darker and lighter patches are definitely discernible. Apparently $\mathbb{N}_{\text{IN}}$ does contain some structure beyond what is captured by $\mathbb{N}_{\text{MSS}}$, but this cannot be exploited by the compression algorithm. We think that the fact that the minimal sufficient statistic is of lower complexity than the best possible denoising result should therefore again be attributed to inefficiencies of the compressor.

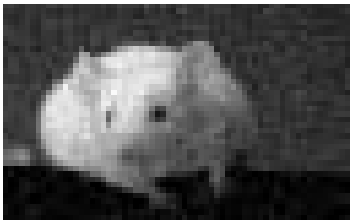
For comparison, we have also denoised $\mathbb{N}_{\text{IN}}$ using the alternative denoising method BayesShrink and the methods based on blurring and JPEG2000 as described in Section 6.3. We found that BayesShrink does not work well for images of such small size: the distortion between $\mathbb{N}_{\text{BS}}$ and $\mathbb{N}_{\text{IN}}$ is only 72.9, which means that the input image is hardly affected at all. Also, $\mathbb{N}_{\text{BS}}$ has a distortion of 383.8 to $\mathbb{N}_{\text{ORIG}}$, which is hardly less than the distortion of 392.1 achieved by $\mathbb{N}_{\text{IN}}$ itself.

Blurring-based denoising yields much better results: $\mathbb{N}_{\text{BLUR}}$ (j) is the result after optimisation of the size of the Gaussian kernel. Its distortion to $\mathbb{N}_{\text{ORIG}}$ lies in-between the distortions achieved by $\mathbb{N}_{\text{MSS}}$ and $\mathbb{N}_{\text{BEST}}$, but it is different from those objects in two important respects. Firstly, $\mathbb{N}_{\text{BLUR}}$ remains much closer to $\mathbb{N}_{\text{IN}}$, at a distortion of 260.4 instead of more than 470, and secondly, $\mathbb{N}_{\text{BLUR}}$ is much less compressible by $\hat{K}$. (To obtain the reported size of 14117 bits we had to switch on the averaging filter, as described in Section 6.2.1.) These observations are at present not well understood. Image (k) shows that the contours of the mouse are somewhat distorted in $\mathbb{N}_{\text{BLUR}}$; this can be explained by the fact that contours contain high frequency information which is discarded by the blurring operation as we remarked in Section 6.3.

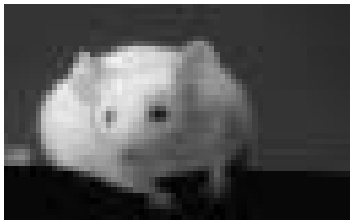
The last denoising method we compared our results to is the one based on the

JPEG2000 algorithm. Its performance is clearly inferior to our method visually as well as in terms of rate and distortion. The result seems to have undergone a smoothing process similar to blurring which introduces similar artifacts in the background noise, as is clearly visible in the residual image. As before, the comparison may be somewhat unfair because JPEG2000 was not designed for the purpose of denoising, might optimise a different distortion measure and is much faster.

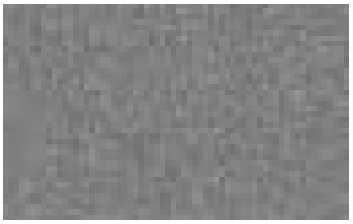
Figure 6.7 Denoising results for the noisy mouse (a). The numbers below each image denote its compressed size $\tilde{K}(\cdot)$, total code length $\tilde{K}(\cdot) + L_{(\cdot)}(\mathbb{N}_{\text{IN}})$, distortion to $\mathbb{N}_{\text{IN}}$ and distortion to $\mathbb{N}_{\text{ORIG}}$, respectively.



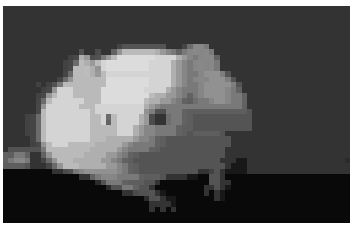
(a) $\mathbb{N}_{\text{IN}}$
16699.7 / 16700.7 / 0 / 392.1



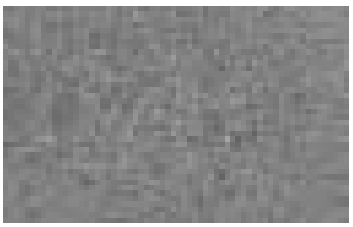
(b) $\mathbb{N}_{\text{ORIG}}(=\mathbb{M}_{\text{IN}})$
7995.1 / 26247.8 / 392.1 / 0



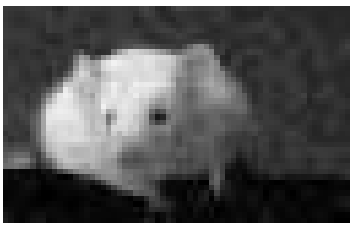
(c) $\mathbb{N}_{\text{NOISE}}$=$\mathbb{N}_{\text{IN}}$-$\mathbb{N}_{\text{ORIG}}$



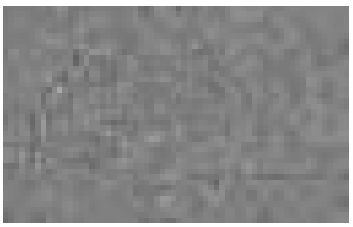
(d) $\mathbb{N}_{\text{MSS}}$
1969.8 / 15504.9 / 474.7 / 337.0



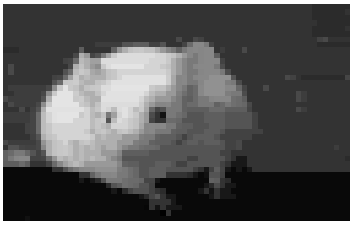
(e) $\mathbb{N}_{\text{MSS}} - \mathbb{N}_{\text{ORIG}}$



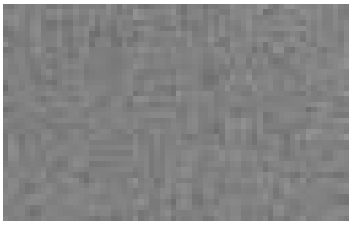
(f) $\mathbb{N}_{\text{JPEG2000}}$
3104 / 16395.4 / 444.4 / 379.8



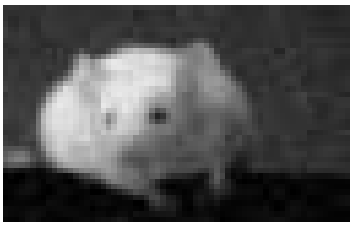
(g) $\mathbb{N}_{\text{JPEG2000}} - \mathbb{N}_{\text{ORIG}}$



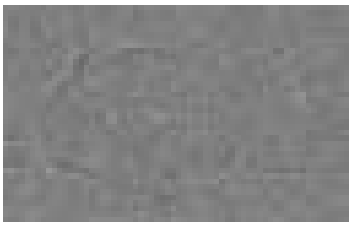
(h) $\mathbb{N}_{\text{BEST}}$
3354.5 / 15952.8 / 368.4 / 272.2



(i) $\mathbb{N}_{\text{BEST}} - \mathbb{N}_{\text{ORIG}}$

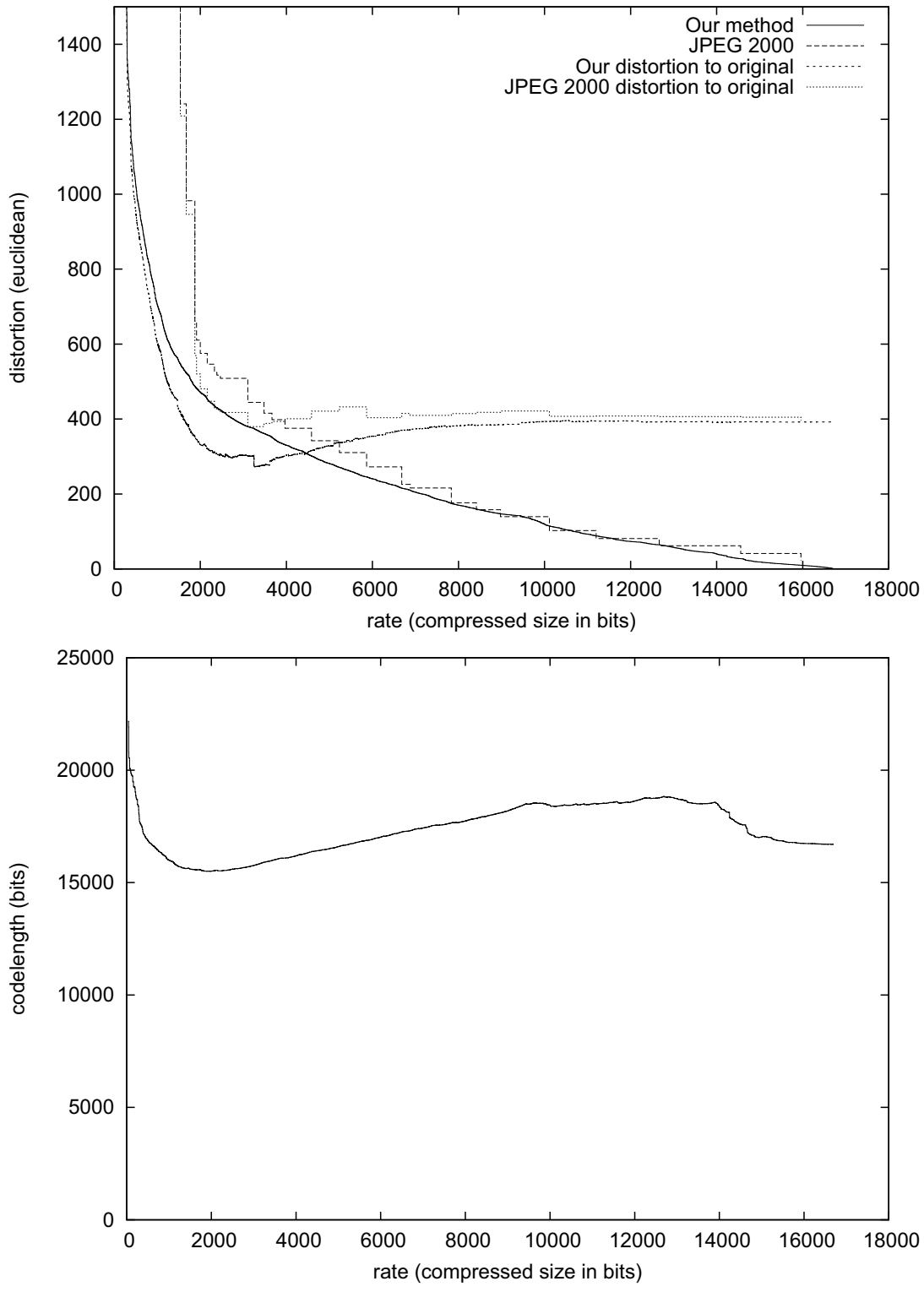


(j) $\mathbb{N}_{\text{BLUR}}$ ($\sigma = 0.458$)
14117.0 / 25732.4 / 260.4 / 291.2



(k) $\mathbb{N}_{\text{BLUR}} - \mathbb{N}_{\text{ORIG}}$

Figure 6.8 Approximate distortion-rate and code length functions for the noisy mouse.



Experiment 4: Oscar 𝕎ilde fragment (edit distortion)

The fourth experiment, in which we analyse $\mathbb{W}_{\text{IN}}$, a corrupted quotation from Oscar Wilde, shows that our method is a general approach to denoising that does not require many domain specific assumptions. $\mathbb{W}_{\text{ORIG}}$, $\mathbb{W}_{\text{IN}}$ and $\mathbb{W}_{\text{MSS}}$ are depicted in Figure 6.10, the distortion-rate approximation, the distortion to $\mathbb{W}_{\text{ORIG}}$ and the three part code length function are shown in Figure 6.11. We have trained the compression algorithm by supplying it with the rest of Chapters 1 and 2 of the same novel as side information, to make it more efficient at compressing fragments of English text. We make the following observations regarding the minimal sufficient statistic:

- In this experiment, $\mathbb{W}_{\text{MSS}} = \mathbb{W}_{\text{BEST}}$ so the minimal sufficient statistic separates structure from noise extremely well here.
- The distortion is reduced from 68 errors to only 46 errors. 26 errors are corrected (▲), 4 are introduced (▼), 20 are unchanged (•) and 22 are changed incorrectly (★).
- The errors that are newly introduced (▼) and the incorrect changes (★) typically simplify the fragment a lot, so that the compressed size may be expected to drop significantly. Not surprisingly therefore, many of the symbols marked ▼ or ★ are deletions, or modifications that create a word which is different from the original, but still correct English. The following table lists examples of the last category:

Line	$\mathbb{W}_{\text{ORIG}}$	$\mathbb{W}_{\text{IN}}$	$\mathbb{W}_{\text{MSS}}$
3	`or`	`Nor`	`of`
4	`the`	`Ghe`	`he`
4	`any`	`anL`	`an`
4	`learned`	`JeaFned`	`yearned`
5	`course`	`corze`	`core`
5	`then`	`ehen`	`when`
8	`he`	`fhe`	`the`

  Since it would be hard for *any* general-purpose mechanical method (that does not incorporate a sophisticated English language model) to determine that these changes are incorrect, we should not be surprised to find a number of errors of this kind.

Side Information

Figure 6.9 shows that the compression performance is significantly improved if we provide side information to the compression algorithm, and the improvement is typically larger if (1) the amount of side information is larger, or (2) if the

compressed object is more similar to the side information. Thus, by giving side information, correct English prose is recognised as "structure" sooner and a better separation between structure and noise is to be expected. The table also shows that if the compressed object is in some way different from the side information, then adding more side information will at some point become counter-productive, presumably because the compression algorithm will then use the side information to build up false expectations about the object to be compressed, which can be costly.

While denoising performance would probably improve if the amount of side information was increased further, it was infeasible to do so in this implementation. Recall from Section 6.2 that he conditional Kolmogorov complexity $K(y|z)$ is approximated by $\tilde{K}(y|z) = \tilde{K}(zy) - \tilde{K}(z)$. The time required to compute this is dominated by the length of z if the amount of side information is much larger than the size of the object to be compressed. This can be remedied by using a compression algorithm that processes its input sequentially, because the state of such an algorithm can be cached after processing the side information z; computing $\tilde{K}(zy)$ would then be a simple matter of recalling the state that was reached after processing z and then processing y starting from that state. Many compression algorithms, among which Lempel-Ziv compressors and most statistical compressors, have this property; our approach could thus be made to work with large quantities of side information by switching to a sequential compressor but we have not done this.

Figure 6.9 Compressed size of models for different amounts of side information. $\mathbb{W}_{\text{ORIG}}$ is never included in the side information. We do not let $\mathbb{W}_{\text{MSS}}$ vary with side information but keep it fixed at the object reported in Figure 6.10(c).

Side information z	$\tilde{K}(\mathbb{W}_{\text{ORIG}}\|z)$	$\tilde{K}(\mathbb{W}_{\text{MSS}}\|z)$	$\tilde{K}(\mathbb{W}_{\text{IN}}\|z)$
None	3344.1	3333.7	3834.8
Chapters 1,2 (57 kB)	1745.7	1901.9	3234.5
Whole novel (421 kB)	1513.6	1876.5	3365.9

Figure 6.10 A fragment of *The Picture of Dorian Gray*, by Oscar Wilde.

Beauty, real beauty, ends where an intellectual expression begins. Intellect is in itself a mode of exaggeration, and destroys the harmony of any face. The moment one sits down to think, one becomes all nose, or all forehead, or something horrid. Look at the successful men in any of the learned professions. How perfectly hideous they are! Except, of course, in the Church. But then in the Church they don't think. A bishop keeps on saying at the age of eighty what he was told to say when he was a boy of eighteen, and as a natural consequence he always looks absolutely delightful. Your mysterious young friend, whose name you have never told me, but whose picture really fascinates me, never thinks. I feel quite sure of that.

(a) $\mathbb{W}_{\text{ORIG}}$, the original text

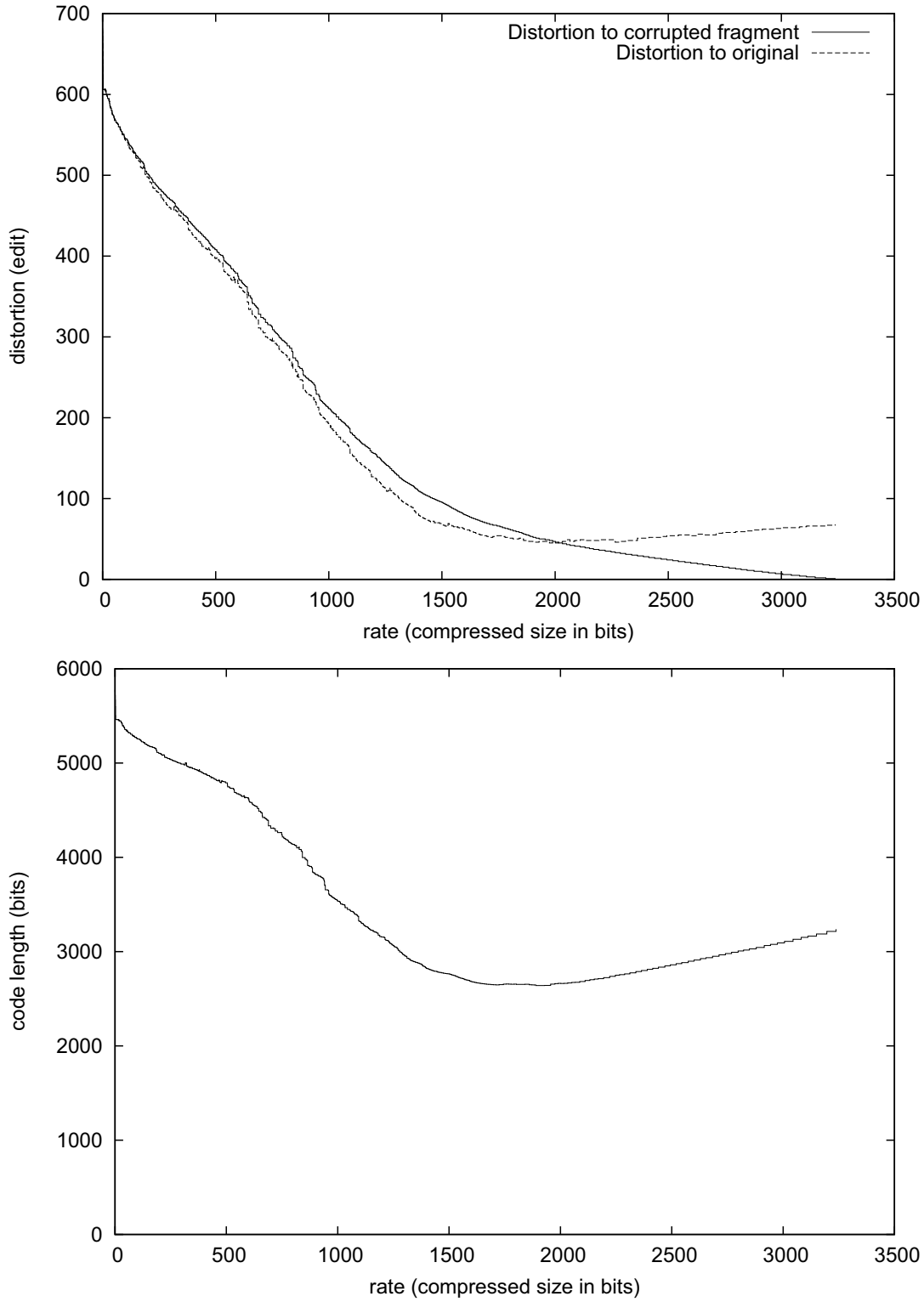
Beauty, real beauty, ends2wheresan intellectual expressoon begins. IntellHct isg in itself a mMde ofSexggeration, an\ destroys theLharmony of n face. :The m1ment one sits down to ahink@ one becomes jll noeˆ Nor all forehbead, or something hNrrid. Look a Ghe successf\l men in anL of te JeaFned professions. How per}ectly tideous 4they re6 Except, of corze, in7 the Ch4rch. BuP ehen in the Church they dol't bthink. =A bishop keeps on saying at the age of eighty what he was told to say wh"n he was aJb4y of eighten, and sja natural cnsequence fhe a(ways looks ab8olstely de[ightfu). Your mysterious youngL friend, wPose name you h\vo never tld me, mut whose picture really fa?scinates Lme,Pnever thinCs. I feel quite surS of that9

(b) $\mathbb{W}_{\text{IN}}$, the corrupted version of the fragment. At 68 randomly selected positions characters have been inserted, deleted or modified. New and replacement characters are drawn uniformly from ASCII symbols 32–126.

Beauty, real beauty, ends-where an intellectual expressoon begins. Intellect is⊡ in itself a mode of ex⊡ggeration, and destroys the harmony of ⊡n⊡ face. ⊡The moment one sits down to think⊡ one becomes ⊡ll no⊡e⊡ ⊡of all forebe⊡d, or something hirrid. Look a⊡ ⊡he successf⊡l men in an⊡ of t⊡e yearned provessions. How perfectly tideous ⊡they ⊡re6 Except, of co⊡r⊡e, in ⊡ the Charch. But when in the Church they dol't ⊡think. ⊡A⊡bishop keeps on saying at the age of eighty what he was told to say wh⊡n he was a⊡bsy of eight⊡en, and ⊡sja natural c⊡nsequence the a⊡ways looks absolutely de⊡ightful. Your mysterious young⊡ friend, whose name you have never t⊡ld me, mut whose picture really fa⊡scinates ⊡me, never thinCs. I feel quite sur⊡ of that9

(c) $\mathbb{W}_{\text{BEST}} = \mathbb{W}_{\text{MSS}}$; it has edit distortion 46 to the original fragment. Marks indicate the error type: ▲=correction; ▼=new error; ●=old error; ★=changed but still wrong. Deletions are represented as ⊡.

Figure 6.11 Results for a fragment of *The Picture of Dorian Gray* by Oscar Wilde (also see Figure 6.10).



6.5 Quality of the Approximation

It is easy to see from its definition that the distortion-rate function must be a non-increasing function of the rate. The implementation guarantees that our approximation is non-increasing as well. In [92] it is assumed that for every $x \in \mathcal{X}$ there exists a representation $y \in \mathcal{Y}$ such that $d(x, y) = 0$; in the context of this chapter this is certainly true because we have $\mathcal{X} = \mathcal{Y}$ and a distortion function which is a metric. The gene pool is initialised with $\mathbb{O}_{\text{IN}}$, which always has zero weakness and must therefore remain in the pool. Therefore at a rate that is high enough to specify x, the distortion-rate function reaches zero.

The shape of the code length function for an object x is more complicated. Let y be the representation for which $d(x, y) = 0$. In theory, the code length can never become less than the complexity of y, and the minimal sufficient statistic witnesses the code length function at the lowest rate at which the code length is equal to the complexity of y. Practically, we found in all denoising experiments that the total code length using the minimal sufficient statistic, $\tilde{K}(\mathbb{O}_{\text{MSS}}) + L_{\mathbb{O}_{\text{MSS}}}(\mathbb{O}_{\text{IN}})$, is *lower* than the code length $\tilde{K}(\mathbb{O}_{\text{IN}})$ that is obtained by compressing the input object directly. This can be observed in Figures 6.6, 6.8 and 6.11. The effect is most pronounced in Figure 6.6, where the separation between structure and noise is most pronounced.

Our hypothesis is that this departure from the theoretical shape of the code length function must be explained by inefficiency of the compression algorithm in dealing with noise. This is evidenced by the fact that it needs 2735.7 bits to encode $\mathbb{C}_{\text{NOISE}}$, while only $\log_2 \binom{4096}{377} \approx 1810$ bits would suffice if the noise were specified with a uniform code on the set of indices of all binary sequences with exactly 377 ones out of $64 \cdot 64$ (see Section 6.2.3). Similarly, $\tilde{K}(\mathbb{N}_{\text{NOISE}}) = 14093$, whereas a literal encoding requires at most 12829 bits (using the bound from Section 6.2.3).

Another strange effect occurs in Figure 6.8, where the code length function displays a strange "bump": as the rate is increased beyond the level required to specify the minimal sufficient statistic, the code length goes up as before, but here at very high rates the code length starts dropping again.

It is theoretically possible that the code length function should exhibit such behaviour to a limited extent. It can be seen in [92] that a temporary increase in the code length function can occur up to a number of bits that depends on the so-called *covering coefficient*. Loosely speaking this is the density of small distortion balls that is required in order to completely cover a larger distortion ball. The covering coefficient in turn depends on the used distortion function and the number of dimensions. It is quite hard to analyse in the case of Euclidean distortion, so we cannot at present say if theory admits such a large increase in the code length function. However, we believe that the explanation is more mundane in this case. Since the noisy mouse is the most complex object of the four we experimented on, we fear that this bump may simply indicate that we

interrupted our search procedure too soon. Quite possibly, after a few years of processing on more expensive hardware, the code length function would have run straight in between $\mathbb{N}_{\mathrm{MSS}}$ and $\mathbb{N}_{\mathrm{IN}}$.

Figure 6.5 shows that our approximation of the distortion-rate function is somewhat better than the approximation provided by either JPEG or JPEG2000, although the difference is not extremely large for higher rates. The probable reason is twofold: on the one hand, we do not know for which distortion function JPEG(2000) is optimised, but it is probably not Euclidean distortion. Therefore our comparison is somewhat unfair, since our method might well perform worse on JPEG(2000)'s own distortion measure. On the other hand, JPEG(2000) is very time-efficient, it took only a matter of seconds to compute models at various different quality levels, while it took our own algorithm days or weeks to compute its distortion-rate approximation. Two conclusions can be drawn from our result. Namely, if the performance of existing image compression software had been better than the performance of our own method in our experiments, this would have been evidence to suggest that our algorithm does not compute a good approximation to the rate-distortion function. The fact that this is not the case is thus reassuring. Vice versa, if we assume that we have computed a good approximation to the algorithmic rate-distortion function, then our results give a measure of how close JPEG(2000) comes to the theoretical optimum; our program can thus be used to provide a basis for the evaluation of the performance of lossy compressors.

6.6 An MDL Perspective

So far we have described algorithmic rate-distortion theory in terms of the uncomputable notion of Kolmogorov complexity, and we developed a practical version of the method using a data compression algorithm. In the context of this thesis, it is natural to ask how the practical method we described above fits within established MDL theory. After all, Minimum Description Length was originally developed to obtain a practical version of universal learning based on Kolmogorov complexity, which can even be interpreted as a special case ("ideal MDL", Section 1.1.3). For example, a major theme in the MDL literature is the motivation of the used codes; some codes are considered acceptable, others are not. To what category belongs the data compression algorithm we used?

First we compare the code length function used in MDL to that we used for algorithmic rate-distortion in this chapter. In the considered practical version of algorithmic rate distortion, by (6.2) and (6.7), the total code length of the source object $x \in \mathcal{X}$ using a representation $y \in \mathcal{Y}$ equals

$$\tilde{K}(y) + L_D(d(x,y)|y) + \log |S_y(d(x,y))|. \tag{6.8}$$

In this three part code, the first term counts the number of bits required to

specify the representation, or hypothesis, for the data and the last term counts the number of bits required to specify the noise. Whether the distortion level of the source object given a representation should be interpreted as part of the hypothesis or as noise is debatable; in this chapter we found it convenient to treat the distortion level as part of the hypothesis, as in (6.2). But to match algorithmic rate-distortion to MDL it is better to think of the distortion level as part of the noise and identify $\mathcal{Y}$ with the available hypotheses.

In our description of MDL in the introductory chapter, the total code length of the data $x \in \mathcal{X}$ with the help of a hypothesis $y \in \mathcal{Y}$ is

$$L(y) + L_y(x). \tag{6.9}$$

Algorithmic rate-distortion theory can be understood as a generalisation of MDL as described in the introduction by making (6.8) and (6.9) match. Suppose that we start with an instance of MDL model selection, specified by the code length functions $L(y)$ and $L_y(x)$, and let $P_y(x) = 2^{-L_y(x)}$ be the mass function corresponding to $L_y(x)$. We will show that the model selected by MDL is equal to a sufficient statistic in the rate-distortion problem that is obtained by setting $\tilde{K}(y) = L(y)$, $d(x,y) = L_y(x)$ and $L_D(d|y) = P_y(L_y(X) = d)$. We have

$$\begin{aligned} P_y(x) &= P_y(X = x, L_y(X) = L_y(x)) \\ &= P_y(X = x | L_y(X) = L_y(x)) P_y(L_y(X) = L_y(x)) \\ &= \frac{P_y(L_y(X) = L_y(x))}{|\{x' : L_y(x') = L_y(x)\}|} = \frac{2^{-L_D(d(x,y)|y)}}{|S_y(d(x,y))|}, \end{aligned}$$

so that the code length function that is minimised by sufficient statistics in this rate-distortion problem is exactly the same code length function that is minimised in the original MDL problem. Note that to make this work we only really need the condition that the distortion function has the property that for all $y \in \mathcal{Y}$ and all $x \in \mathcal{X}, x' \in \mathcal{X}$ we have $d(x,y) = d(x',y)$ iff $L_y(x) = L_y(x')$. We chose $d(x,y) = L_y(x)$ because it is a simple function with that property, and because it allows an interpretation of the distortion as the incurred logarithmic loss.

This correspondence actually works both ways: starting with a rate-distortion problem specified by some $\tilde{K}$, code L_D and distortion function d, we can also construct an MDL problem that identifies a sufficient statistic by defining $L(y) = \tilde{K}(y)$ and $L_y(x) = L_D(d(x,y)|y) + \log |S_y(d(x,y))|$.

Note that in the introductory chapter we were somewhat unspecific as to *which* hypothesis should be selected if more than one of them minimises the code length; the issue becomes much clearer in a rate-distortion analysis because it allows us to easily express a preference for the *minimal* sufficient statistic as the best hypothesis for the data. If a thorough analysis of the data is required, it also seems sensible to look at the whole rate-distortion function rather than just at the minimal sufficient statistic, since it provides additional information about the structure that is present in the data at each level of complexity.

When we introduced MDL, we stressed that a suitable code L for the specification of hypotheses must have two properties: (a) it has small regret in the worst case over all considered input objects, and (b) it may be a luckiness code that does especially well on some inputs (see Section 1.1). In the experiments in this chapter we used $L = \tilde{K}$, where $\tilde{K}$ is a general purpose data compression algorithm. We need to check that it achieves small regret in the worst case. We will do this for the experiment on the noisy mouse $\mathbb{N}_{\mathrm{IN}}$. In that experiment we only considered finitely many hypotheses, so as in Example 2, the best possible worst-case regret is achieved by the uniform code. This code requires $64 \cdot 40 \cdot 8 = 20480$ bits for all representations in $\mathcal{Y}$. The encoding produced by the data compression algorithm should never be much longer. We did not prove any bounds for our compressor, but we did try to compress ten files of $64 \cdot 40$ random bytes each, to see whether or not the compressed representation would be much larger; the worst result was a compressed size of 20812 bytes. This blowup of 332 bits does seem larger than necessary, but similar or worse overheads are incurred by other compression software such as Lempel-Ziv based compressors and PPM compressors, unless as a built-in feature it checks whether or not it actually manages to compress the input object, reverting to a literal encoding when this fails. All in all, the worst-case performance of the compressor appears to lie within reasonable bounds, although admittedly we did not check this very thoroughly.

The fact that we subject the source object to a rate-distortion analysis in the first place suggests that we believe that reasonable hypotheses can be formulated at different levels of complexity, which means that we should certainly use a luckiness code to differentiate between simple and more complex representations. This is what data compression software is designed to do really well, and what allows for an interesting analysis in the case of our example. Another way to construct a suitable luckiness code L, which is more in the spirit of the standard MDL approach, is the following. We start out with a countable set of models of different complexity $\mathcal{M}_1$, $\mathcal{M}_2$, $\ldots$, and take its union $\mathcal{Y} = \cup_n \mathcal{M}_n$ as the set of representations. We can then encode y by first specifying its model index i using some worst-case efficient code, and then specifying y within model i using a second worst-case efficient code. The details of such an approach to rate-distortion have not yet been investigated.

6.7 Conclusion

Algorithmic rate-distortion provides a good framework for analysis of large and structured objects. It is based on Kolmogorov complexity, which is not computable. We nevertheless attempted to put this theory into practice by approximating the Kolmogorov complexity of an object by its compressed size. We also generalised the theory in order to enable it to cope with side information, which is interpreted as being available to both the sender and the receiver in a transmission

over a rate restricted channel. We also describe how algorithmic rate-distortion theory may be applied to lossy compression and denoising problems.

Finding the approximate rate-distortion function of an individual object is a difficult search problem. We describe a genetic algorithm that is very slow, but has the important advantage that it requires only few assumptions about the problem at hand. Judging from our experimental results, our algorithm provides a good approximation, as long as its input object is of reasonably low complexity and is compressible by the used data compressor. The shape of the approximate rate-distortion function, and especially that of the associated three part code length function, is reasonably similar to the shape that we would expect on the basis of theory, but there is a striking difference as well: at rates higher than the complexity of the minimal sufficient statistic, the three part code length tends to increase with the rate, where theory suggests it should remain constant. We expect that this effect can be attributed to inefficiencies in the compressor.

We find that the algorithm performs quite well in lossy compression, with apparently somewhat better image quality than that achieved by JPEG2000, although the comparison may not be altogether fair. When applied to denoising, the minimal sufficient statistic tends to be a slight underestimate of the complexity of the best possible denoising (an example of underfitting). This is presumably again due to inefficiencies in the used compression algorithm.

Beside an *approximation* of algorithmic rate-distortion theory, the computable version of the theory can also be interpreted as a *generalisation* of MDL model selection. We concluded this chapter with a discussion of this relationship.