PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría
en Informática

Lossless Data Compression via Sparse Models and its application to Binary Images

Alix Lhéritier

2010

Lossless data compression via sparse models and its application to binary images
Lhéritier, Alix
ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 10-04
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, febrero de 2010

Master's Thesis in Computer Science

Lossless Data Compression via Sparse Models and its application to Binary Images

Alix Lhéritier

alherit@gmail.com

Thesis Advisor:

Dr. Gadiel Seroussi

Hewlett-Packard Laboratories and Facultad de Ingeniería de la Universidad de la República

gseroussi@ieee.org

Academic Advisor:

Dr. Alfredo Viola

Facultad de Ingeniería de la Universidad de la República

viola@fing.edu.uy

February 2010

Abstract

In lossless data compression, the goal is describing a given sequence $x^n = x_1 x_2 \dots x_n$ of n symbols in a shorter manner by using a code. This can be shown to be equivalent to the problem of capturing statistical regularities of the given sequence by means of a statistical model that assigns probabilities to sequences.[1] If a model assigns a probability $P(x^n)$ to a given sequence, then the sequence can be described [Pas76, Ris76] with a length $L(x^n) = -\log P(x^n) + O(1)$ bits when $n \to \infty$. Thus, the higher the probability assigned by the model, the shorter the sequence can be described. For example, in order to better capture these regularities and get shorter sequence descriptions, models may condition the probability of each symbol on the values of symbols belonging to a neighboring context. Although the probability assignments need not be sequential, in this thesis, we focus on the sequential ones and, in particular, on finite memory models, i.e., models where the probability of a symbol is conditioned on a finite number of past symbols.

When considering a class of models, the goal of universal coding is to find an algorithm that describes any given sequence x^n with a length per input symbol that is asymptotically as short as the one given by the best model in the class for x^n. In this case, the model is not assumed known in advance and, therefore, it needs to be described along with the data or learned from the data itself. Indeed, while a richer class may have an optimum model that assigns a higher probability (i.e. a model that better "fits" the data), Rissanen's lower bound [Ris84, Theorem 1] shows, in a stochastic setting, that there is a "model description cost" to be paid, which is proportional to the number of free parameters that describes the models of the class.

Going further, twice-universal coding aims at finding an optimum for this trade-off between model fitness and model cost, by considering a sequence of model classes of growing dimensionality and optimizing the size of the class at the same time it optimizes the particular model.

Contiguous context modeling (i.e., Markov modeling in which the probability of a symbol is conditioned on the K contiguous symbols that precede it), though quite popular, has the drawback that the number of free parameters, which is proportional to the number of possible conditioning states, increases exponentially with the distance, K, to the furthest conditioning location. Thus, a high model cost is incurred if locations are needed in order to capture distant dependencies. Such distant dependencies do occur in some types of data, e.g., images of text, where the value of a pixel may depend on traces that occur far from the pixel.

Tree models, for which twice-universal coding algorithms exist, allow much economy (in terms of model cost) by allowing a variable length context. Still, in this case, the context is contiguous and, sometimes, this forces the inclusion of dependencies that unnecessarily increase model cost.

Sparse models are a type of statistical model in which each sample can be conditioned on neighboring samples that are not necessarily contiguous. Therefore, they allow to capture some distant dependencies without being forced to include all the contiguous locations in between, potentially saving model cost. Additionally, these sparse contexts can be of variable length, which can provide additional model cost savings.

The main problem studied in this thesis is how to efficiently estimate, for some given data, the best set of fixed conditioning locations within a window of size K. This problem has been addressed with different approaches. For example, exact algorithms with high complexity that are

[1]Probabilities must be consistent, i.e., the sum of the probabilities of all the sequences of length n must be 1.

practical only for small values of K have been proposed in [FWA04] and exact algorithms for a highly restricted class of sparse models that are practical for larger K values (around 100) have been proposed in [FLSV, FSV08, Fra08].

In this work, we adopt an heuristic approach similar to that of [RSP08, SIH01]. We present and empirically analyze two heuristic algorithms (a greedy and a genetic one) for the search of good location sets (templates) for two specific classes of sparse models. The compression performance of the greedy algorithm is quite satisfactory in many cases but it is prone, in other cases, to getting stuck in some bad local minima because of its deterministic nature. Although computationally more expensive in general when compared to the greedy one, our genetic algorithm, which improves in many aspects the genetic algorithm implementation of [Ser04], overcomes these difficulties, and has very good compression performance and reasonable computational cost even in the case of sparse models with variable length conditioning and rather large window sizes (around 1000). Additionally, even in cases where the encoding is slower, due to the search for the best context template, the decoding is fast, since the optimized template is appended to the encoded data and, therefore, available to the decoder. This is appropriate in applications of data compression where the data is encoded once and stored, but it is accessed (and decompressed) many times.

Based on the results given by these algorithms, we observe that sparse context modeling has an important potential for compressing binary images because of their great ability to economically capture different types of regular structures that can be found in this kind of data. Our algorithms, in many cases, largely outperform standard binary image compression algorithms.

keywords: statistical modeling, context modeling, lossless compression, twice-universal coding, binary images, genetic algorithms, greedy algorithms, sparse contexts, tree models, sparse tree models

Resumen

En compresión de datos sin pérdida, el objetivo es describir una secuencia dada $x^n = x_1 x_2 \dots x_n$ de n símbolos en una forma más corta mediante el uso de un código. Esto puede demostrarse que es equivalente al problema de capturar regularidades estadísticas de la secuencia dada, por medio de un modelo estadístico que asigna probabilidades a las secuencias.[2] Si un modelo asigna una probabilidad $P(x^n)$ a una secuencia dada, la secuencia se puede describir [Pas76, Ris76] con una longitud $L(x^n) = -\log P(x^n) + O(1)$ bits cuando $n \to \infty$. Por lo tanto, cuanto mayor es la probabilidad asignada por el modelo, más corta puede ser la descripción de la secuencia. Por ejemplo, con el fin de capturar mejor estas regularidades y obtener una descripción más corta de la secuencia, los modelos pueden condicionar la probabilidad de cada símbolo basándose en los valores de los símbolos que pertenecen a un contexto cercano. Si bien las asignaciones de probabilidad no necesitan ser secuenciales, en esta tesis, nos enfocamos en las secuenciales y, en particular, en los modelos de memoria finita, es decir, modelos en los que la probabilidad de un símbolo está condicionada a un número finito de símbolos del pasado.

Cuando se considera una clase de modelos, el objetivo de la codificación universal es encontrar un algoritmo que describe cualquier secuencia dada x^n con una longitud por símbolo de entrada asintóticamente tan corta como aquella dada por el mejor modelo de la clase para x^n. En este caso, el modelo no se asume conocido de antemano y, por tanto, debe ser descrito junto con los datos o extraído de los propios datos. De hecho, mientras que una clase rica puede tener un modelo óptimo que asigna una probabilidad más alta (es decir, un modelo que se "ajusta" mejor a los datos), la cota inferior de Rissanen [Ris84, Teorema 1] pone de manifiesto, en una configuración estocástica, que existe un costo de descripción del modelo a pagar, que es proporcional al número de parámetros libres que describe los modelos de la clase.

Yendo más lejos, la codificación doblemente universal aspira a encontrar un grado óptimo para esta disyuntiva entre el ajuste del modelo a los datos y el costo del mismo, al considerar una secuencia de clases de modelos de creciente dimensionalidad y al optimizar el tamaño de la clase al mismo tiempo que se optimiza el modelo en particular.

El modelado de contexto contiguo (es decir, modelos de Markov en el cual la probabilidad de un símbolo está condicionada a los K símbolos contiguos que lo preceden), aunque muy popular, tiene el inconveniente de que el número de parámetros libres, que es proporcional a la cantidad de posibles estados, aumenta exponencialmente con la distancia, K, a la ubicación del símbolo condicionante más lejano. Así, se incurre en un alto costo de modelo si se necesitan ubicaciones para capturar dependencias lejanas. Estas dependencias lejanas se producen en algunos tipos de datos, por ejemplo, en las imágenes de texto, donde el valor de un píxel puede depender de huellas que se producen lejos del mismo.

Los modelos árbol, para los cuales existen algoritmos de codificación doblemente universales, facultan importantes ahorros (en términos de costo del modelo) al permitir un contexto de longitud variable. Sin embargo, en este caso, el contexto es contiguo y, a veces, esto obliga a la inclusión de dependencias que aumentan innecesariamente el costo del modelo.

Los modelos dispersos son un tipo de modelo estadístico en los cuales cada muestra puede ser condicionada a muestras vecinas que no son necesariamente contiguas. Por lo tanto, permiten la

[2] Las probabilidades deben ser coherentes, es decir, la suma de las probabilidades de todas las secuencias de largo n debe ser 1.

captura de algunas dependencias distantes sin estar obligados a incluir a todos los lugares contiguos en el medio, con potencial ahorro en el costo del modelo. Además, estos contextos dispersos pueden ser de longitud variable, lo cual puede proporcionar ahorros adicionales en el costo del modelo.

El principal problema estudiado en esta tesis es cómo estimar eficientemente, para ciertos datos dados, el mejor conjunto fijo de ubicaciones condicionantes dentro de una ventana de tamaño K. Este problema ha sido abordado con diferentes enfoques. Por ejemplo, algoritmos exactos de alta complejidad que son prácticos sólo para valores pequeños de K han sido propuestos en [FWA04] y algoritmos exactos para una clase muy restringida de modelos dispersos que son prácticos para valores más grandes de K (alrededor de 100) han sido propuestos en [FLSV, FSV08, Fra08].

En este trabajo, adoptamos un enfoque heurístico similar al de [RSP08, SIH01]. Se presentan y analizan empíricamente dos algoritmos heurísticos (uno voraz y otro genético) para la búsqueda de buenos conjuntos de ubicaciones (plantillas) para dos clases específicas de modelos dispersos. El rendimiento de compresión del algoritmo voraz es bastante satisfactorio en muchos casos, pero es propenso, en otros casos, a quedarse atascado en algunos mínimos locales malos debido a su naturaleza determinista. Aunque computacionalmente más costoso, en general, en comparación con el algoritmo voraz, nuestro algoritmo genético, el cual mejora en muchos aspectos el algoritmo genético de [Ser04], supera estas dificultades, y tiene un rendimiento de compresión muy bueno y un costo computacional razonable, incluso en el caso de modelos dispersos con condicionamiento de longitud variable y tamaños de ventana más grandes (alrededor de 1000). Además, incluso en los casos en que la codificación es más lenta, debido a la búsqueda de la mejor plantilla de contexto, la decodificación es rápida, ya que la plantilla optimizada se añade a los datos codificados y, por tanto, queda disponible para el decodificador. Esto es adecuado en aplicaciones de compresión de datos donde los datos se codifican una vez y se almacenan, pero son accedidos (y descomprimidos) muchas veces.

Basado en los resultados obtenidos por estos algoritmos, se observa que el modelado de contexto disperso tiene un importante potencial para la compresión de imágenes binarias, debido a su gran capacidad para capturar económicamente los diferentes tipos de estructuras regulares que se pueden encontrar en este tipo de datos. Nuestros algoritmos, en muchos casos, superan en gran medida a algoritmos estándares de compresión de imágenes binarias.

palabras claves: modelado estadístico, modelado de contexto, compresión sin pérdida, codificación doblemente universal, imágenes binarias, algoritmos genéticos, algoritmos voraces, contextos dispersos, modelos árbol, modelos árbol dispersos

Contents

Acknowledgments

First, I would like to thank my wife Ana, my daughter Amélie and my son Yann for supporting me and for enduring this long process, and my parents for their support and for encouraging me to take the path of science in a country where science is undervalued. I would also like to thank the rest of my family and my friends for giving me their support at all times.

I would like to express my gratitude to my advisors for giving me the chance to work on such an interesting and challenging project, for their guidance throughout the whole project and for giving me the opportunity of visiting the Mathematical Sciences Research Institute and the University of Minnesota, which was an enriching and unique experience.

I would like to especially thank Guillermo Sapiro for receiving me so warmly in his laboratory in which I met many interesting people that I would like to thank for the good times we shared. These people are Iman Aganj, Pablo Arias, Gloria Haro, Hstau Liao, Mona Mahmoudi, Anish Mohan, Kedar Patwardhan , Alexis Protière, Diego Rother and Pablo Sprechmann.

I would also like to thank Álvaro Martín, Gonzalo Tejera, Gustavo Brown and Ignacio Ramírez of the “Facultad de Ingeniería”, who gave me their moral support and valuable advices in tough times. I also want to highlight and acknowledge the help that Álvaro Martín gave me to integrate its software libraries into my programs.

I would also like to thank Margot for caring about me and for his optimistic messages in difficult times.

Special thanks go to my ex-coworkers of the Banco Central del Uruguay and friends Guillermo Pérez and Gabriel Yermán for their moral support and for the valuable computing resources that they gave me to run a large number of experiments. For the same computing reasons, I would also like to thank my mother and Andrzej Lipinski who also ran several experiments on their PCs.

Agradecimientos

En primer lugar, quisiera agradecer a mi esposa Ana, a mi hija Amélie y a mi hijo Yann por apoyarme y por soportar este largo proceso, y a mis padres por su apoyo y por alentarme a tomar el camino de la ciencia en un país donde la ciencia no se valora adecuadamente. También me gustaría dar las gracias al resto de mi familia y a mis amigos por darme su apoyo en todo momento.

Me gustaría expresar mi gratitud a mis tutores por haberme dado la oportunidad de trabajar en un proyecto tan interesante y desafiante, por su orientación a lo largo de todo el proyecto y por darme la oportunidad de visitar el Mathematical Sciences Research Institute y la Universidad de Minnesota, lo cual fue una experiencia enriquecedora y única.

Quisiera agradecer especialmente a Guillermo Sapiro por haberme recibido tan calurosamente en su laboratorio en el cual conocí a muchas personas interesantes a las cuales me gustaría agradecer por los buenos momentos que compartimos. Estas personas son Iman Aganj, Pablo Arias, Gloria Haro, Hstau Liao, Mona Mahmoudi, Anish Mohan, Kedar Patwardhan, Alexis Protière, Diego Rother y Pablo Sprechmann.

También me gustaría dar las gracias a Álvaro Martín, Gonzalo Tejera, Gustavo Brown e Ignacio Ramírez de la Facultad de Ingeniería, los cuales me dieron su apoyo moral y consejos valiosos en momentos difíciles. Además, quiero destacar y agradecer la ayuda que Álvaro Martín me brindó para integrar sus bibliotecas de software en mis programas.

También quiero agradecer a Margot por preocuparse por mí y por sus mensajes optimistas en momentos difíciles.

Un agradecimiento especial va para mis ex-compañeros del Banco Central del Uruguay y amigos Guillermo Pérez y Gabriel Yermán por su apoyo moral y por los valiosos recursos computacionales que me brindaron para ejecutar una gran cantidad de experimentos. Por las mismas razones computacionales, también me gustaría dar las gracias a mi madre y a Andrzej Lipinski que corrieron también muchos experimentos en sus PCs.

Notations and Abbreviations

A	A discrete finite alphabet of symbols, page 1
α	The alphabet size, i.e., $\alpha = \|A\|$, page 1
$()$	Notation for indexed arrays or vectors, e.g., $s = (x_1, x_2, x_3)$, page 10
$:=$	Assignments in algorithms, page 26
BRGTO	Basic Randomized Genetic Template Optimization, page 22
$C_{\mathcal{T}}\left(x^{i-1}\right)$	Conditioning state $x_{i-l_k} x_{i-l_{t+1}} \dots x_{i-l_1}$ defined by a template $\mathcal{T} = l_1, l_2, \dots, l_k$, page 17
$C_{\mathcal{T}}(x^{m \times n}, i, j)$	Conditioning state of the current sample $x_{i,j}$ of a two-dimensional sequence, according to the template $\mathcal{T}$, page 26
$C(x^{i-1})$	Conditioning state function of x_i, page 14
$\sim$	"distributed as", e.g., $X \sim p(x)$ means that the random variable x obeys the probability law function $p(x)$, page 9
DITO	Deterministic Incremental Template Optimization. Subscripts $_D$ or $_S$ indicate that, respectively, deletions or substitutions are performed (additions are always performed). As a superscript we indicate in which model space the algorithm searchs, i.e., K-SCMs or K-WLSTMs. For example, $\mathrm{DITO}_D^{K-\mathrm{WLSTM}}$ refers to the greedy algorithm that performs deletions in the setting of K-WLSTMs, page 43
DjVu	A system for compressing images, page 28
$\mathbf{E}_p[F]$	The expectation of F with respect to the probability distribution p, page 13
ERGTO	Enhanced Randomized Genetic Template Optimization. As a superscript we indicate in which model space the algorithm searchs, i.e., K-SCMs or K-WLSTMs. For example, $\mathrm{ERGTO}^{K-\mathrm{WLSTM}}$ refers to the enhanced genetic algorithm that searchs in the space of K-WLSTMs. When omitted, the superscript is assumed to be K-WLSTM, page 55
italics	Concepts that are introduced for the first time are shown in italics, page 1
JBIG2	Joint Bi-level Image experts Group Standard, ISO/IEC 14492 and ITU T.88, page 28

JBIG	Joint Bi-level Image experts Group Standard, ISO/IEC 11544 and ITU-TRec. T. 82, page 28
K	Order of finite memory (or Markov) model, page 2
k	Weight (i.e., number of locations) of the context template, page 17
K-SCM	K-window sparse context model, page 4
K-WLSTM	K-window whole level sparse tree model, page 4
KT	Krichevsky-Trofimoff, page 12
$\tilde{K}$	Actual memory window, page 17
$L_c(x^n)$	Code length assigned by a code c to the sequence x^n, page 11
log	Base 2 logarithm, page 9
$n_{x^t}\,(a \mid s)$	Number of times that the symbol a occurs on a state s in x^t, page 12
$n_{x^t}\,(s)$	Number of times that the state s occurs in x^t, page 12
PRNG	Pseudo random number generator, page 21
$P(x_i\|C(x^{i-1}))$	Conditional probability distribution defining a causal context model, page 14
$\{\}$	Notation for sets, e.g., $S = \{a, b, c\}$, page 16
STL	C++ Standard Template Library, page 40
STM	Sparse tree model, page 4
$\mathcal{T}$	A template, i.e., an ordered set of locations that defines the conditioning states of a sparse model, page 16
x_1^∞ or x^∞	A infinite sequence $x_1x_2...x_t...$ of symbols belonging to some alphabet A, page 10
x_1^n or x^n	A finite sequence $x_1x_2...x_n$ of symbols belonging to some alphabet A, page 1
x_i	The current sample, page 14
x_j^k	A subsequence $x_jx_{j+1}...x_k$ of x^n, page 2

Chapter 1

Introduction

1.1 Statistical data modeling and universal lossless compression

In a general framework, the problem we want to address is the one of finding good *statistical models* in order to describe, in a probabilistic fashion, the regularities of a given sequence of data $x^n = x_1x_2 \ldots x_n$, where x_i takes values in some alphabet A of size α. A statistical model assigns probabilities to sequences in a consistent way, i.e., the sum of the probabilities of all the sequences of length n must be 1. There are many applications for which it is necessary or useful to have such models, such as lossless compression, forecasting, denoising, genomic data analysis, financial modeling, etc.

Lossless compression (or *lossless source coding*) aims at describing some given input data in a shorter manner so that the original data can be recovered from this shorter sequence without any loss. This description is always based, explicitly or implicitly, on a statistical model whose suitability is measured by the length of the output data. In opposition, in *lossy compression*, we are allowed to recover a slightly different sequence, which may be appropriate enough for the application (e.g., music or photography for non-professional usage). Both kinds of compression can be very valuable since they allow the efficient use of important resources like data storage (hard disks, optical media, etc.), bandwidth or energy. Lossless compression is of particular interest when modification of the original data is not allowed or not desired, for example, when compressing images that are intended for further analysis and processing or that were obtained at great cost, or when loss might have legal implications. Besides its resource-saving importance, lossless compression is also useful in a pure statistical setting, as the *Minimum Description Length principle* developed by J. Rissanen [Ris78, Ris96, Ris87] states that, if we can achieve a short description of the data, then we are capturing appropriately the redundancies of the data and thus its statistical regularities.

In his seminal paper [Sha48], Shannon showed a fundamental limit for the average description length when compressing the output of a random process, in the case when the parameters that define the process are known. *Universal data compression* deals with the case when the parameters are not known but the process is known to belong to a certain class of models.[1] In this case, Rissanen's lower bound [Ris84] states that Shannon's limit is still asymptotically attainable but at

[1]Universal compression also applies when the input sequence is arbitrary, and is not assumed to have been emitted by a random process, as we will see in Chapter 2.

a convergence rate that includes a *model cost* term proportional to the number of parameters that describe the models in the class.[2]

1.2 Efficient parametrizations of Markov models

In a Kth order *finite memory* (random) process, the probability of a sample x_i in a sequence $x^n = x_1, x_2, ..., x_n$ defined over a finite alphabet A is given by a discrete conditional distribution $P\left(x_i|x_{i-K}^{i-1}\right)$, conditioned on the value of the consecutive $K-$tuple immediately preceding x_i, i.e., the *conditioning state* or *context* of x_i. The probability of the whole sequence x^n is:

$$P\left(x^n\right) = \sum_{x^0_{-K+1}\in A^K} P'\left(x^0_{-K+1}\right)\prod_{i=1}^{n} P\left(x_i|x_{i-K}^{i-1}\right).$$

where P' is some probability distribution that governs the *initial condition* (for example, we could assume that x^0_{-K+1} is some fixed string, which captures all the mass of P').

Finite memory random processes are always Markov chains and, thus, they can be naturally described (or *parametrized*) as Markov models of order K. However, these models require $\alpha^K(\alpha-1)$ parameters corresponding to $\alpha - 1$ free conditional symbol probabilities for each possible conditioning $K-$tuple, where α is the alphabet size.[3] Therefore, the number of free parameters increases exponentially with K.

In a *tree model* [Ris83, WRF95] of the same process, the memory length is allowed to vary from location to location in the sequence.[4] These models are efficient parametrizations of finite memory processes, as the exponential number of statistical parameters in the Markov model can often be dramatically reduced in a well tuned model. Thus, tree models can improve the rate at which the average length of a universal code converges to Shannon's limit. In practice, tree models are appealing because they seem to economically capture redundancies typical of real life data (e.g., text or images) and because of the existence of computationally efficient universal coding schemes [Ris83, WRF95, WST95, Wil98, Noh93, MSW04] for this class of models (see Chapter 2).

The savings in the number of statistical parameters realized by tree models can be seen as the result of lumping together equivalent states, i.e., K-tuples that induce the same conditional distribution in the "full" Markov model. Thus, a conditioning state of $t = K - l$ symbols in a tree model for a Kth order Markov model corresponds to the merging of α^l equivalent states in the full Markov model. This observation, in fact, characterizes the special structure of the sets of full-length states that can be lumped together in a tree model: each such set must consist of all the extensions of a given string of length $K - l$, $0 \le l \le K$. Hence, tree models are represented by *full* α-ary *trees*, i.e., trees in which every node other than the leaves has exactly α children. Figure 1.1 shows a graphical example in which some states of a Markov model get lumped together in a tree model.

With real data, other sets of equivalent states might arise, and it is natural to ask whether it is possible to optimize models where more general sets of states are allowed to merge. In practice, the distributions that are merged are empirical and not necessarily identical, and an exhaustive search for the best state space partition is unfeasible. The problem is also known as the context

[2]From a statistical point of view, this model cost term avoids models that overfit the input data and thus unreliable statistics.

[3]The α-th conditional probability for each K-tuple is not free, as probabilities must add to one.

[4]Tree models are sometimes termed *variable length Markov chains* in the statistical literature (see, e.g., [BW99]).

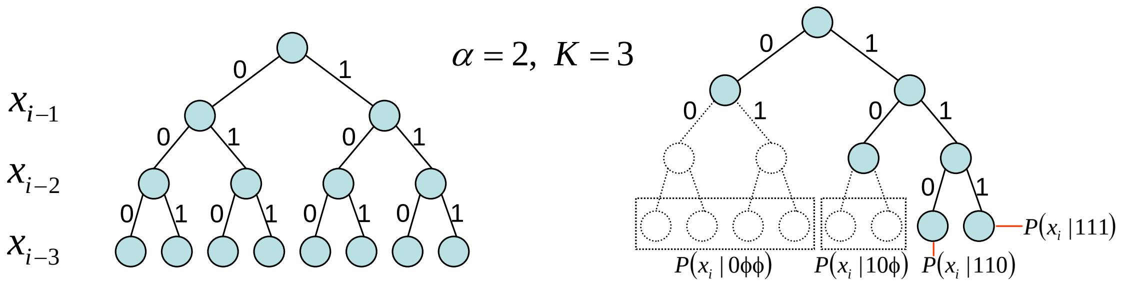


Figure 1.1: The tree model (right) shows the lumping of some states of a fully parametrized Markov model (left). The symbol ϕ indicates that a context location is ignored and, thus, the corresponding states are lumped together. The value of the conditioning state appearing in conditional probabilities is written in reverse order, e.g., $P(x_i|100)$ indicates that the conditioning state is $x_{i-3}x_{i-2}x_{i-1} = 001$.

quantization problem, and it has been studied in various settings, with the proposed solutions being generally ad-hoc, and of varying degrees of complexity as a function of K, which is usually kept relatively small (see, e.g., [FWA04], for a recent setting).

This work focuses on models that use sparse dependencies in the past symbols in order to condition the probability of each symbol of the sequence, with the goal of significantly extending K in practice while keeping the number of conditioning states feasible as shown in Figure 1.2. These sparse models allow the merging of more general sets of states with similar conditional distributions. As for tree models, ϕ refers to an ignored location but, in the case of sparse models, in order to emphasize that these locations may break the context contiguity, we also call them "holes".

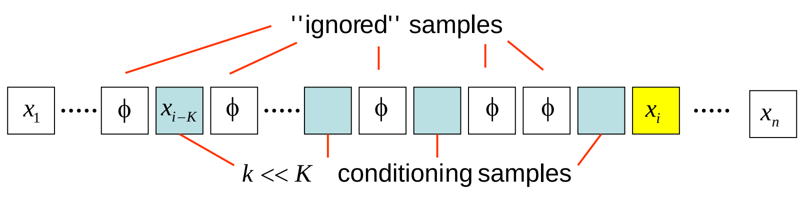


Figure 1.2: Sparse dependencies allowing to extend K, the size of the memory window.

The technique of conditioning on non-contiguous past symbols is known in the literature. The JBIG bi-level image compression standard [Joi93], for example, conditions the current sample on a template of contiguous past samples close to x_i, plus a *floating* past sample that is allowed to be located away from the template. Its successor, JBIG2 [Joi01], allows 4 floating past samples to be used to condition each sample. In [SIH01], the authors suggest using 16 adaptive positions and no fixed locations, in order to extend the JBIG2 standard. They address the problem of choosing these locations, in a window of size $K = 2^{16}$, by using a genetic algorithm as we do in our work (in our case, the number of adaptive locations is not fixed, and will often be larger than 16, see Chapters

3 and 7). Sparse dependencies have also been studied in biology (e.g., [ZHS05, BR04]) but the algorithms presented are practical for rather small values of K. Also in the biology field, [RSP08] takes advantage of sparsity for estimating probability density functions in very high dimensions when sample size is not accordingly high. The algorithm used in that work is similar to the one studied in Section 6.1.

In [Suz95, VW96], *generalized tree models* were studied and a coding scheme with computing complexity exponential in K was presented. Similar to generalized tree models, *sparse tree models (STMs)* are a generalization of tree models where more general sets of states with similar conditional distributions are allowed to merge. In an STM, samples are conditioned on finite strings of not necessarily contiguous past symbols, and the context determine not only how far into the past the conditioning samples are, but also what their (possibly non-contiguous) locations are. In STMs, holes are allowed to be in any place of the tree representation as shown in Figure 1.3. More precisely, every internal node must have either exactly α children or 1 child, the latter case corresponding to a hole.

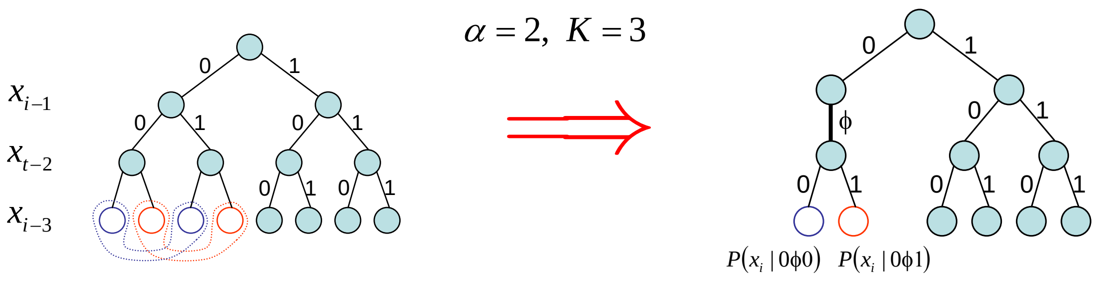


Figure 1.3: Lumping of equivalent states not obeying the complete subtree restriction imposed by tree models.

K-window r-hole STMs have memory bounded by a positive integer K, and each conditioning context used is allowed to have at most r runs of ϕ symbols (i.e., concatenations of holes). In [FSV08, FLSV, Fra08], a semi-predictive algorithm with time complexity $O\left(K^{2r+1}\right)$ that estimates a K-window r-hole STM for a given input sequence is presented and used as the basis for a lossless compression scheme that is universal in the class of K-window, r-hole STMs. The algorithm is implementable in practice up to moderately large values of K around 100 for $r = 1$.

In this work, we focus on two classes of sparse models. The first one is referred to as *K-window sparse context models* (K-SCMs). K-SCMs condition the probability of each sample on a fixed set of locations and, thus, no variable length conditioning is allowed. Figure 1.4 shows an example of how states are lumped together in this type of model.

The other class studied in this work is the one of *K-window whole level sparse tree models (K-WLSTM)*, a subclass of STMs with memory window length limited to K, that has the additional restriction of every level in the tree having either no hole at all or a hole in every edge of the level. In other words, for each tree level, one of these two conditions must be satisfied:

1. all the nodes in the level are either internal ones that have one child (i.e., a hole edge) or are leaves

2. all the nodes in the level have α children.

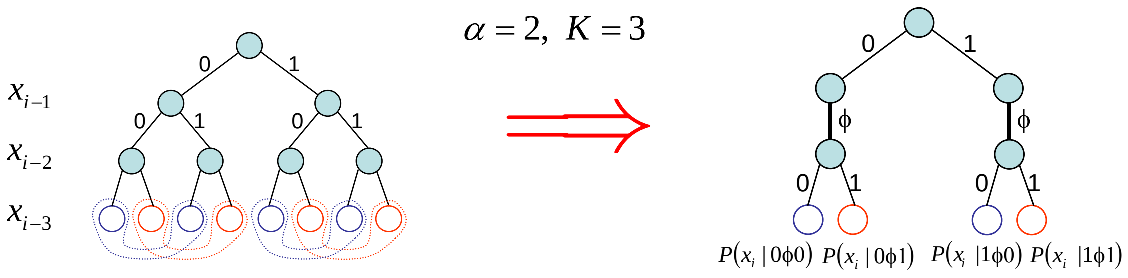


Figure 1.4: Lumping of equivalent states obeying the K-SCM restrictions.

Notice that the example on the right side of Figure 1.3 is not a K-WLSTM, since nodes in the second level do not satisfy either of these conditions. Figure 1.5 shows an example of a K-WLSTM.

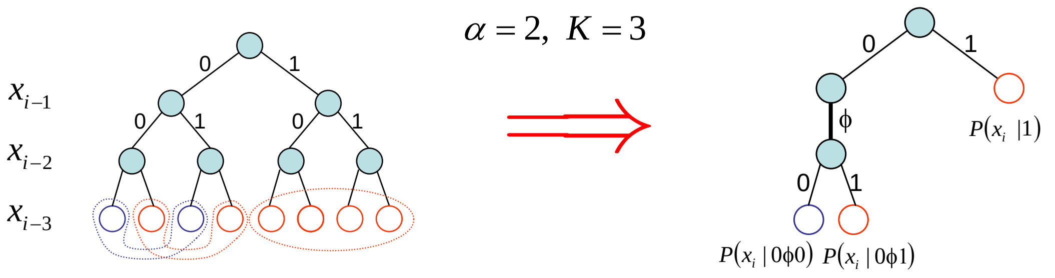


Figure 1.5: Lumping of equivalent states obeying the K-WLSTM restrictions.

Notice that K-SCMs are to K-WLSTMs as fixed-length contiguous context models are to tree models. K-SCMs condition the probability of each sample on a fixed set of locations while K-WLSTMs allow variable length conditioning, the conditioning samples being taken also among a fixed set of locations. Figure 1.6 shows the inclusion relations between the different parametrization of Markov models, where a class A includes a class B if any model of B can be represented by a model of A with the same or a smaller number of states.

One important problem addressed in this work is how to efficiently estimate which is the best set of past dependency locations (or *template*) for these models in order to achieve the minimum description length for some given data. A brute force approach to this problem is of exponential complexity because of its combinatorial nature and, thus, it is not of practical interest if we want to use large values for K.

1.3 Main contributions and organization of the thesis

The starting point of this thesis was a genetic algorithm for searching good templates for K-SCMs, proposed and implemented by G. Seroussi with some promising results on binary images [Ser04]. Then, the primary goals of this thesis were defined as follows:

1. Improve and optimize Seroussi's algorithm and software implementation in order to get better computational performance, thus allowing more experimentation, larger values of K and,

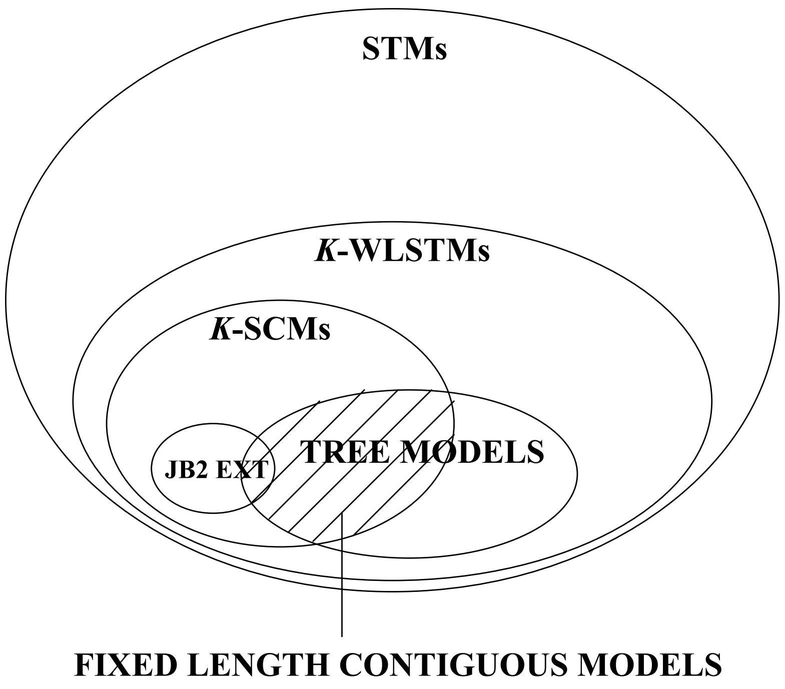


Figure 1.6: Inclusion relations between different Markov model parametrizations. “JB2 EXT” refers to the JBIG2 extension proposed in [SIH01].

ultimately, better compression results.

2. Explore other heuristics.

3. Extend the genetic algorithm implementation, originally only for K-SCMs, to the broader and more flexible class of K-WLSTMs, while keeping a reasonable and practical computational complexity.

4. Experiment with a larger set of binary images in order to get a better understanding of the suitability of sparse models for this kind of data.

In line with these goals, the main contributions in this thesis are the following:

1. An heuristic greedy algorithm for the search of good templates for both K-SCMs and K-WLSTMs (Chapter 6). In general, this algorithm finds good solutions but, because of its determinism, in some cases, it gets stuck in some poor local minima. Nevertheless, the algorithm is relatively fast and practical, even in the case of the K-WLSTMs.

2. An improved genetic algorithm whose components and parameters were chosen by criteria learned from the greedy algorithm (Chapter 7). Although this genetic algorithm is slower than the greedy one, it is more robust and is able to search in the space of K-WLSTMs for rather large values of K (of order 1000). Notice that, for both algorithms, even in cases where the encoding is slower, due to the search for the best context template, the decoding

is fast since the template is described at the beginning of the encoded sequence and does not need to be searched again.

3. Using the results given by our algorithms on a large set of binary images (Chapter 8), we found that, K-WLSTMs are, in some cases, significantly better than K-SCMs thanks to variable-length conditioning that allows to better "fit" the data by considering more dependencies at a reduced model cost. In worst cases, their performance is similar since K-WLSTMs are a superset of K-SCMs and, in exchange, only a constant cost (negligible when the image is large) for tree description has to be paid. We also found that, in many cases, sparse models are significantly better than contiguous models. In worst cases, they are just as good since sparse models are a superset of contiguous models and, in exchange, only a very small constant cost has to be paid for sparse template description. Finally, we found that our improved genetic algorithm for K-WLSTMs outperform, in most general cases, the standard compression methods, and, in some cases, by significant margins.

The rest of this thesis is organized as follows. Chapter 2 sets up the mathematical background and definitions needed for this work. These include some of the theory of lossless data compression and definitions and properties of fixed-length contiguous context, tree and sparse models. Chapter 3 introduces the paradigm of genetic algorithms and describes the scheme proposed by G. Seroussi in [Ser04]. Chapter 4 introduces some types of binary images found in practice and describes some standard methods for compressing them. Chapter 5 discusses some computational issues in sparse model code length evaluation and describes some important optimizations in relation to the computation time. Chapter 6 presents a greedy deterministic algorithm for searching good sparse models. Chapter 7 describes and analyzes a modified and improved version of the original genetic algorithm of [Ser04]. Chapter 8 presents the results and establishes the suitability of sparse models and our algorithms on a large set of binary images. Chapter 9 concludes this work and gives future directions.

Chapter 2

Mathematical background

In this chapter, we present some of the basic mathematical tools and definitions from the area of information theory and the fundamental limits of compression with the associated concept of model cost. Then, we formally define the classes of fixed-length contiguous context, tree, and sparse models and some related coding schemes.

2.1 Lossless compression and model cost

In lossless compression, a *compressor* (or *encoder*) describes a given sequence of symbols in a shorter manner so that a *decompressor* (or *decoder*) can recover the exact original sequence given the compressed one. Thus, the performance is measured by the relation between the length of the compressed sequence (the *code length*, usually measured in bits) and the length of the original one, which is called the *compression ratio* (usually measured in bits per original symbol). Now, we may ask: how much can a given sequence be compressed? The exact answer to this question depends on the nature of the sequence to be compressed and how much we know about it. More precisely, we are going to consider the following two scenarios:

- the probabilistic setting, in which the sequence is a realization of a random process whose parameters we may or may not know.
- the deterministic setting, in which the sequence is deterministic and arbitrary. However, in this case, it is useful to describe the sequence regularities using probability tools, as we will see later.

In the probabilistic setting, the symbols of the input sequence are realizations of random variables. The concept of *entropy* presented by C. Shannon in [Sha48], measures the amount of uncertainty (usually expressed in *bits*) that an observer has about a random variable or, equivalently, the amount of information received by him when the random variable gets realized. Given a random variable X defined over A such that $X \sim p(x)$, the entropy of X is defined as:

$$H(X) = -\sum_{x \in A} p(x) \log p(x), \quad \left[0 \log 0 \triangleq 0\right]$$

The entropy is maximized when the probability is uniformly distributed. When considering the realization of multiple variables, the concept of *joint entropy* arises. The joint entropy of the random variables $X_1, X_2, \ldots, X_n$ is defined as

$$\mathbf{H}(X_1, X_2, \ldots, X_n) = -\sum_{(x_1,x_2,\ldots,x_n)\in A^n} p(x_1, x_2, \ldots, x_n) \log p(x_1, x_2, \ldots, x_n).$$

When normalized, we express it in bits per symbol and write it as

$$H(X_1, X_2, \ldots, X_n) = \frac{1}{n}\mathbf{H}(X_1, X_2, \ldots, X_n)$$

or $H_n(P)$ if we want to emphasize that the generating model assigns a probability $P(\cdot)$ to the sequence.

The extension of this concept to the case of a semi-infinite random process leads to the definition of *entropy rate*:

$$H(X_1^\infty) = \lim_{n\to\infty} \frac{1}{n}\mathbf{H}(X_1^n)$$

which is expressed in bits per symbol, if the limit exists.

A *source code* $\mathcal{C}$ for a random variable X is a mapping $\mathcal{C} : A \to D^*$ where D is a finite *coding alphabet* of size d, and D^* is the set of finite strings over D. $\mathcal{C}(X)$ is the *codeword* corresponding to X and $L(X) = |\mathcal{C}(X)|$ its code length. A code $\mathcal{C} : A \to D^*$ extends naturally to a code $\mathcal{C}^* : A^* \to D^*$ defined by

$$\mathcal{C}^*(\lambda) = \lambda, \qquad \mathcal{C}^*(x_1, x_2 \ldots x_n) = \mathcal{C}(x_1)\mathcal{C}(x_2)\ldots\mathcal{C}(x_n)$$

where λ is the empty string. $\mathcal{C}$ is called *uniquely decodable* if its extension is injective (in other words, every codeword is identifiable when immersed in a sequence of codewords). Kraft-McMillan's inequality [Kra49, McM56] states that the codeword lengths $l_1, l_2, \ldots, l_m$ of any uniquely decodable code satisfy

$$\sum_{i=1}^{m} d^{-l_i} \leq 1$$

where d is the coding alphabet size. Using this inequality, it can be shown (see, for example, [Mac03, p. 97]) that $H(X)$ is a lower bound on the expected number of bits required to describe an outcome of X by a uniquely decodable code.

Using these concepts, Shannon derived a fundamental limit of compression [Sha48] that says that L_n^*, the minimum expected code length per symbol of the sequence to be encoded, satisfies

$$H(X_1, X_2, \ldots, X_n) \leq L_n^* < H(X_1, X_2, \ldots, X_n) + \frac{1}{n}$$

Furthermore, if X_1^∞ is a random process with an entropy rate, then

$$\lim_{n\to\infty} L_n^* = H(X_1^\infty)$$

Thus, the entropy rate can be interpreted as the expected number of bits per symbol required to describe the process.

Given a model that assigns a probability $P(x^n)$ to some given data sequence x^n, the method of *arithmetic coding* presented in [Pas76, Ris76] provides a way to achieve a code length of

$-\log P(x^n) + O(1)$ as $n \to \infty$.[1] Thus, when considering symbols as realizations of random variables, the expected code length given by an arithmetic encoder is minimal up to an additive constant. Even when the sequence is not a realization of random variables, $l^*(x_1^n) = -\log P(x_1^n)$ is called the *ideal code length* for the string x_1^n relative to any probability assignment P to sequences of length n.

What happens when we encode a random variable $X \sim p(x)$ assuming a wrong distribution $q(x)$? The increase in expected description length due to the incorrect distribution is the *relative entropy* (see [CT06, Theorem 5.4.3] for a proof) which is defined as:

$$D(p \,\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

In most practical applications, the model that generates the data is not given to us or, even more, the sequence can be completely arbitrary (i.e., in the deterministic setting). Universal data compression deals with the optimal description of data in the absence of an a priori statistical model. Arithmetic coding provides an effective mean to *sequentially* assign a code word of almost ideal length given a probability assignment, which means that the model probabilities can vary and adapt to the input data as it is observed. This allows us to treat the compression problem as a problem of finding the model that gives the minimum ideal code length for the input data. The model can be based on the whole sequence and, in this case, it is necessary to first describe the model to the decompressor or it can be learned on the fly (in the case of a *strongly* sequential model) and, in this case, the decompressor can do the same. In both cases, there is a model description cost to be paid (in the second case, it manifests itself implicitly as a “learning cost”).

In the deterministic setting, the optimality idea of universal data compression makes sense considering a class of models. We say that a code that assigns a length $L(\cdot)$ to some given data sequence x^n is *pointwise universal* with respect to a class of models C if the *pointwise redundancy* defined as $R_C(L, x^n) = \frac{1}{n}\left[L(x^n) - \min_{c \in C} L_c(x^n)\right]$ satisfies $\lim_{n \to \infty} R_C(L, x^n) = 0$.

Essentially, the choice of a class is an art and some criteria can be the complexity, some prior knowledge on the data and the practical success of some models already presented. One useful way to define a class is to consider a set of models that can be characterized only by a d-dimensional parameter θ. This kind of class is called a *parametric class*. The *maximum likelihood estimate* $\hat{\theta}(x^n)$ characterizes the model of a given parametric class that assigns the highest probability to an input sequence x^n. Therefore, in these cases, $\min_{c \in C} L_c(x^n) = -\log P_{\hat{\theta}(x^n)}(x^n)$.

The *Normalized Maximum Likelihood (NML) code* [Sht87, Ris96] assigns the following probability to the input sequence:

$$Q(x^n) = \frac{2^{-\min_{c \in C} L_c(x^n)}}{\sum_{x^n \in A^n} 2^{-\min_{c \in C} L_c(x^n)}}$$

Therefore, in the case of parametric classes, the NML model assigns to each sequence a probability that is proportional to the probability given by the maximum likelihood estimate. The pointwise redundancy of the NML code is

$$R_C(L_{NML}, x^n) = \frac{1}{n} \log \left[\sum_{x^n \in A^n} 2^{-\min_{c \in C} L_c(x^n)}\right].$$

[1] See [WNC87] for a practical description.

Since this quantity depends only on the class and is the same for all the sequences $x^n \in A^n$, it can be easily shown that this code is optimal in the sense that it attains the best worst case pointwise redundancy which is $R_C = \min_{L} \max_{x^n \in A^n} R_C(L, x^n)$. Under certain reasonable assumptions on a parametric model class, the redundancy can be written as

$$R_C = \frac{d}{2n} \log \frac{n}{2\pi} + \frac{1}{n} \log \int_{\Theta_d} \sqrt{|I(\theta)|} d\theta + o\left(\frac{1}{n}\right)$$

where $I(\cdot)$ is the *Fisher information matrix*[2] and Θ_d is the parameter space. Therefore, R_C grows with the number of parameters and vanishes as n tends to infinity (i.e., the code is pointwise universal). It can be shown (see [BRY98]) that this model can be thought as a *two-part code* which means that we first encode the parameter (i.e., we describe the model) and then the data based on this parameter. Using this point of view, we can see that the model description has a cost that is asymptotically equivalent to $\frac{d}{2} \log n$. Although this code attains the best worst case pointwise redundancy, it has two important drawbacks: it is hard to compute and the sequential probability assignment depends on the horizon n.

Since the NML model has some important drawbacks, an approximation to this model is desirable. The *Krichevsky-Trofimoff (KT) estimator* [KT81] defines a model that is asymptotically (when the data length goes to infinity) as good as the NML model when we consider the class of models that conditions each sample of the sequence on a finite set of *states* S. Each state is determined by some of the previous samples of the sequence. The KT estimator sequentially assigns a probability to a symbol a occurring in a state s according to the following formula: $q_{x^t}(a \mid s) = \frac{n_{x^t}(a|s)+1/2}{n_{x^t}(s)+\alpha/2}$ where α is the alphabet size, $n_{x^t}(s)$ counts the number of times that the state s occurred from the beginning of the sequence up to time t and $n_{x^t}(a \mid s)$ counts the number of times the symbol a occurred on a state s in the same subsequence. As desired, the estimator does not depend on the horizon n and the formula is easy to compute. This estimator assigns the following probability to the whole sequence up to time n:

$$Q(x^n) = \prod_{s \in S} \frac{\Gamma\left(\frac{\alpha}{2}\right) \prod_{a \in A} \Gamma\left(n_{x^n}(a \mid s) + 1/2\right)}{\Gamma\left(n_{x^n}(s) + \frac{\alpha}{2}\right) \Gamma\left(\frac{1}{2}\right)^{\alpha}}$$

where Γ is the Gamma function.

The code length assigned by the KT probability assignment to the symbols of x^n that occur in a state s is

$$L_{KT}(x^n \mid s) \leq n_{x^t}(s) \hat{H}(x \mid s) + \frac{(\alpha - 1)}{2} \log n_{x^t}(s) + O(1)$$

where $\hat{H}(x \mid s) = -\frac{1}{n} \log P_{\hat{\theta}(x^n|s)}(x^n \mid s)$ ($x^n \mid s$ being the subsequence of symbols occurring at state s), i.e., the code length given by the maximum likelihood memoryless model for $x^n \mid s$ (also called *empirical entropy rate*). Adding all the KT code length contributions for every state in S we get

$$L_{KT}(x^n) \leq n\hat{H}(x) + |S| \frac{(\alpha - 1)}{2} \log n_{x^t}(s) + O(|S|)$$

[2]Let $X(\mathbf{x}) = X(x_1, x_2, \ldots, x_n)$ be a random vector in $\mathbb{R}^n$ and let $f_X(\mathbf{x})$ be a probability distribution on X with continuous first and second order partial derivatives. The Fisher information matrix of X is the $n \times n$ matrix I_X whose (i, j)th entry is given by

$$(I_X)_{i,j} = \int_{\mathbb{R}^n} \frac{\partial \ln f_X(\mathbf{x})}{\partial x_i} \frac{\partial \ln f_X(\mathbf{x})}{\partial x_j} f_X(\mathbf{x})\, d^n\mathbf{x}$$

where $\hat{H}(x) = \sum_{s \in S} \frac{n_{x^t}(s)}{n} \hat{H}(x \mid s)$ is the empirical entropy rate of x^n considering a model conditioned on the set of states S. Since $n\hat{H}(x)$ is the minimum code length assigned by a model in the class, pointwise universality of the KT estimator follows from the previous inequality.

Considering a worst-case pointwise redundancy means guaranteed performance, but maybe there are only a few such "unlucky" input sequences? In order to consider averages, it is reasonable to assume that the data was drawn from a model in C, which takes us to the probabilistic setting. Given a probabilistic source model described by a probability function P, the *pointwise redundancy* of a code that assigns a length $L(\cdot)$ to the data is defined as

$$R_{P,L}(x^n) = L(x^n) + \log P(x^n).$$

Also in the stochastic setting, we say that a code that assign a length L to the data is *pointwise universal* if the *normalized maximum pointwise redundancy*, defined as $\frac{1}{n} \max_{x^n \in A^n} R_{P,L}(x^n)$ converges to zero, when $n \to \infty$, for each model in C assigning a probability P. When considering a class of models conditioned on a finite set of states S, a pointwise universal code in the deterministic setting assigns a normalized code length of $\hat{H}(x) + o(1)$ bits. Since $P_{\hat{\theta}(x^n)}$ minimizes $-\log P(x^n)$ among all the distributions P of the models in the considered class, it follows that for a pointwise universal code in the deterministic setting, the normalized maximum pointwise redundancy (assuming the data has been generated by some model in the class) in the same class vanishes with n and, therefore, the code is pointwise universal also in the probabilistic setting. This is the case for the code given by the KT estimator.

The expectation of $R_{P,L}(x^n)$ with respect to P is the *expected redundancy* of the code for the given source, and it is given by

$$\overline{R}_{P,L} = \mathbf{E}_P[L(x^n)] - H_n(P).$$

This quantity is equal to $\frac{1}{n} D(P \| Q)$, namely the normalized relative entropy between the distributions P_θ (that gives the probability of a sequence according to the model) and the distribution $Q(x^n) = 2^{-L(x^n)}$.[3]

A theorem derived by J. Rissanen [Ris84, Theorem 1] for parametric classes, says that if the parameters in Θ_d are "distinguishable" (in some precise sense defined there), then the expected redundancy satisfies, for all Q and all $\varepsilon > 0$, the following inequality:

$$\frac{1}{n} \overline{R}_C(L, \theta) \geq d \frac{\log n}{2n} (1 - \varepsilon), \; L = -\log Q(x^n)$$

for all points θ in Θ_d except in a set whose volume tends to 0 as $n \to \infty$.

The meaning of this bound, which applies in the probabilistic setting, is that $(d/2)\log n$ is the inevitable cost of universality. This lower bound parallels Shannon's coding theorem: when the model is unknown, a model cost that depends on the size of the parameter space gets added to the entropy. The number of parameters affects the achievable convergence rate of a universal code length to the entropy or to the best of the class in the deterministic scenario. Thus, it is important to find classes that fit appropriately the probabilistic source or the individual sequence characteristics, using the fewest possible parameters. In this sense, prior knowledge of the source

[3]By Kraft-McMillan's inequality, it can be shown (see, for example, [Mac03, p. 97]) that, under certain conditions, an encoder that produces a code length $L(\cdot)$ implicitly defines a probability distribution $Q(x^n) = 2^{-L(x^n)}$ for which the code length function is optimal.

data is of paramount importance in order to avoid learning characteristics already known.

One further step in the model optimization problem is to optimize the model size d at the same time that we optimize the particular model. More precisely, consider a model class to be the union of nested parametric classes of growing dimensionality: $\Theta_1 \cup \Theta_2 \cup \ldots \cup \Theta_d \cup \ldots$ where $\Theta_1 \subset \Theta_2 \subset \ldots \subset \Theta_d \subset \ldots$. We say that a code is *twice-universal* in this union of classes, if for any source in the class defined by Θ_d, for any d, the normalized expected redundancy of the code vanishes as $\frac{d}{2n}\log n$ up to lower terms. In the deterministic setting, the definition is analogous but using pointwise redundancy instead of expected redundancy.

A trade-off occurs since increasing the model size allows the model to better fit the data and, thus, the data description length decreases while the model description length increases.

For more details on lossless compression (also called *lossless source coding*) see, e.g., [CT06]. Next, we discuss some important models used in practice under the light of universal data compression theory and the model cost concept.

2.2 Context models

One important class of models is the one of *causal context models*. A causal context model for a sequence x^n is a conditional probability distribution $P\left(x_i \left| C\left(x^{i-1}\right)\right.\right)$, where $C\left(x^{i-1}\right)$ is a function taking values in some arbitrary finite set. The value $C\left(x^{i-1}\right)$ is called the *conditioning state* of x_i and the number of samples on which it depends is called the *memory* of the model. When discussing the conditioning state $C\left(x^{i-1}\right)$ or the conditional probability $P\left(x_i \left| C\left(x^{i-1}\right)\right.\right)$, we refer to x_i as the *current sample*.

When using higher-dimensional data (e.g., images), there is an order in which samples are considered (e.g. raster scan)[4] that makes the data appear as one-dimensional, nevertheless, the samples on which the conditioning states depend can be ordered differently, for example, using a bi-dimensional distance criterion (see Chapter 4 for the details on the criterion used in this thesis). In order to keep the notation clean, we continue using a one-dimensional notation but it can be easily extended through a remapping of the conditioning locations.

2.2.1 Fixed-length contiguous context models

For a non-negative integer K, a *K-th order fixed-length contiguous context model* is a causal context model in which $C\left(x^{i-1}\right) = x_{i-K}x_{i-K+1}\ldots x_{i-1}$.[5] In many applications, it makes sense to look for statistical dependencies in a close contiguous neighborhood and that is a reason why fixed-length contiguous context models are so popular. Nevertheless, the model description cost grows exponentially with the memory of the model, which makes fixed-length contiguous context models unable in practice to capture far dependencies.

[4] *Raster scan* is the method used in this work and it consists in traversing an image from top to bottom, from left to right.

[5] In lossless compression, since every sample must be encoded and thus modeled, when the conditioning state depends on samples with negative indexes (the *border* samples) some arbitrary conditioning state has to be chosen, with a rule that must be repeatable by the decompressor. These border samples determine the initial condition of the model, as discussed in Section 1.2. In our implementation, these locations get an initial arbitrary value of 0. Then, the first relevant (depending on K) samples of the sequence get reflected as soon as they are known. More precisely, when evaluating the sample x_i, $x_{-h} = x_h$ if $h < i$, otherwise, if $h \geq i$, $x_{-h} = 0$.

2.2.2 Tree models

A *tree model* (see, e.g., [Ris83, WRF95]) consists of a full α-ary tree (defined in Section 1.2) and a set of conditional probability distributions on the alphabet A, one associated with each leaf of the tree. Each leaf of the tree represents a state. Each edge of the tree is labeled with a symbol from A. Given an input sequence x^n, the state selected for the sample x_i is determined by descending from the root of the tree, matching the labels of the edges with the symbols in the sequence in reverse order, i.e., $x_{i-1}, x_{i-2}, \ldots$ until a leaf is reached. Therefore tree models are causal context models where $C\left(x^{i-1}\right) = x_{i-t}x_{i-t+1}\ldots x_{i-1}$ and t is determined by the conditioning state $x_{i-t}\ldots x_{i-1}$ itself.

In the example tree model of Figure 2.1, all subsequences ending with the symbol 0 select the same state, while for subsequences ending with the symbol 1, it may be necessary to examine one, or even two more past symbols of the subsequence in order to determine the state.

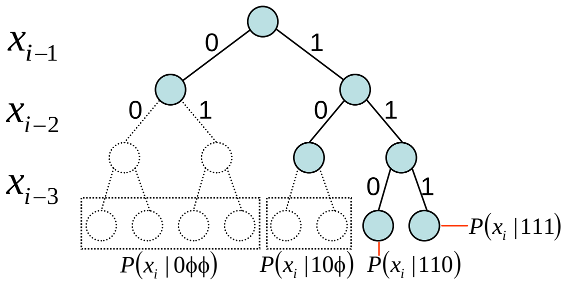


Figure 2.1: Example of a tree model defined over $A = \{0, 1\}$.

Notice that a full parametrization of a binary Markov source of order K corresponds to an underlying full balanced[6] tree of depth K.

This ability to reduce memory length has the potential to reduce the number of parameters and thus model cost. A major advantage of tree models is that statistical information needed to optimize the model can be stored in a context tree data structure, which is grown as the sequence is observed, recording essentially all the occurrences of each symbol in every context (including those represented by internal nodes). The precise manner in which this information is used in the model optimization varies from approach to approach, but the recursive combinatorial structure is key for algorithmic efficiency in all cases. Twice-universal coding schemes for this class of models attain a redundancy term $\frac{\alpha-1}{2}|S|\log n$, where $|S|$ is the number of leaves of the tree, since there are $\alpha - 1$ parameters per leaf.[7] Twice-universality can be achieved with the "plug-in" type of approach of the *Context Algorithm* [Ris83, WRF95], with the mixture approach of the *Context Tree Weighting (CTW)* algorithm [WST95, Wil98], or with the *two-pass* version of the Context Algorithm outlined in [Noh93]. Using the KT probability assignment, the latter two algorithms attain pointwise twice-universality and the normalized pointwise redundancy is at most ${}^{(d\log n)}\!/_{(2n)} + O\left({}^{d}\!/_{n}\right)$ where $d = (\alpha-1)|S|$ (see [MF98]), while the first one is shown to be twice-universal only when considering expected redundancy. In the two-pass approach (also called

[6]A tree T is full balanced if it is full and all its leaves are at the same depth.

[7]Notice that the set of tree models represented by unbalanced trees has measure zero in the space of Markov models of any order, otherwise, twice-universality would contradict Rissanen's lower bound.

semi-predictive), the best tree structure is estimated and described in a first pass, and then the data is sequentially encoded (based on this tree) in a second pass. This approach lacks sequentiality (since the full sequence must be seen in order to estimate the best tree structure) and is redundant in the sense that once the best tree is found, not all the input sequences x^n are possible and the algorithm ignores this fact. For these reasons, CTW is preferred in theory but, since the convergence rate differences with the two-pass approach are only of order $O\left({}^{d}\!/_{n}\right)$ and CTW has some practical issues,[8] the two-pass approach is sometimes preferred. In [MSW04], an efficient implementation of this approach is presented.

At the encoder side, the two-pass algorithm has the following steps:

1. First pass: gather all the context statistics for x^n in a context tree data structure.

2. *Prune* the context tree (lumping together equivalent states). In [Noh93], it is shown that, due to the (full) tree structure of the model class, pruning reduces essentially to a dynamic programming problem,[9] with the cost function given by the code length that each potential node in the context tree would contribute in case it were selected as a state. This problem can be solved in time that is linear in the number of nodes of the context tree. The context tree associates to each node the cost $L(s) + \frac{\alpha}{\alpha-1}$, s being the state represented by the node and L the code length achieved using the KT estimator and, then, the dynamic programming algorithm prunes the tree in order to find a subtree T with total minimum cost for the leaves. The total cost for T is: $L_T(x^n) + \frac{l_T\alpha}{\alpha-1}$, where l_T is the number of leaves in T.[10]

3. Second pass: describe T to the decoder using n_T bits (see Footnote 10) and encode x^n conditioned on T with KT.

The decoder, once it knows the structure of T, decodes x^n sequentially conditioned on the states of T using the KT estimator as well.

2.2.3 Sparse models

We now define the main objects of this investigation. A *relative location* l refers to the sample x_{i-l} when x_i is the current sample.

Definition 2.1. A *template* $\mathcal{T}$ is a set of relative locations $\{l_1, l_2, \ldots, l_k\}$ with $1 \leq l_1 < l_2 < \ldots < l_k \leq K$.

The elements of $\mathcal{T}$ define the conditioning states of sparse models as we see in the following definitions.

Definition 2.2. A *K-sparse context model (K-SCM)* with respect to a given template $\mathcal{T}$ is a causal context model where $C\left(x^{i-1}\right) = x_{i-l_k} x_{i-l_{t+1}} \cdots x_{i-l_1}$.

Therefore, the memory size of a K-SCM is fixed and equal to k.

[8]Because of finite precision, complexity and alphabet size restriction.

[9]Based on the observation that, by recursively assigning costs to sub-trees, an optimal tree consists of optimal sub-trees.

[10]Notice that $n_T = \frac{l_T\alpha-1}{\alpha-1}$, the total number of nodes in the tree, is also the cost of describing the full tree using a *natural* code (see, e.g., [WST95] and [Noh93]), with one bit per node that determine if each node is internal or not, assuming a preorder traversal. Therefore, minimizing $\frac{l_T\alpha}{\alpha-1}$ implies minimizing also the description cost of the tree.

Definition 2.3. A *K-window whole level sparse tree model (K-WLSTM)* with respect to a given template $\mathcal{T}$ is a causal context model where $C\left(x^{i-1}\right) = x_{i-l_t}x_{i-l_{t-1}}\dots x_{i-l_1}$ where $t \le k$ is determined by the samples of the conditioning state itself according to an underlying tree as in tree models[11] except that, when descending from the tree root, edge labels are matched with the symbols in the sequence selected by the template T in reverse order, i.e., $x_{i-l_1}x_{i-l_2}\dots$. until a leaf is reached.

Thus, K-WLSTMs can be considered as tree models whose dependencies can lie only among the finite set of relative locations given by $\mathcal{T}$. In particular, a K-WLSTM whose template is $\mathcal{T} = \{1, 2, \dots, k\}$ is a tree model with memory bounded by k. The maximum memory size of a K-WLSTM is bounded by the *weight* of the template $k = |\mathcal{T}|$ and is equal to the maximum j such that x_{i-l_j} appears in some context (i.e., the length of the longest branch in the underlying tree). Notice that every K-SCM can be represented, with the same number of states, by a K-WLSTM with a fixed $t = k$.

Additionally, we define the *actual memory window* as $\tilde{K} = \max_j (l_j)$. We write $C_{\mathcal{T}}\left(x^{i-1}\right) = x_{i-l_k}x_{i-l_{t+1}}\dots x_{i-l_1}$ to emphasize that the conditioning state is given by the template $\mathcal{T}$.

For a given template $\mathcal{T}$, the code given by the KT estimator is pointwise universal in the class of K-SCMs defined over $\mathcal{T}$ since the models in this class are conditioned on a finite set of states as the one considered in Section 2.1. Analogously, it can be shown[12] that, given a template $\mathcal{T}$, the code given by the two-pass approach[13] is twice-universal[14] in the class of K-WLSTMs defined over $\mathcal{T}$.

This thesis focuses in the following problem. We are given an input sequence x^n and a window size bound K. The ultimate goal is to find the best sparse model (K-WLSTM or K-SCM), for x^n. Since the number of different templates is 2^K an exhaustive search is not of practical interest for large values of K. In addition, no practical universal algorithms are known for these model classes for large K. Therefore, we look for approximate solutions, i.e., reasonably good models for the given input sequence. More precisely, we aim at finding model sequences in which each model gives a shorter code length than previous models. This sequence of models ends when no better models are found or some maximum computational time is reached. For this reason, we study heuristic methods like genetic and greedy algorithms.

Notice that, when variable length conditioning is allowed, a richer class of sparse models could be obtained if we allowed different orderings of template locations, but this significantly increases the problem complexity. In addition, intuitively, we assume that once a sparse set of locations is selected, locations closer to the modeled sample are considered "more relevant". Therefore, in this thesis, we do not consider other orders.

In Chapters 3, 6 and 7, we present and analyze some heuristic algorithms that are intended to address this problem. In Chapter 8, we empirically show that compression rates yielded by K-WLSTMs found by these algorithms are generally better in comparison to algorithms widely used in practice when the input data are binary images.

[11]Notice that, here, the underlying tree does not have ϕ labeled edges as in Section 1.2 (where we used a definition analogous to the one of STMs in order to show differences between the two classes) since holes are specified by the template.

[12]For example, the proof for [Mar09, eq. 3.5] extends for this case since there is no restriction on which context locations determine each state of the trees. Therefore, the states can be determined using sparse locations.

[13]The first step (statistics gathering) is performed considering the relative locations and the order given by $\mathcal{T}$.

[14]Notice that, in this case, the union of growing dimensional classes is finite since memory is bounded by k.

Chapter 3

Genetic algorithms for sparse template approximation

As mentioned before, the difficulty in optimizing sparse models comes from the combinatorial complexity of finding the best template to use to condition each sample. A starting point for this thesis was an implementation, proposed in [Ser04], of a genetic algorithm for finding approximate solutions to the template optimization problem for the K-SCMs defined in Definition 2.2. Next, we will present the paradigm of genetic algorithms in general and then its application to the template optimization problem.

3.1 Genetic algorithms in general

Genetic algorithms (GAs) are a metaheuristic[1] for finding approximate solutions to optimization problems. The method was introduced in [Bar54, Hol75] and is inspired on the natural selection mechanism described in Darwin's theory of evolution [Dar59]. In fact, GAs are a particular class of *evolutionary algorithms* (also known as *evolutionary computation*) that uses techniques inspired by evolutionary biology.[2]

In a GA, an *individual* is the representation of a candidate solution to an optimization problem and a *fitness function* allows to evaluate each individual: it corresponds to the optimization problem's objective function. The goal of a GA is to find the individual with optimum fitness. For instance, an individual, for the template optimization problem for K-SCMs, would be some template representation and its fitness would be the code length for the input sequence given by the KT estimator, described in Section 2.1, conditioned on this template. Finding the individual with optimal fitness is equivalent to finding the template minimizing code length for the input sequence, when using K-SCMs.

There are many optimization problems that can be proven to be hard (e.g., NP-hard [GJ79]) and for which no efficient (e.g., polynomial time) algorithms for finding such optimal solutions are known. In theses cases, the best one can aspire to is to find "good" solutions for which, in many cases, we cannot prove how close to optimal they are. However, we can often still evaluate the solution and find it more appropriate than other solutions to the problem. This kind of algorithm

[1]According to [Bla09], a metaheuristic is "a high-level algorithmic framework or approach that can be specialized to solve optimization problems".

[2]For a comprehensive tutorial on GAs see, for example, [Whi94].

aims at finding a sequence of increasingly better candidate solutions until solutions can no longer be improved or a prespecified time limit has been reached.

At each stage, GAs handle a set of M candidate solutions (a *population*) that contains the best solutions found up to this point. For a given population, individuals are combined and changed (through *crossovers* and *mutations*) with the goal of finding a new population in which better solutions show up (i.e., individuals with better fitness than the individuals of the previous population).

As mentioned before, GAs are a generic framework whose components can be adjusted to address many specific problems. Usually there are only two main components of GAs that are necessarily problem dependent, and they are:

- a *genetic representation* of the solution domain: GAs represent candidate solutions as individuals using linear binary representations called *chromosomes* or *genotypes.* The most standard one is a binary vector. Each vector location is called a *gene.* Since genetic operators work with the chromosomes' genes, the genetic representation together with the genetic operators play an important role in GAs.

- a fitness function to evaluate the solution domain. In opposition to the genetic representation, the fitness function is normally given as part of the problem description as the objective function.

The other components are, a priori, independent of the problem to be solved but can be tailored for better performance. Those usually are:

- a set (or population) of individuals. The cardinality of this set is called the *population size.* The population must be appropriately initialized, which is usually done in a random manner.

- a *recombination* mechanism needed to produce new populations of individuals (or *generations*). This generally comprises the following steps:

  - *selection*: individuals are chosen from the population as *parents*, to produce new individuals (*offspring*) through the crossover and mutation processes. There are several generic selection algorithms but the common idea is to choose the parents through a fitness-based process, where fitter individuals (as measured by the fitness function) are typically more likely to be selected.
  - *crossover*: parents are combined in some way to produce new individuals seeking that the offspring inherits good characteristics from its parents, and can become fitter than them. For example, in the *uniform* crossover scheme, in each recombination, from each pair of parents P_1 and P_2, two children H_1 and H_2 are generated. Then, for each vector location i, the following relationship holds:

$$(H_1(i), H_2(i)) = \begin{cases} (P_1(i), P_2(i)) \; \textit{with probability} \; \frac{1}{2} \\ (P_2(i), P_1(i)) \; \textit{with probability} \; \frac{1}{2} \end{cases}$$

  - *mutation*: after the crossover process, the new individuals undergo a mutation process in order to introduce some variations. Without mutation, the simulated evolution may converge to a very limited set of genes, which is very undesirable as it leaves the rest of the solution space unexplored. To improve the efficiency of a GA as a solver, it must explore the solution space as much as possible. On the other hand, a high mutation rate may lead to the destruction of good genes, and degrades the GA to some sort of random search. Thus, the mutation rate controls an important trade-off and is an important parameter of GAs. A common method of implementing the mutation operator involves generating a random variable for each bit in the vector to determine if that bit will be flipped or not.

- an *elitist* strategy that consists in copying the best m individuals (*survivors*) from the current population into the next one, without undergoing any modification. It is not required that $m > 0$, nevertheless, it is helpful in order to guarantee a strictly increasing fitness of the best individual of each generation and to avoid the loss of the best individuals.
- a termination criterion that determines when the generational process must be stopped. Common terminating criteria are:
  - a solution that satisfies a minimum criteria of goodness is found
  - some fixed number of generations is reached
  - some allocated budget (e.g., computation time) is reached
  - the highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results or improvement falls below a threshold

A basic generic GA would have the steps described in Algorithm 3.1.

Algorithm 3.1 A basic generic GA.

```
Choose an initial population of size $M$
Repeat until the termination criterion is reached:
  Recombine individuals from the current population in order to produce $M - m$
   individuals to be included in the next population
  Copy the $m$ best individuals from the current population into the next one
```

Since some steps of GAs are of a random nature, they require random numbers to be generated. These are normally obtained using *pseudo random number generators (PRNGs)* which are algorithms for generating sequences of numbers that approximate the properties of random numbers. These numbers are not truly random since they are completely determined by some initial *seed state* or *seed,* for short, which is an input for the algorithm. For more information on PRNGs, see, e.g., [Knu69, Lub96].

3.2 A genetic algorithm for the template optimization problem

We first present an example that illustrates why GAs are suitable for the sparse template optimization problem. Let us suppose that in a given sequence x^n, we gather the statistics described in Table 3.1.

If the two relative locations j and k were included in a K-SCM template, the model would assign a probability 1 to the sequence, which gives a zero code length if we ignore model cost.

$s = x_{i-j}, x_{i-k}$	$n_{x^n}(0 \mid s)$	$n_{x^n}(1 \mid s)$
00	m	0
01	0	m
10	0	m
11	m	0

(a)

$s = x_{i-j}$	$n_{x^n}(0 \mid s)$	$n_{x^n}(1 \mid s)$
0	m	m
1	m	m

(b)

$s = x_{i-k}$	$n_{x^n}(0 \mid s)$	$n_{x^n}(1 \mid s)$
0	m	m
1	m	m

(c)

Table 3.1: Example statistics of a sequence x^n showing how template locations work as "teams". $n = 4m$.

Nevertheless, if a K-SCM template includes only one of the locations individually, the resulting code length is $-4m \log(0.5) = n$, i.e., the sequence is not compressed. This example illustrates how locations work as "teams" and why the template optimization problem is a good candidate to be solved by GAs, since GAs proceed by considering sets of genes and combining them.

Using the generic framework of GAs, G. Seroussi [Ser04] implemented an heuristic solution to the sparse template optimization problem for the case of K-SCMs. We call this algorithm BRGTO (for Basic Randomized Genetic Template Optimization) in opposition to our enhanced version described in Chapter 7. BRGTO has the structure of Algorithm 3.1 with the following specific components:

- genetic representation: each individual is a binary vector I of K genes whose support represents a template of context locations. Each gene (bit) corresponds to a context location of the K-window. The location is included in the template if and only if, the corresponding bit is set to 1. Since I is often sparse, it is convenient to describe it as the set of vector locations whose corresponding value is 1, i.e.,

$$I = \{i_1, i_2, \ldots, i_k\} \Leftrightarrow I(i_1) = I(i_2) = \ldots = I(i_k) = 1.$$

- fitness function: it is the code length for the input sequence given by the K-SCM whose template is represented by the individual being evaluated.
- initial population: for each individual of the population of size M, randomly select with uniform distribution the weight w of the individual, then randomly select with uniform distribution w context locations to be included in the template represented by the individual.
- elitism: the best m individuals from the current population are copied into the next population
- recombination:
  - selection: $M-m$ pairs of individuals are chosen, with replacement, to play the role of parents for each recombination with the following method:
    - ∗ sort the current population according to the fitness of each individual and group individuals into M/B bins of size B. Thus, the best B individuals are grouped into bin 0, the next B individuals are grouped into the bin 1 and so on.

    * randomly select a bin using a geometric distribution with a finite support $[0 \ldots {}^{M}/_{B} - 1]$, the i-th bin having a probability of being selected $P(i) = c\gamma^i$, $\gamma < 1$, where c is a the normalization constant $c = \frac{1-\gamma}{1-\gamma^{M/B}}$.
    * randomly select with uniform distribution one individual of the selected bin.[3]
  - crossover: two individuals $I_1 = \left\{i_1^{(1)}, i_2^{(1)}, \ldots, i_{k_1}^{(1)}\right\}$ and $I_2 = \left\{i_1^{(2)}, i_2^{(2)}, \ldots, i_{k_2}^{(2)}\right\}$ produce one child $I_3 = \left\{i_1^{(3)}, i_2^{(3)}, \ldots, i_{k_3}^{(3)}\right\}$ where $k_3 = \left\lfloor \frac{k_1+k_2}{2} \right\rfloor$ and the locations $i_j^{(3)}$ are randomly chosen with uniform distribution from the union of the two sets I_1 and I_2 taken with multiplicities (a gene that appears in both parents has a higher chance of being selected). Thus, I_3 will tend to preserve common genes of I_1 and I_2. In particular, if $I_1 = I_2$, then $I_3 = I_1 = I_2$.
  - mutation: two types of mutations are considered, namely, *flips* (the bit value of a gene is flipped, which is equivalent to adding or deleting locations from the template) and *swaps* (the bit value of a gene is swapped with the value of a gene with different value, which is equivalent to substituting one template location with another one not already present in it). Notice that flips modify the weight of the template (and therefore the resulting model cost) while swaps preserve it. The *mutation rate* parameter μ_r controls the probability of both types of mutations and the *mutation mix* parameter μ_m controls the specific probability of each type of mutation. The algorithm for the mutation process is the following one:
    for each individual resulting from the crossover process, represented by a vector of bits I, and for each location j
    * with probability $\mu_r \mu_m$, where $0 \leq \mu_r \leq 1$ and $0 \leq \mu_m \leq 1$, $I(j) \leftarrow 1 - I(j)$.
    * independently of the previous event, with probability $\mu_r(1 - \mu_m)$, swap $I(j)$ and $I(k)$, where k is selected randomly with uniform distribution within the set of locations whose value is different from $I(j)$.

- termination criterion: a fixed number of generations G is reached

In order to avoid unnecessary computation, after each new population is produced, the algorithm identifies which individuals were present in the previous population and uses the already evaluated fitness value. Additionally some primary tuning of the parameters of the algorithm was done heuristically in [Ser04]. Nevertheless, the algorithm was still very time consuming. Additionally, a priori, this algorithm could be used for K-WLSTMs by replacing the fitness function with one that evaluates code length using K-WLSTMs instead of K-SCMs, but, in practice, the algorithm tends to evaluate high weight templates that imply building large trees that consume high amounts of memory and a lot of computation time, as explained in Chapter 5 and Section 6.3.

Therefore, in order to allow a faster experimentation with K-SCMs and to extend the experimentation to the setting of K-WLSTMs, in this thesis, we aimed at improving or adjusting Seroussi's implementation or at proposing some alternative heuristics. One way to improve the computing performance of BRGTO is to reduce the cost of the evaluation of each individual by carefully optimizing the software. This can be done as described in Chapter 5 and gives significant

[3]Therefore, if we consider the full sorted list of individuals, bins make the probability of selecting each individual have an exponential decay by stages. If B was chosen too small, the set of selected individuals for recombination would be concentrated on the very first individuals of the ranking and, thus, with high probability, we would exclude other individuals that are not the best ones but that, potentially, through recombination, could lead to better ones.

gains in execution time (a 160$\times$ speed-up in some cases) when comparing to the original implementation. In Chapter 6, we describe a greedy deterministic method for sparse template optimization which gives some good results with good time performance but that is far from optimal for some binary images. In Chapter 7, we propose some adjustments of BRGTO in order to emulate some of the desired characteristics of the greedy algorithm, with the purpose of getting a similar time performance while keeping the stochastic GAs properties as an aid against getting stuck in local minima.

Chapter 4

Binary images

Several of the experiments reported on in this thesis are performed on certain types of binary images that seem particularly amenable to modeling with sparse models. In this chapter, we define the structure of binary images and the context geometry that may be appropriate for modeling them. We describe some common types of images found in practice, and explain what makes them appropriate for sparse modeling. We also present some widely used binary image lossless compression methods with which we will compare our algorithms based on sparse models. We describe two methods for generating synthetic binary images, based on sparse models, which are useful for assessing the performance of our algorithms for sparse template approximation. Finally, we introduce a test set of binary images to be used in later chapters to assess the performance of our sparse model algorithms.

4.1 The context geometry

A *digital image* $x^{m \times n}$ is a two-dimensional array of m rows and n columns where each *pixel* (for "picture element") $x_{i,j} \in A$, where A is some alphabet. Pixels are the smallest individual elements in an image, holding quantized values that represent the brightness of the image at a specific point. A *binary* (or *bi-level*) *image* is a digital image that has only two possible values for each pixel ($|A| = 2$), usually representing black and white.

As mentioned in Section 2.2, by traversing an image from top to bottom and left to right, raster scan makes the array $x^{m \times n}$ appear as a one-dimensional sequence $\bar{x}^{mn}$ and, thus, each $x_{i,j}$ is mapped to $\bar{x}_{(i-1)n+j}$. Nevertheless, it is useful to capture two-dimensional (2D) spatial regularities and this requires 2D templates. In the same way that in the 1D case context locations are naturally ordered according to their distance to the conditioned sample x_i, we will define a total order for 2D contexts locations, according to their relative positions with respect to a conditioned sample $x_{i,j}$. Formally, a *2D relative location* is a vector (r, c) that refers to the sample $x_{i-r,j-c}$ when $x_{i,j}$ is the current sample.

Since lossless compression requires the models being causal, templates can include only past locations according to raster scan order, i.e., relative locations (r, c) such that

$$r < 0 \text{ or } (\text{r} = 0 \text{ and c} < 0).$$

We assume that some total order on this set of causal relative locations is given. For instance, in this work, we order the elements of this set according to their 2-norm (denoted as $\|\cdot\|$) and an arbitrary but fixed tie breaking rule as follows:

$$(r,c) < (r',c') \Leftrightarrow \begin{cases} \|(r,c)\| < \|(r',c')\| \text{ or} \\ (\|(r,c)\| = \|(r',c')\| \text{ and } |\mathrm{r}| < |\mathrm{r}'|) \text{ or} \\ (\|(r,c)\| = \|(r',c')\| \text{ and } |\mathrm{r}| = |\mathrm{r}'| \text{ and } \mathrm{c} < \mathrm{c}') \end{cases}$$

The set of the first $K = 32$ causal relative locations according to this order is shown graphically in Figure 4.1. Appendix B shows the causal relative locations and their order for K up to 1024. Here, as in the 1D case, other orders could be considered, but, in this thesis, we do not consider them for the same reasons explained for the 1D case.

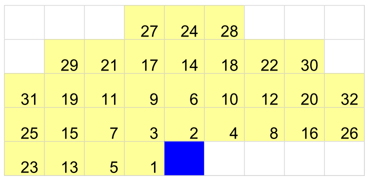


Figure 4.1: 2D relative locations with their corresponding order for $K = 32$. The dark rectangle represents the current sample.

Then, a *2D template* $\mathcal{T}$ is a set of 2D relative locations $\{l_1, l_2, \ldots, l_k\}$ where $l_1 < l_2 < \ldots < l_k$ (according to the given total order) and $l_h = (r_h, c_h)$ belongs to the set of the first K 2D relative locations. Context models using such templates have

$$C_{\mathcal{T}}(x^{m\times n}, i, j) = x_{i-r_1,j-c_1}, x_{i-r_2,j-c_2}, \ldots, x_{i-r_K,j-c_K}$$

which corresponds to

$$C_{\mathcal{T}}(\bar{x}^{mn}, i) = \bar{x}_{(i-r_1-1)n+(j-c_1)}, \bar{x}_{(i-r_2-1)n+(j-c_2)}, \ldots, \bar{x}_{(i-r_k-1)n+(j-c_k)}$$

in the one-dimensional notation.[1]

It is important to notice that a template that is contiguous in the 2D interpretation, in most of the cases, is not contiguous in the one-dimensional template mapped according to raster scan order.

4.2 Common types of binary images

In this section, we describe two important classes of binary images that benefit from sparse modeling: images of text or similar[2] documents and *halftoned* pictures.

Binary text images are normally obtained through a scanning device. They are usually the result of applying a *thresholding* operation on an image represented by a larger alphabet A' (e.g.,

[1]In our implementation, border samples are assigned analogously to the way described in Footnote 5 of Chapter 2: when $x_{i,j}$ gets known, then the relevant border samples are assigned in the following way: $x_{-i,j} := x_{-i,-j} := x_{-i,2C-j} := x_{i,-j} := x_{2R-i,j} := x_{2R-i,-j} := x_{2R-i,2C-j} := x_{i,2C-j} := x_{i,j}$ where R is the number of rows and C is the number of columns.

[2]For example, music scores.

gray scale images with $|A'| = 256$). The thresholding operation compares each pixel value of the original image against a fixed threshold, if the value is above the threshold then the output is 1, otherwise it is 0. The example of Figure 4.2 shows that the thresholding technique is quite appropriate for representing text documents as binary images.

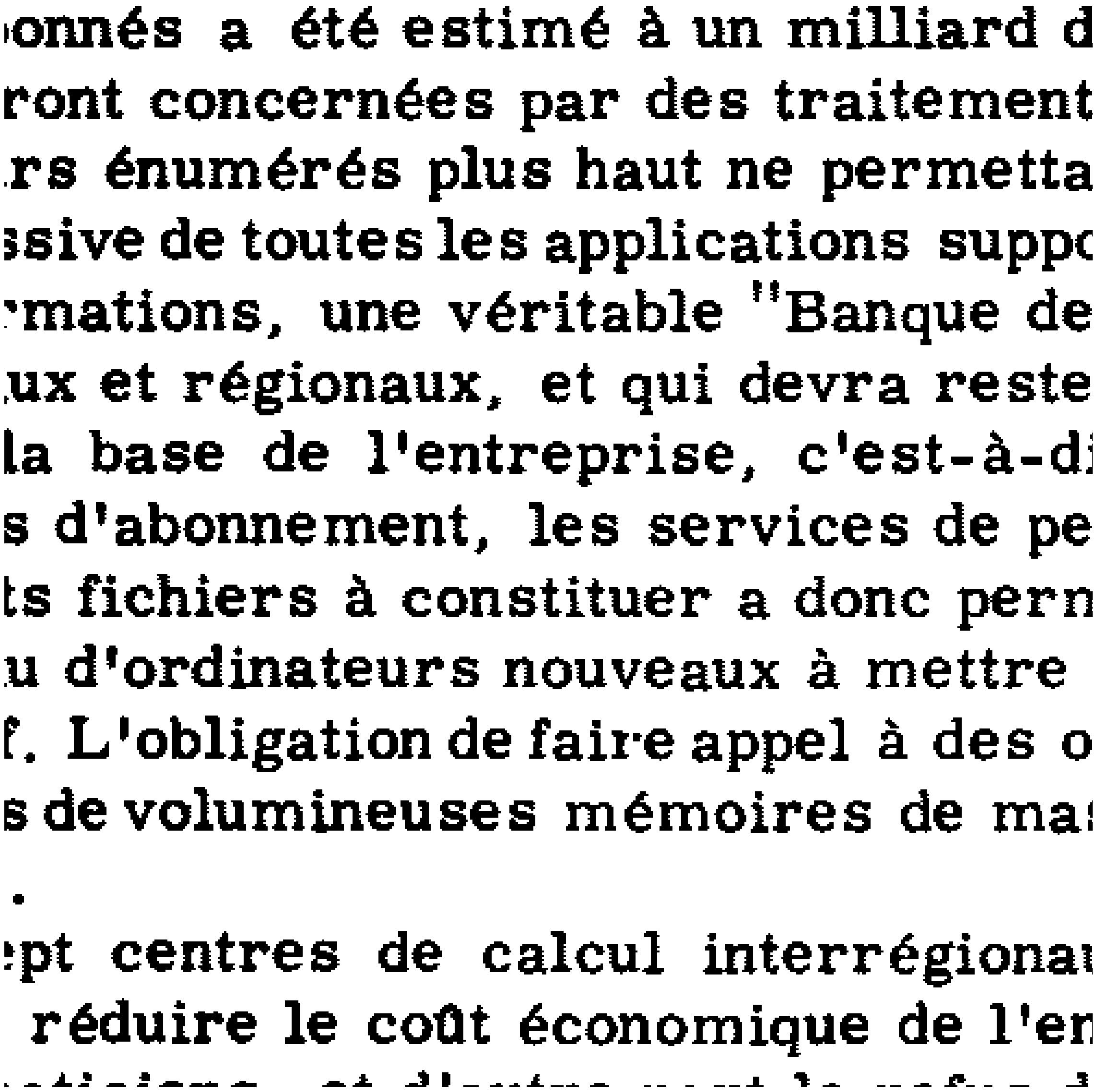

onnés a été estimé à un milliard d
ront concernées par des traitement
rs énumérés plus haut ne permetta
ssive de toutes les applications suppo
mations, une véritable "Banque de
ux et régionaux, et qui devra reste
la base de l'entreprise, c'est-à-di
s d'abonnement, les services de pe
ts fichiers à constituer a donc pern
u d'ordinateurs nouveaux à mettre
f. L'obligation de faire appel à des o
s de volumineuses mémoires de ma
.
pt centres de calcul interrégionau
réduire le coût économique de l'en
[illegible]

Figure 4.2: Portion of a text document binary image obtained through the thresholding operation.

For this class of images, there are intrinsic regularities (like repeated symbol shapes, distances between symbols) that could be efficiently (in terms of model cost) captured by sparse models.

When applied to pictures, the thresholding operation normally results in great loss of detail and contouring when comparing the resulting image to the original one (see Figure 4.3b). Halftoned pictures are obtained as the result of a *digital halftoning* operation on images represented by a larger alphabet. Digital halftoning is a technique for achieving satisfactory image rendering and color reduction (i.e., create the illusion of a large palette of colors using a much smaller set of colors). Initially, it was principally associated with the rendering of continuous-tone (gray-scale) images on binary video displays, which could only display full black or full white pixels, or on printers, which could produce only full black spots on a printed page. In [Uli87], digital halftoning is defined as "... any algorithmic process which creates the illusion of continuous-tone images from the judicious arrangement of binary picture elements."

Two important types of digital halftoning methods are described below (see Figure 4.3 for examples).[3]

- *ordered dithering*: given a fixed matrix $P^{v \times w}$ of values in A' (called *pattern*), an input image $I^{m \times n}$, the pixels of the binary output image $O^{m \times n}$ are obtained by the following formula:

$$O_{i,j} := \begin{cases} 1, \; if \; I_{i,j} > P_{i \mathrm{mod} v, j \mathrm{mod} w} \\ 0, \; \text{otherwise} \end{cases}$$

  This can be seen as an extension of the thresholding method, where the threshold is allowed to vary in a periodic way. Different patterns can generate completely different effects and artifacts. Two examples are the Halftone and Bayer's patterns (cases 4.3d and 4.3c in the figure).

- *error-diffusion dithering*: in this method, a single thresholding value is used for each pixel. The input image is scanned in some order (e.g., raster scan) and the value of each pixel of the input image is compared against a fixed threshold t. The corresponding output pixel is obtained as in the thresholding method. Then, before processing the next pixel, the difference between the input value and the threshold (the *quantization error*) is distributed and added ("diffused") to some of the input neighboring pixels not yet processed. For instance, in the Floyd-Steinberg dithering, the image is traversed in raster scan order and the quantization error $Q_{i,j}$ is distributed in the following way:

$$\begin{aligned} I_{i,j+1} &:= I_{i,j+1} + {}^{7}\!/_{16} Q_{i,j} \\ I_{i+1,j-1} &:= I_{i+1,j-1} + {}^{3}\!/_{16} Q_{i,j} \\ I_{i+1,j} &:= I_{i+1,j} + {}^{5}\!/_{16} Q_{i,j} \\ I_{i+1,j+1} &:= I_{i+1,j+1} + {}^{1}\!/_{16} Q_{i,j} \end{aligned}$$

  Error-diffusion dithering displays a very pleasing randomness, without the visual sensation of rows and columns of dots (see Figures 4.3e and 4.3f).

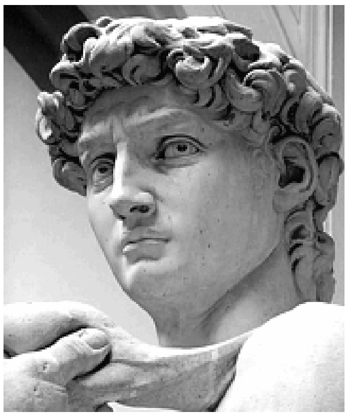
It is important to notice that these halftoning methods produce periodicities that could also be efficiently captured by sparse models. For example, in order to capture an horizontal period h and a vertical period v, the sparse template $\mathcal{T}(i,j) = \{x_{i-v,j-h}\}$ would be sufficient.

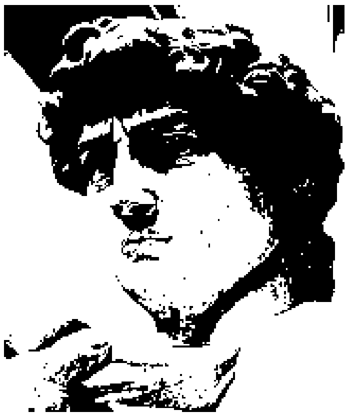
Section 4.5 introduces a set of images chosen to represent the different types of binary images selected for the experiments presented in the next chapters. In Chapter 8, we present compression results demonstrating the suitability of sparse models for these types of images.

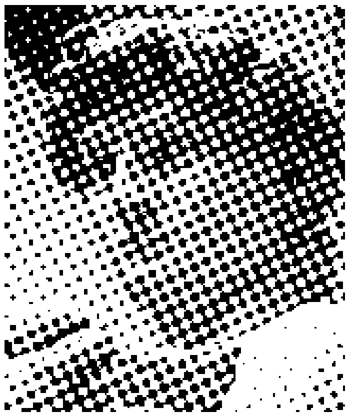
4.3 Standard compression methods for binary images

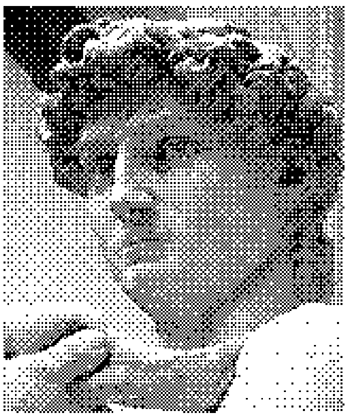
In this thesis, we compare the compression rates yielded by our algorithms based on sparse models against the results given by three widely used lossless compression methods for binary images: JBIG [Joi93], JBIG2 [Joi01] and DjVu [Liz05].

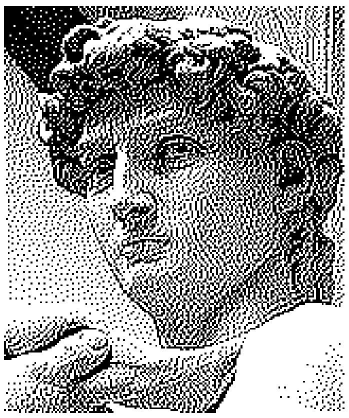
[3] Examples taken from http://en.wikipedia.org/wiki/Dither as of April 2009


(a) Original gray-scale picture


(b) Thresholding


(c) Ordered dithering: Halftone pattern


(d) Ordered dithering: Bayer's pattern


(e) Error-diffusion: Jarvis, Judice & Ninke algorithm

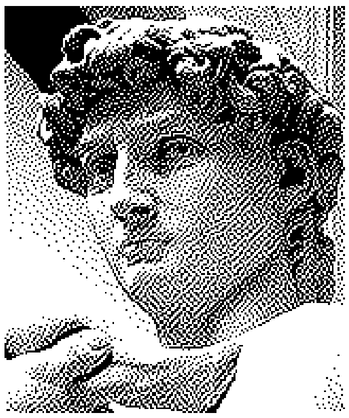

(f) Error-diffusion: Atkinson algorithm

Figure 4.3: Thresholding and dithering examples.

4.3.1 JBIG

JBIG is a standard that defines a lossless compression system essentially aimed at binary images. It has a variety of features, some of them being optional or allowed to be implemented with some freedom. It has two main modes called *progressive* and *sequential*. The progressive mode includes, in the encoded output, several versions of the input image in lower resolutions, in exchange for some overhead in code length. While this mode can be quite valuable for some applications, it is not of interest for our comparison purposes. On the other hand, in the sequential mode, the input image is read in raster scan order and encoded as a single image as we do in our work and, therefore, we focus on this mode.

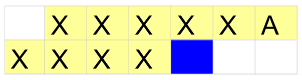
JBIG's sequential mode algorithm has the following steps:

1. Stripes division (optional): the input image can be broken into horizontal stripes that can be treated as separate images.

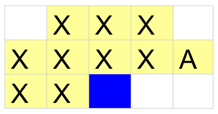
2. Typical prediction (optional): for each line of the input image, the algorithm determines if it is identical to the previous line. One bit in the output indicates the result of this comparison. If the line is identical to the previous one, it does not need to be encoded, otherwise it is encoded with steps 3 and 4.

3. Adaptive template modeling: each pixel of the lines to be encoded is modeled by conditioning its value on a template. Two kinds of templates can be used for this purpose (depicted in Figure 4.4), the three line template being the most used one. Both have nine fixed locations and an adaptive location whose default place is indicated with an "A" in Figure 4.4. The horizontal relative range in which adaptive locations can be moved is $[-M_x, M_x]$ from the current sample. The vertical relative range is $[-M_y, 0]$. M_x can be set up to a maximum of 127 and M_y up to a maximum of 255. These two parameters limit the range of values that can be used for a given binary image. The actual location of the adaptive location can be varied within a stripe of data and up to four moves per stripe are allowed. If M_x and M_y are set to zero for a given image, then the default location will be the only possibility for the adaptive location. The standard does not determine how to decide when to move the adaptive location.

4. Arithmetic coding: For each encoded pixel, its value and the context in which it occurs are passed to a binary tailored variant of arithmetic coder named QM-coder, which is a variation of the Q-coder [PMLJA88], patented by IBM. This encoder estimates the probability of each symbol given its context, based on previously seen symbols, and encodes based on this probability.



(a) Two line template



(b) Three line template

Figure 4.4: JBIG templates. The dark square is the current sample and the shaded squares are the template locations. Adaptive locations are marked with an "A" and fixed ones with an "X".

JBIG offers between 10% and 50% gains in compression rate over previous fax standards (i.e., CCITT Group 3 (Recommendation T.4) [Int96] and Group 4 (Recommendation T.6) [Int88]) for business-type documents (scanned images of line art and printed text) and, for halftone images, it offers gains in compression rates from 50% to 80% (see p. 471 of [BG+00] and references therein).

4.3.2 JBIG2

The JBIG2 standard adds a lossy compression mode and aims at improving lossless compression performance of JBIG. It allows binary images to be divided into three parts: *text regions*, *halftone regions*, and *generic regions*.

Text regions consist primarily of symbols (letters, numbers, etc.) that are relatively small, such as the standard font sizes used for document preparation. These symbols are aligned in either a left-right or top-bottom row format. For compressing text regions, a pattern matching technique can be used. This technique tries to recognize characters and encodes their shape only once in a *symbol dictionary* and then it encodes the coordinates where they appear. For lossy compression the difference between similar symbols (e.g., slightly different impressions of the same letter) can be neglected; for lossless compression, this difference is taken into account by compressing each symbol conditioned on its corresponding similar symbol in the dictionary. Therefore, this technique performs especially well when compressing high resolution text documents since symbols appear almost identically in each occurrence.

Halftone regions are identified when regularly occurring halftoning patterns are detected. These regions can also be compressed using a pattern matching technique which in some way creates a pseudo-grayscale image whose values lie in an alphabet of size 2^8. These values are indexes to a *pattern dictionary* that keeps a record of the different halftone patterns found in the region. Then, this gray scale image is compressed considering *bit-planes*, i.e. the eight binary images that result from considering each bit at a time of each pixel value of the pseudo-grayscale image. Then, these bit-planes are compressed as generic regions.

Generic regions consist of line drawings, large symbols, or other components that have not been identified or encoded as halftone or text regions. For instance, when no lossy or progressive compression is intended, halftone regions are normally not compressed as halftone regions but as generic ones. Generic region encoding is quite similar to JBIG's method. Nevertheless, JBIG2 allows larger templates that have up to 6 conditioning locations more, 4 of them being adaptive and allowed to be located away from the fixed template, in the same window defined in JBIG. It uses a form of arithmetic coding -another variant of the Q-coder- known as MQ-coder also patented by IBM.

In addition, the algorithm used for compressing generic regions is the basis of the representations for the symbol dictionary components, the pattern dictionary components, and the bit-planes of the pseudo-grayscale images used in the halftone regions. Generic region encoding also allows refinements to be applied to other regions. This permits quality progressive representations to be encoded, as in JBIG. JBIG2 is claimed to have lossless compression rates that are 30% better than JBIG (see, e.g., [SIH01]).

4.3.3 DjVu

DjVu is a system for encoding images (not only binary) with a broader scope than the previous standards. For compressing binary images, DjVu uses an algorithm named JB2 which is a variation of AT&T's original proposal to the JBIG2 standard (see, e.g., [BHH+98]) and has similar components.

4.3.4 Concluding remarks

We note that, besides the pattern matching technique especially designed for text regions in JBIG2 and DjVu, the three standard systems we described have at their core a sort of restricted K-SCM that can be slightly adapted thanks to the adaptive locations. In our approach, all the locations are adaptive and, in the case of the K-WLSTMs, even more adaptiveness is available thanks to variable length conditioning. In Chapter 8, we show that in many cases the additional flexibility provided by our approach is advantageous in comparison to the standard methods described.

4.4 Random binary image generation based on sparse models

For the purposes of experimentation and testing, it is sometimes advantageous to use synthetic images generated by sparse models. These sparse model generated images are used in this thesis for assessing the performance of our algorithms since, except for some pathological cases, with very high probability, the input template used in the generator is also the optimal template for the generated sequence. Consistency issues for sparse models and their estimators were studied in [FSV08, Fra08]. In this section, we describe two methods for generating random binary images based on sparse models and some input template: one is based on a randomly built K-SCM and the other is based on a K-WLSTM built by the statistics gathered from some given image.

4.4.1 Generation based on random K- SCMs

The procedure for random binary image generation using K-SCMs is described in Algorithm 4.1. We start from a given template, and choose the parameters of the model at random. Two random distributions are used and the corresponding PRNGs are initialized by different seeds. A *Beta distribution* is used for drawing the probability P_s of generating a symbol of value 1 for each occurring state s. The Beta distribution has two free parameters α and β and its probability function is defined on the domain $(0, 1)$ as

$$P(x) = \frac{(1-x)^{\beta-1}x^{\alpha-1}}{B(\alpha,\beta)}$$

where B is the *beta function* defined as

$$B(a,b) = \frac{(a-1)!\,(b-1)!}{(a+b-1)!}.$$

The parameters of the Beta distribution give us some control on the probabilities in each state and, thus, on the entropy of the process. For instance, if we set $\alpha = \beta = v$ for some value v, the higher the value v, the higher is the kurtosis[4] of the distribution and the higher tends to be the entropy of the resulting process. A uniform distribution is used for generating the border values of the image, i.e. the initial condition of the process. The samples of the image itself are drawn according to the probabilities P_s.

4.4.2 Generation based on trained K-WLSTMs

Another approach for building the sparse model used to generate the data is to build (or *train*) it using the statistics gathered on a given data sequence for a given template. This approach is not quite amenable for K-SCMs since it is common to find occurring states in the generated sequence that do not occur in the input sequence and for which statistics must be arbitrarily chosen. Nevertheless, in the case of a trained K-WLSTM based on a template $\mathcal{T}$, if a state $C_{\mathcal{T}}(x^n, i) = x_{i-l_t}x_{i-l_{t-1}}\dots x_{i-l_1}$ occurs in the generated sequence but, in the original sequence, it does not occur[5] or the total number of statistics for this state is below some given threshold τ, we can use the statistics of the state given by $x_{i-l_{t-1}}x_{i-l_{t-1}}\dots x_{i-l_1}$ (where t is determined by the state and the shape of the tree), i.e., the parent node in the tree. This climbing can be repeated until enough statistics are found. For the generated images used in this work, we used $\tau = 30$. The procedure for data generation using trained K-WLSTMs is described in Algorithm 4.2.

Figure 4.5 shows two examples of images generated by this algorithm and the corresponding original images used for the training. The input templates are the ones found by our algorithms for K-WLSTM approximation (presented in the next chapters) on the original images.

[4]The kurtosis of the Beta distribution is negative for $v < 1$ and positive for $v > 1$.

[5]This situation can occur because the full tree restriction (see Section 1.2) may force the pruning algorithm to include non occurring states in order to include other important states (in terms of data fitness).

Algorithm 4.1 Random binary image generation using a random K-SCM

```
Input: a template $\mathcal{T}$, two random seeds $s_1$ and $s_2$, beta distribution parameters
$\alpha$ and $\beta$
output: a random image $x^{m\times n}$
initialize_uniform($s_1$)
initialize_beta($\alpha$,$\beta$,$s_2$)
create an empty hash table T
For each location $(i,j)$ of the border
  $x_{i,j}$:=uniform()
For $i$:=1 to $m$
  For $j$:=1 to $n$
    $c$:=$C_{\mathcal{T}}(x^{m\times n},i,j)$
    If T($c$) is empty
      repeat
        T($c$):=Beta()
      until 0<T($c$)<1
    If uniform()$\leq$T($c$)
      $x_{i,j}$:=1
    Else
      $x_{i,j}$:=0
  End For
End For
Return $x_{1..m,1..n}$
```

4.4.3 Entropy estimation of the generating processes

In order to assess our algorithms in different cases of entropy, it is necessary to estimate the entropy of the generating processes. One way to evaluate the entropy of a process based on a sparse model requires the stationary distribution of the process to be evaluated, since the entropy of the process is the sum of the entropy in each state weighted by the stationary distribution (see, e.g., [CT06, Theorem 4.2.4]). The simple method for calculating the stationary distribution of a sparse model by representing it as a fixed-length contiguous context model of memory K has an exponential complexity in K. It is an open problem to find a more efficient way to calculate the stationary distribution of a process generated by a sparse model.

As a consequence of the Law of Large Numbers, the *asymptotic equipartition property* for a stationary ergodic[6] process (known as Shannon-McMillan-Breiman theorem [Sha48, McM53, Bre57]) states that $-\frac{1}{n}\log P\left(X_1^n\right) \to H\left(X_1^\infty\right)$ with probability 1. Thus, if the process is stationary and ergodic, the entropy can be estimated by the normalized ideal code length relative to the probability assignment given by the generating model.

A random process generated by a K-SCM is equivalent to a Markov chain of order K. A Markov chain is ergodic if it is possible to go from every state to every state with positive probability (not necessarily in one transition) (see, e.g., [Doo53] for precise definitions). In our K-SCM generated models, we guarantee that all conditioned probabilities are positive. Thus, starting from any state, any sequence of symbols (and, therefore, any state) can be generated with positive probability.

In order to be stationary, the state that determines the initial condition of the process must be drawn accordingly to the stationary distribution of the process (which is hard to compute in the

[6] A stochastic process is said to be stationary when $P(x_i^j) = P(x_{i+k}^{j+k})\ \forall i,j,k$ and ergodic when its time averages equal the ensemble averages; thus, when a random process has both properties, its statistical properties can be deduced from a single, sufficiently long realization of the process, see, for example, [Doo53] for a precise definition.

Algorithm 4.2 Random binary image generation using a trained K-WLSTM. $T(c)$.s represents the number of occurrences of the symbol s in the context c of the tree T. The tree pruning is performed as in the second step of the two-pass algorithm described in Subsection 2.2.2.

```
Input: a template $\mathcal{T} = \{l_1 l_2 \dots l_k\}$, a random seed $s_1$, a threshold $\tau$
output: a random image $x^{m \times n}$
initialize_uniform($s_1$)
gather statistics from $x^{m \times n}$ using $\mathcal{L}$ in a tree $T$
prune($T$)
For each location $(i,j)$ of the border
  $x_{i,j}$:=uniform()
For $i$:=1 to $m$
  For $j$:=1 to $n$
    $p$:=0
    Repeat
      $c$:=$x_{i-l_{t-p}} x_{i-l_{t-1}} \dots x_{i-l_1}$
      $p$:=$p+1$
    Until $T(c)$.0 + $T(c)$.1 $\geq \tau$
    If uniform()$\leq T(c)$
      $x_{i,j}$:=1
    Else
      $x_{i,j}$:=0
  End For
End For
Return $x_{1..m,1..n}$
```

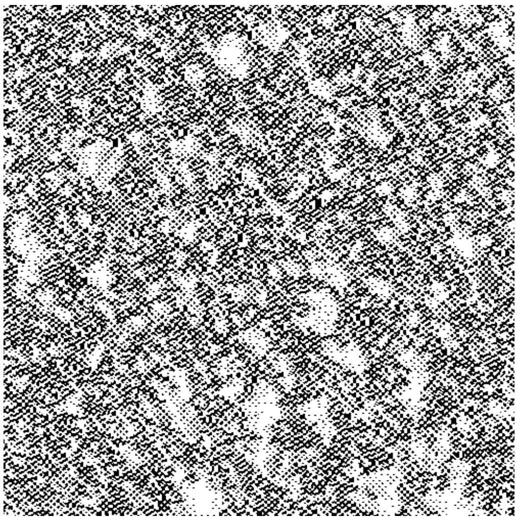
case of sparse models with large K, as previously mentioned). Nevertheless, in an ergodic Markov chain, the probability distribution of states converges rapidly to the stationary distribution (see, e.g., [Doo53]). Therefore, in order to estimate the entropy using the asymptotic equipartition property, an initial portion of the generated sequences should be discarded and the sequence must be large enough so that the Law of Large Numbers applies. For this reason, for the experiments of this thesis, only large generated images were considered.

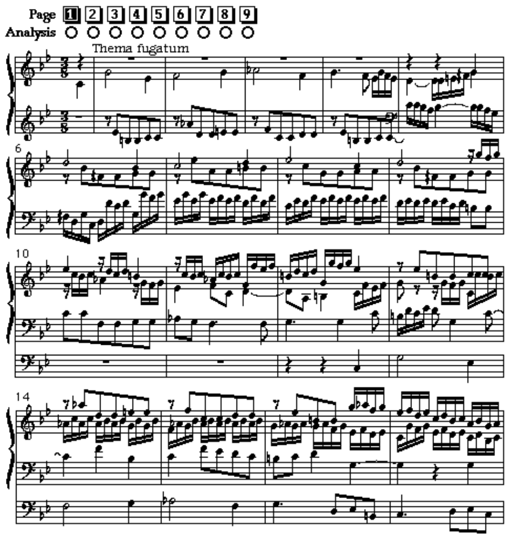
4.5 Test image set

Now, we introduce the images that we selected for assessing the performance of our sparse model algorithms presented in this thesis. The set comprises the following images that are shown in Appendix A:

- halftoned pictures with ordered dithering:
  - "albert2D" (Figure A.5)
  - "amb" (Figure A.6)
  - "Halftone2" (Figure A.15)
  - "Halftone3" (Figure A.16)
  - "HALFTONE" (Figure A.14)
- halftoned pictures with error-diffusion:
  - "lena_j" (Figure A.21)

(a) Portion of the original image


(b) Generated image



(c) Original image

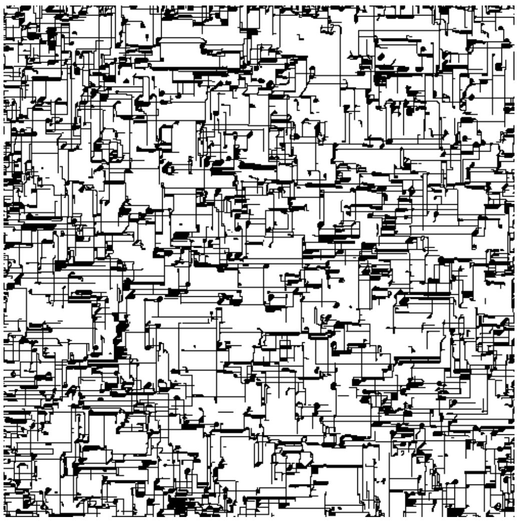

(d) Generated image

Figure 4.5: Examples of K-WLSTM generated images using a trained model.

  - “pep_j” (Figure A.24)

- machine-printed text or similar documents:
  - “A-fixedwidth6and8” (Figure A.4): a text document with two perfectly periodic zones showing one repeated character
  - “Bach_CPE-Sonata_flauto_solo_La_min-fl” (Figure A.7): a music score document
  - “cmfugue1-0” (Figure A.12): a music score document
  - “otoosfont12” (Figure A.22): a highly dense text document
  - “otoosfont24” (Figure A.23): a highly dense text document with double the resolution of “otoosfont12”
  - “ccitt4small” (Figure A.9): a lower resolution version of a text document from the reference set chosen by the International Telegraph and Telephone Consultative Committee

(CCITT, from the French name "Comité Consultatif International Téléphonique et Télégraphique") for evaluating proposals for binary image compression standards as JBIG, JBIG2 and previous fax standards.

  - "ccitt7small" (Figure A.10): a lower resolution version of a chinese text document also from the CCITT reference set.
  - "chinese_text" (Figure A.11): a chinese text document

- thresholded versions of texture[7] images:
  - "1.1.01M" (Figure A.1)
  - "1.1.13M" (Figure A.2)
  - "1.5.02M" (Figure A.3)
  - "flakes006-inca-100dpi-00M" (Figure A.13)
  - "texmos1.p512M" (Figure A.25)
  - "wallpaper003-inca-100dpi-00M" (Figure A.26)
  - "wallpaper004-inca-100dpi-00M" (Figure A.27)
  - "wallpaper010-inca-100dpi-00M" (Figure A.28)
- other thresholded images:
  - "Bobbys_letter_page_1" (Figure A.8): a hand-written text document
  - "leeleter" (Figure A.20): a hand-written text document
  - "hamilton_bw" (Figure A.17): a drawing
  - "hamilton_ed" (Figure A.18): the same drawing as "hamilton_bw" but obtained with a different threshold
  - "hieroglyph" (Figure A.19): a picture of hieroglyphs which has a regular layout of symbols as in text documents
  - "writing" (Figure A.29): a drawing

Additionally, some synthetic images based on trained K-WLSTM (whose templates are the best ones found by our algorithms on the training image) were included in the test set for an empirical consistency check, since they are cases where our K-WLSTM algorithms ought to perform significantly better than any other method. These images are:

- GEN_A-fixedwidth6and8 (Figure A.30): based on "A-fixedwidth6and8" (Figure A.4) with $K = 128$
- GEN_cmfugue (Figure A.31): based on "cmfugue1-0" (Figure A.12) with $K = 1024$
- GEN_otoosfont12 (Figure A.32): based on "otoosfont12" (Figure A.22) with $K = 512$

[7] Obtained from the Brodatz (http://www.ux.uis.no/~tranden/brodatz.html) and Oulu (http://www.outex.oulu.fi/index.php?page=image_database) texture databases.

Chapter 5

Computational issues in sparse model code length evaluation

The code length evaluation for sparse models is the most expensive step of the algorithms studied in this thesis. In this chapter, we first describe the computations involved in these evaluations and analyze their cost for both K-SCMs and K-WLSTMs defined in definitions 2.2 and 2.3. Then we describe some optimizations that yielded a significant reduction of the computation time for K-SCMs, and enabled the implementation of K-WLSTMs.

5.1 K-SCM evaluation

Given a template $\mathcal{T}$ and a sequence x^n, the code length given by the K-SCM implied by $\mathcal{T}$ on x^n is calculated in our implementation by the following steps:

1. Two hash tables $\mathcal{H}_K(x^n)$ and $\mathcal{H}_\mathcal{T}(x^n)$ are defined with the following structure $\mathcal{H}$:
   (a) the key is a sequence $c \in A^*$ representing an occurring context in x^n
   (b) the value associated to the key c, denoted as $\mathcal{H}[c]$, is an array of 2 non negative integers that count the number of times each symbol of the alphabet occurs in the context c in x^n
2. Full context statistics are gathered in $\mathcal{H}_K(x^n)$ as follows. For each sample x_i of the sequence:
   (a) extract its full context[1] of length K, i.e., $c := x_{i-K}, x_{i-K+1}, \ldots, x_{i-1}$
   (b) if the entry $\mathcal{H}_K(x^n)[c]$ does not exists, then create it and set $\mathcal{H}_K(x^n)[c] := (0,0)$
   (c) increase $\mathcal{H}_K(x^n)[c][x_i]$ by one
3. Sparse context statistics according to $\mathcal{T}$ are gathered in $\mathcal{H}_\mathcal{T}(x^n)$ as follows. For each key c in $\mathcal{H}_K(x^n)$:
   (a) extract the sparse context $c_\mathcal{T} := c\&\mathcal{T}$, where & denotes a bit masking operation resulting from taking the $|\mathcal{T}|$ bits from c in the positions selected by the template $\mathcal{T}$
   (b) if the entry $\mathcal{H}_\mathcal{T}(x^n)[c_\mathcal{T}]$ does not exists, then create it and set $\mathcal{H}_\mathcal{T}(x^n)[c_\mathcal{T}] := (0,0)$

[1]The border samples are initialized as described in Footnote 1 of Chapter 4.

(c) increase $\mathcal{H}_{\mathcal{T}}(x^n)[c_{\mathcal{T}}]$ by $\mathcal{H}_K(x^n)[c]$

4. Code length is calculated as follows. Set $L := 0$. For each key $c_{\mathcal{T}}$ in $\mathtt{H}_{\mathcal{T}}(x^n)$, increase L by the KT code length contribution of $c_{\mathcal{T}}$ using the following formula (see Section 2.1):

$$L_{KT}(x^n | c_{\mathcal{T}}) = -\log \frac{\Gamma(1)\prod_{a\in A}\Gamma(\mathcal{H}_{\mathcal{T}}(x^n)[c_{\mathcal{T}}][a] + 1/2)}{\Gamma\left(\sum_{a\in A}\mathcal{H}_{\mathcal{T}}(x^n)[c_{\mathcal{T}}][a] + 1\right)\Gamma\left(\frac{1}{2}\right)^2}$$

Notice that the first two steps are performed only once during the execution of any of the algorithms discussed in this work.

In order to get an overall evaluation cost, we need to add the cost implied by each step above in a weighted manner. For this purpose, we first derive an approximated formula for the evaluation time on any computer architecture:

- the iteration over $\mathcal{H}_K(x^n)$ and the searches and insertions in $\mathcal{H}_{\mathcal{T}}(x^n)$ take a time that is roughly[2] proportional to $|\mathcal{H}_K(x^n)|$, assuming that searches and insertions in the hash table are $O(1)$ on average (which is reasonable if the hash table and function are well designed and load balance is appropriately managed).

- the & operation takes a time proportional to $W(K)|\mathcal{H}_K(x^n)|$, where $W(K)$ is the number of computer words used to represent each context in the given computer architecture.

- the iteration over $\mathcal{H}_{\mathcal{T}}(x^n)$ and the evaluation of the KT code length contribution of each context take a time roughly[3] proportional to $|\mathcal{H}_{\mathcal{T}}(x^n)|$.

Therefore, the total execution time for the evaluation of the code length for x^n given by the K-SCM implied by $\mathcal{T}$ is, on average,

$$t_{K-SCM(\mathcal{T})}(x^n) \approx a_1|\mathcal{H}_K(x^n)| + a_2 W(K)|\mathcal{H}_K(x^n)| + a_3|\mathcal{H}_{\mathcal{T}}(x^n)|.$$

In order to be able to use the previous formula for comparing the evaluation cost of different models, we need to obtain values for the coefficients. For estimating these coefficients empirically, we used a linear regression. On the image "amb" (Figure A.6), for each K in $\{32, 64, 128, 256, 512, 1024\}$, and for each weight in $\{0..18\}$ (since the weight of the best templates found for this image using K-SCMs is around 18 and for heavier weights the high number of non-occurring states makes the linear approximation inaccurate, as mentioned before), 20 random templates were generated and their evaluation time was measured. Notice that this combination of parameters generates multiple distinct samples values for each variable of the formula and, therefore, the coefficients resulting from this linear regression apply to other images as well. The estimated coefficients and the correlation coefficient[4] are shown in Table 5.1.

Since the linear approximation turned out to be quite accurate, we can use these coefficients for comparing evaluation costs of experiment runs, independently of the specific computer used. Using

[2]It is an approximation because sometimes insertions are performed and sometimes a counter is increased and, also, because insertion time in the hash table is just guaranteed to be $O(1)$ in average.

[3]The approximation is because for templates with many locations there can be several states with non occurring symbols. For a non-occurring symbol a in a state s, the contribution to the total KT code length (see the formula in Section 2.1) is $-\log\Gamma(\mathcal{H}_{\mathcal{T}}(x^n)[c_{\mathcal{T}}][a] + 1/2) = \log_2\Gamma\left(\frac{1}{2}\right)$. Therefore, the evaluation of the function $\log_2\Gamma(\cdot)$ can be avoided using a precalculated value of $\log_2\Gamma\left(\frac{1}{2}\right)$.

[4]The correlation coefficient is an indicator of how well the equation resulting from the regression analysis explains the relationship among the variables. If it is 1, there is a perfect correlation in the sample — there is no difference between the estimated y-value and the actual y-value. At the other extreme, if the coefficient of determination is 0, the regression equation is not helpful in estimating a y-value.

a_1	a_2	a_3	error term	correlation coefficient
9.73E-5	3.99E-7	4.48E-3	-0.69	0.9976

Table 5.1: Coefficients for K-SCM evaluation time formula (in milliseconds) on an Intel Centrino Duo T2300 1,66 GHz processor running Windows XP. Notice that time is measured by quanta of approximately 15 ms in this kind of system.

the values of $|\mathcal{H}_K(x^n)|$ and $|\mathcal{H}_\mathcal{T}(x^n)|$ reported by the algorithms, we calculate the *evaluation cost for K-SCMs* as

$$c_{K-SCM(\mathcal{T})}(x^n) = a_1 |\mathcal{H}_K(x^n)| + a_2 W(K) |\mathcal{H}_K(x^n)| + a_3 |\mathcal{H}_\mathcal{T}(x^n)| .$$

5.2 K-WLSTM evaluation

Given a template $\mathcal{T}$ and a sequence x^n, the code length given by the K-WLSTM implied by $\mathcal{T}$ on x^n is calculated in our implementation by the following steps:

1. Same as steps 1 and 2 of the K-SCM evaluation.
2. Although the insertion of the extracted contexts could be done directly into the tree structure, they are first inserted in a hash table $\mathcal{H}_\mathcal{T}(x^n)$ as in step 1 of the K-SCM evaluation. This is to avoid visiting leaves each time the corresponding context occurs. For each entry of $\mathcal{H}_\mathcal{T}(x^n)$, insert into a binary tree structure $T_\mathcal{T}(x^n)$ the corresponding leave, its not yet inserted parent nodes and the necessary nodes to obey to the full tree structure restriction. Notice that the holes are not represented in this structure since they are determined by the template and, thus, the tree is just like one for a plain (contiguous) tree model.
3. Prune $T_\mathcal{T}(x^n)$ using the dynamic programming algorithm (see Subsection 2.2.2), obtaining a tree $T'(x^n)$.

As in the case of K-SCMs, the total cost will be a weighted sum of costs of the steps above, which we analyze next.

The tree building and pruning steps have a time complexity proportional to $|T_\mathcal{T}(x^n)|$ where $|T_\mathcal{T}(x^n)|$ is the number of nodes in $T_\mathcal{T}(x^n)$. Therefore, the total execution for the evaluation of the code length for x^n using the K-WLSTM implied by $\mathcal{T}(x^n)$ is, on average,

$$t_{K-WLSTM(\mathcal{T})}(x^n) \approx a_1 |\mathcal{H}_K(x^n)| + a_2 W(K) |\mathcal{H}_K(x^n)| + a_3' |\mathcal{H}_\mathcal{T}(x^n)| + a_4 |T_\mathcal{T}(x^n)| .$$

The first three terms are explained as in the K-SCM evaluation. Nevertheless, $a_3' \leq a_3$ since in the K-WLSTM evaluation case, the KT code length contributions are calculated during the tree pruning and thus this cost is not included in a_3' . Also in this case, the coefficients were estimated by a linear regression (on the same image but for weights up to 30, since the best templates found for K-WLSTMs on this image have weights around this value). The estimated coefficients and the correlation coefficient are shown in Table 5.2.

Therefore, we calculate the *evaluation cost for K-WLSTMs* as

$$c_{K-WLSTM(\mathcal{T})}(x^n) = a_1 |\mathcal{H}_K(x^n)| + a_2 W(K) |\mathcal{H}_K(x^n)| + a_4 |T_\mathcal{T}(x^n)| .$$

a_1	a_2	a'_3	a_4	error term	correlation coefficient
1.9E-4	4.37E-7	forced to 0	5.4E-3	-46.05	0.9916

Table 5.2: Coefficients for K-WLSTM evaluation time formula (in milliseconds) on an Intel Centrino Duo T2300 1,66 GHz processor running Windows XP. Notice that time is measured by quanta of approximately 15 ms in this kind of system. a'_3 is negligible since it only accounts for the iteration over $H_{\mathcal{T}}(x^n)$.

Thus, the difference in the evaluation time between K-STMs and K-WLSTMs is given essentially by the cost of building and pruning the corresponding tree, which is proportional to $|T_{\mathcal{T}}(x^n)| \geq |\mathcal{H}_{\mathcal{T}}(x^n)|$, because of the full tree structure restriction. Additionally, it is important to notice that for a given sequence x^n, the K-WLSTMs evaluated by the algorithms studied in this work tend to have larger $|\mathcal{H}_{\mathcal{T}}(x^n)|$ than for K-SCMs, since they tend to use templates with more locations.

5.3 Optimizations

The original framework of [Ser04] used the hash_map class of the C++ Standard Template Library (STL) of Microsoft Visual Studio 7 (2003). The first observation we made was that a substantial amount of time was involved in allocating and deallocating memory for building each $\mathcal{H}_{\mathcal{T}}$ table. Thus, in our implementation, we have a static closed hash table in order to reuse the same memory for each $\mathcal{H}_{\mathcal{T}}$. For this, a customized hash structure stores pointers to locations of an array in which counters are stored contiguously in order to speed up iterations over the tables. For the hash function $h(\cdot)$ (that maps a key to an integer $[0, s-1]$, where s is the size of the table), we chose P. Hsieh's SuperFastHash function [Hsi04] for its time performance and its key distribution equivalent to a uniform random map which is appropriate to minimize collisions. Collisions are handled by a quadratic probing method in which the i-th probe position for a key k is given by the function $h(k,i) = \left(h(k) + c_1 i + c_2 i^2\right) (mod\, s)$, where $c_2 \neq 0$. For a table size that is a power of 2, a good choice for the constants is $c_1 = c_2 = 1/2$, as the values $h(k,i)$ for i in $[0, s-1]$ are all distinct [Cor01, Problem 11.3]. This leads to a probe sequence of $h(k), h(k)+1, h(k)+3, h(k)+6, \ldots$ where the added values increase by $1, 2, 3, \ldots$ Quadratic probing avoids better the clustering problem that can occur with linear probing. In order to guarantee that there is enough space in each table and the load factor is kept below 0.5, we initially count the number of elements of $|\mathcal{H}_K|$ before building the table and then $\mathcal{H}_K$ is built of size$\lceil \log_2 (2|\mathcal{H}_K|) \rceil$. Then, the static structure for $\mathcal{H}_{\mathcal{T}}$ is built of the same size since $|\mathcal{H}_{\mathcal{T}}| \leq |\mathcal{H}_K|$.

Obviously, there is an initial overhead for building these structures but it is quite negligible if more than a few models are evaluated in the same run. The improvements of running time of the optimized version are significant (especially for heavier templates) as seen in the example shown in Table 5.3. The table shows an average speedup ranging from 1.3× in the (unrealistic) case of a template of weight 0 to 164.2× in the case of templates of weight 18, which are templates with only 3 locations more than the best found K-SCM template for this image. This improvement is also quite noticeable when evaluating K-WLSTMs since the first steps of the evaluation are the same and K-WLSTM algorithms tend to evaluate heavier templates than the ones for K-SCMs.

Template weight	STL version average time in ms	Optimized version average time in ms	Improvement Ratio
0	128.1	96.4	**1.3**
1	124.8	63.6	**2.0**
2	131.1	63.6	**2.1**
3	135.5	68.3	**2.0**
4	140.1	63.6	**2.2**
5	149.4	58.8	**2.5**
6	171.4	68.4	**2.5**
7	218.3	68.4	**3.2**
8	215.1	71.6	**3.0**
9	301.2	74.7	**4.0**
10	530.8	77.9	**6.8**
11	954.3	85.6	**11.1**
12	1184	91.6	**12.9**
13	2977.8	105.8	**28.1**
14	5676.1	123.3	**46.0**
15	3808.9	132.5	**28.7**
16	9252.7	148	**62.5**
17	7152.7	163.7	**43.7**
18	28890.3	175.9	**164.2**
19	23237.1	196.4	**118.3**
20	19913.6	205.8	**96.8**

Table 5.3: Running time comparison between STL and optimized hash versions on the image "cmfugue1-0" (Figure A.12) with $K = 512$. The same random templates were evaluated in each case (10 templates per weight). The initial overhead for building the structures in the optimized version is approximately of 300 ms, which is less than three average evaluations of templates of the optimal weight.

Chapter 6

Greedy algorithms for sparse template approximation

In order to propose an alternative algorithm for sparse template optimization, we explored the field of *greedy* algorithms. A greedy algorithm is any algorithm that follows the problem solving metaheuristic of making the locally optimum choice at each stage with the goal of finding the global optimum. It is important to have in mind that, although greedy algorithms find the globally optimal solution for some optimization problems, they are generally characterized as "short-sighted" and "non-recoverable" and thus they may find suboptimal solutions for some instances of other problems. This is because these algorithms, at early stages, can take some decisions that cannot be modified later and, although locally optimal, can lead to a solution not as good as the global optimum.

We first present a basic greedy algorithm that we call DITO (for Deterministic Incremental Template Optimization). This algorithm can be used in the setting of K-SCMs, in which case it is denoted as DITO$^{K-\text{SCM}}$, or in the setting of K-WLSTMs, in which case it is denoted as DITO$^{K-\text{WLSTM}}$. Then we present some variants to the algorithm that aim at providing some recoverability to it.

6.1 A basic greedy algorithm

When running BRGTO, we noticed that the earliest generations of the algorithm were the most expensive ones because of the heavy weight of the templates considered there. With that issue in mind, we designed DITO, a basic greedy algorithm, similar to the one used in [RSP08], for sparse template approximation. As described in Algorithm 6.1, DITO starts from an empty template (a memoryless model) and, in each stage, it tries to add every location not already included and keeps the one that gives the greatest improvement in code length, and repeats the step until no more improvements are possible. In the case of K-WLSTMs, templates are evaluated using the two-pass algorithm explained in Subsection 2.2.3. Then it is possible that, after pruning a tree, some location of the template gets no longer used. In this case, the location is removed from the template.

Since we have no other algorithm at hand for finding good sparse models against which the results of this greedy algorithm can be compared, we need alternative methods to assess its per-

Algorithm 6.1 DITO: a basic greedy algorithm for sparse template approximation. Templates are encoded as arrays of bits (the corresponding value is 1 when a location is included). flip(t,i) is a function that returns a new array that is a copy of t except for the i-th value that is flipped. codelength(t) evaluates the code length given by the template t for the input data, using K-SCMs or K-WLSTMs, depending on the case. weight(t) returns the number of locations included in the template t. When evaluating K-WLSTMs templates, after pruning a tree, some locations can get no longer used. In these cases, the locations are also removed from the templates for the next steps of the algorithm.

```
Input: a sequence $x^n$, a window size $K$
Output: a template $\mathcal{T}$

set tempBestTemplate:=0$^K$
set auxBestTemplate:=0$^K$
Repeat
  For each i such that tempBestTemplate[i]=0 //additions
    If codelength(flip(tempBestTemplate,i))<codelength(auxBestTemplate)
      set auxBestTemplate:=flip(tempBestTemplate,i)
  End For
  set tempBestTemplate:=auxBestTemplate
While improvements are achieved and weight(tempBestTemplate)<$K$
Return $\mathcal{T}$:=tempBestTemplate
```

formance. For this purpose, we evaluate it on images for which we have good indications of what the result should be.

6.1.1 Compression performance on images generated by K-SCMs

As explained in Section 4.4, one way to evaluate the compression performance of this greedy algorithm is to run it on data generated by sparse models, since we know the templates that are used for the data generation, which are, reasonably, good candidates to be the optimal ones or close to. Thus, we compare the code length given by the templates found by DITO$^{K-\mathrm{SCM}}$ on K-SCM generated images against the code length given by the templates of the generating models.

Some images were generated using K-SCMs, with $K = 32$, by combining the following parameter values:

- Templates: 3 random templates of weight 13, 14 and 15, whose binary array representations are, respectively, 00001111110001111001101100000000, 11110110000001111010101000000100 and 01010010101101001010010100101011 (the rightmost bit represents the first relative location and the other bits to the left represent the other relative locations in the order depicted in Figure 4.1).

- For the Beta distribution: $\alpha = \beta,\ \alpha \in \{0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8, 25.6\}$

The DITO$^{K-\mathrm{SCM}}$ algorithm was run on each generated image. We classify the generated images according to their estimated entropy (see Subsection 4.4.3). Table 6.1 compares the code length given by the template found by DITO$^{K-\mathrm{SCM}}$ with the code length given by the generating template for each entropy level. Notice that the difference between the estimated entropy and the code length given by the generating template is due to model cost (i.e., the “learning” cost of the KT estimator) included in the former but not in the latter. We ignore some combinations of parameters that produce high entropy images that cannot be compressed with the generating template (i.e., the compression ratio is greater or equal than 1).

Estimated entropy	$L_{\mathcal{T}}/n$	L_{DITO}/n	$(L_{\text{DITO}}-L_{\mathcal{T}})/L_{\mathcal{T}}$
0.1907	0.2686	0.2686	0%
0.1959	0.3085	0.3085	0%
0.1965	0.2530	0.2530	0%
0.3260	0.3747	0.3747	0%
0.3300	0.4299	0.4299	0%
0.3362	0.4046	0.4046	0%
0.4981	0.5847	0.5847	0%
0.5006	0.5589	0.5589	0%
0.5009	0.5408	0.5408	0%
0.6703	0.7040	0.7040	0%
0.6713	0.7476	0.7476	0%
0.6732	0.7241	0.7241	0%
0.8048	0.8345	0.8345	0%
0.8058	0.8769	0.8769	0%
0.8085	0.8548	0.8548	0%
0.8949	0.9222	0.9222	0%
0.8964	0.9658	1.0000	+3.54%
0.8973	0.9415	0.9415	0%
0.9457	0.9719	0.9719	0%
0.9467	0.9897	1.0000	+1.04%
0.9720	0.9978	1.0000	+0.22%

Table 6.1: DITO$^{K-\text{SCM}}$ results on K-SCM generated images of different entropy levels. $L_{\mathcal{T}}$ is the code length given by the generating template $\mathcal{T}$, L_{DITO} is the code length given by DITO$^{K-\text{SCM}}$ and n is the number of samples of the image.

We observe that DITO$^{K-\text{SCM}}$ generally performs very well on this kind of images for different entropy levels.

6.1.2 Compression performance on non-synthetic images

Another way to evaluate the compression performance of this algorithm is to run it for many K values on the same data: if the algorithm is robust, the results should not worsen when the search space is augmented. On the image "albert2D" (Figure A.5), DITO$^{K-\text{SCM}}$ gives a solution whose code length for $K = 512$ is 9.28% worse than for $K = 256$, which is quite significant. On the image "A-fixedwidth6and8" (Figure A.4), DITO$^{K-\text{WLSTM}}$ gives a solution whose code length for $K = 512$ is 45.62% worse than for $K = 128$.

It is important to remark that "albert2D" has a highly regular grid layout of halftoning patterns with a well defined horizontal and vertical period p (in a loose sense). By increasing K, the algorithm faces new choices of locations that capture greater periods (multiples of p) which can significantly change the sequence of decisions it takes. In fact, when we observe the template found for $K = 256$ (Figure 6.1a), we find that it includes several locations that are within a distance smaller than p. This can be useful for modeling the halftoning patterns themselves. While, for $K = 512$ (Figure 6.1b), most of these locations are not included and the weight of the template is much lower (12 vs. 19). Therefore, in the second case, the algorithm mostly captures many periods of the halftoning grid instead of the regularities inside the halftoning patterns themselves and therefore does not capture so well the overall structure of the image.

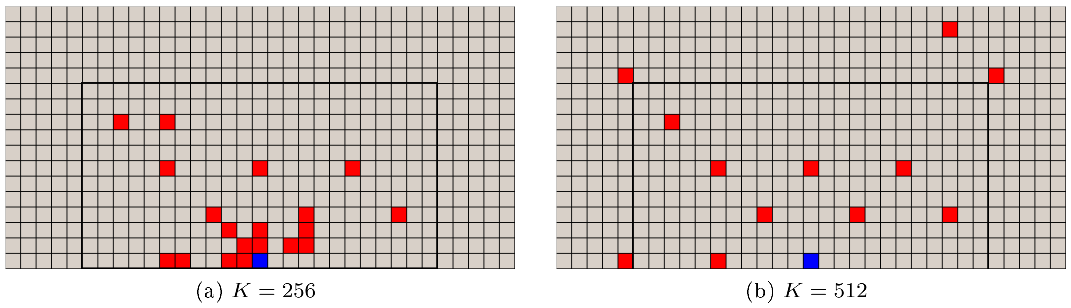

(a) $K = 256$ (b) $K = 512$

Figure 6.1: K-SCM templates found by DITO$^{K-\text{SCM}}$ on "albert2D". The darkest square is the current sample and the thick line shows the window for $K = 256$.

Something similar seems to happen for "A-fixedwidth6and8" (see templates in Figure 6.2), where for the biggest value of K, the algorithm mostly captures the distances (and its multiples) between letters instead of the regularities of the characters themselves.

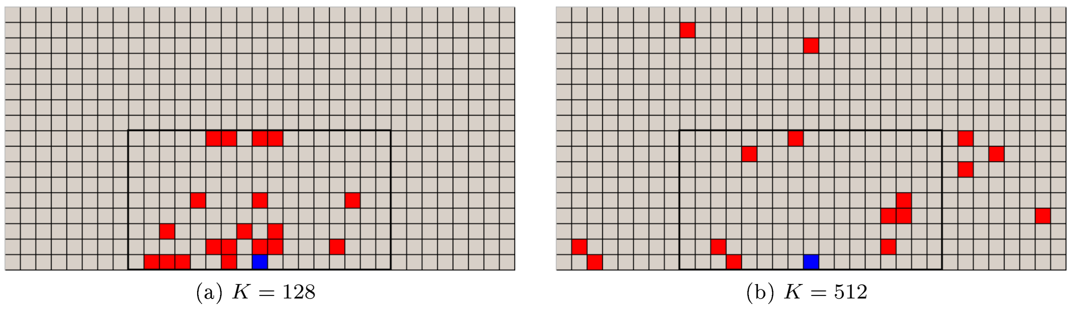

(a) $K = 128$ (b) $K = 512$

Figure 6.2: K-WLSTM templates found by DITO$^{K-\text{WLSTM}}$ on "A-fixedwidth6and8". The darkest square is the current sample and the thick line shows the window for $K = 128$.

6.2 Greedy algorithms with corrections

Although DITO$^{K-\text{SCM}}$ seems to perform well for K-SCM generated images with randomly generated distributions, we found some images that clearly show that both DITO$^{K-\text{SCM}}$ and DITO$^{K-\text{WLSTM}}$ can get stuck in some really bad local optima. With the goal of improving the results, we tried some variants that aim at giving some recoverability to the algorithm.

6.2.1 Deletion of template locations

A first extra step is added to DITO: after considering all the template locations to add and eventually adding one, the algorithm now considers deleting one of the locations already added, as described in Algorithm 6.2 which we call DITO$_D$.

This extra step has low cost and we observe that, in some cases, it gives important improvements over the results given by DITO. For example, for $K = 256$ on "albert2D" DITO$_D^{K-\text{SCM}}$ gives gains in code length of 5.7% over DITO$^{K-\text{SCM}}$. Nevertheless, DITO$_D$ fails considerably on the same cases where DITO does as shown in Tables 6.2 and 6.3, with losses even bigger. Although deletions

Algorithm 6.2 DITO$_D$.

```
Input: a sequence $x^n$, a window size $K$
Output: a template $\mathcal{T}$

set tempBestTemplate:=$0^K$
set auxBestTemplate:=$0^K$
Repeat
  For each i such that tempBestTemplate[i]=1 //deletions
    If codelength(flip(tempBestTemplate,i))<codelength(auxBestTemplate)
      set auxBestTemplate:=flip(tempBestTemplate,i)
  End For
  set tempBestTemplate:=auxBestTemplate

  For each i such that tempBestTemplate[i]=0 //additions
    If codelength(flip(tempBestTemplate,i))<codelength(auxBestTemplate)
      set auxBestTemplate:=flip(tempBestTemplate,i)
  End For
  set tempBestTemplate:=auxBestTemplate
While improvements are achieved and weight(tempBestTemplate)<$K$
Return $\mathcal{T}$:=tempBestTemplate
```

can provide some correction ability to the algorithm, the algorithm still considers one location at a time and no real backtracking is allowed.

greedy algorithm	512 vs. 256
DITO^{K-SCM}	+9.28%
DITO$_D^{K-SCM}$	+15.91%

Table 6.2: Difference in code length given by DITO$^{K-\text{SCM}}$ and DITO$_D^{K-\text{SCM}}$ on "albert2D" when increasing window size. Differences are calculated as $(L_{\text{DITO}}^{K=512}-L_{\text{DITO}}^{K=256})/L_{\text{DITO}}^{K=256}$, expressed as a percentage. Positive numbers represent losses in compression rate, and larger magnitude numbers represent larger losses.

greedy algorithm	256 vs. 128	512 vs. 128
DITO$^{K-\text{WLSTM}}$	+25.85%	+45.62%
DITO$_D^{K-\text{WLSTM}}$	+38.39%	+52.85%

Table 6.3: Difference in code length given by DITO$^{K-\text{WLSTM}}$ and DITO$_D^{K-\text{WLSTM}}$ on "A-fixedwidth6and8" when increasing window size. Differences are calculated as $(L_{\text{DITO}}^{2K}-L_{\text{DITO}}^{K})/L_{\text{DITO}}^{K}$, expressed as a percentage. Positive numbers represent losses in compression rate, and larger magnitude numbers represent larger losses.

6.2.2 Substitution of template locations

Another extra step is added to the algorithm: between the additions and deletions steps, the algorithm consider substituting each of the locations already included in the template with the ones not already included. This substitution step is repeated until no more improvements can be achieved with substitutions, as shown in Algorithm 6.3 which we call DITO$_{DS}$. This step may provide an additional correction ability, in which locations are considered by pairs in contrast with the additions and deletions steps in which locations are considered individually.

Algorithm 6.3 DITO$_{DS}$. swap(t,i,j) returns a copy of t with the i-th and j-th values swapped.

```
Input: a sequence $x^n$, a window size $K$
Output: a template $\mathcal{T}$

set tempBestTemplate:=0$^K$
set auxBestTemplate:=0$^K$
Repeat
  For each i such that tempBestTemplate[i]=1 //deletions
    If codelength(flip(tempBestTemplate,i))<codelength(auxBestTemplate)
      set auxBestTemplate:=flip(tempBestTemplate,i)
  End For
  set tempBestTemplate:=auxBestTemplate

  For each i such that tempBestTemplate[i]=0 //additions
    If codelength(flip(tempBestTemplate,i))<codelength(auxBestTemplate)
      set auxBestTemplate:=flip(tempBestTemplate,i)
  End For
  set tempBestTemplate:=auxBestTemplate

  Repeat //substitutions
   For each i such that tempBestTemplate[i]=1
    For each j such that tempBestTemplate[i]=0
     If codelength(swap(tempBestTemplate,i,j))<
        codelength(auxBestTemplate)
        set auxBestTemplate:=swap(tempBestTemplate,i,j)
    End For
   End For
   set tempBestTemplate:=auxBestTemplate
  While substitutions keep achieving improvements

While improvements are achieved and weight(tempBestTemplate)<$K$
Return $\mathcal{T}$:=tempBestTemplate
```

In the simpler case where this step is executed only once between additions and deletions, the cost that this extra step adds to the algorithm is $O\left(w^2K\right)$, where w is the number of locations in the template when the algorithm stops. This step is not that practical in the K-WLSTM setting because of the high cost of evaluating these models (detailed in Chapter 5) and usually a higher w than for K-SCMs. Thus, we focus on the K-SCM setting. Comparing DITO$_D$ and DITO$_{DS}$ on all the non-synthetic images[1] of the set described in Appendix A for K in $\{32, 64, 128, 256, 512, 1024\}$, we find that the greatest gain that the substitution step has achieved was only of 1.36% and in some cases there was a loss, the maximum being of 0.92%. The robustness problems still remain with DITO$_{DS}^{K-\mathrm{SCM}}$ since the loss in code length with $K = 512$ vs. $K = 256$ on "albert2D" is of 15.65%. In conclusion, it seems that the substitution step is not really worth its high price.

This shows that both corrections steps (deletions and substitutions) are not enough for avoiding falling in some bad local minima for some kind of data and suggests that a more robust heuristic ought to consider less restricted kinds of corrections like, for example, the ones performed by a genetic algorithm's crossover and mutation steps.

[1]Except "A-fixedwidth6and8".

6.3 Cost of model evaluation: greedy vs. random templates

In this section, we analyze why the DITO algorithms tend to be much faster than BRGTO. This analysis gave us important ideas about how to improve BRGTO and make it amenable to use in a K-WLSTM setting. These ideas were used to design the enhanced version of BRGTO presented in Chapter 7.

In order to analyze how expensive is the evaluation of the models considered by the greedy algorithm, we need to recall the formula, derived in Chapter 5, that gives the code length evaluation cost (defined in the same chapter). Given a template $\mathcal{T}$ and a sequence x^n, the evaluation cost for x^n using a K-SCM m based on $\mathcal{T}$ is:

$$c_{K-SCM(\mathcal{T})}\left(x^n\right) = a_1 \left|\mathcal{H}_K\left(x^n\right)\right| + a_2 W\left(K\right) \left|\mathcal{H}_K\left(x^n\right)\right| + a_3 \left|\mathcal{H}_{\mathcal{T}}\left(x^n\right)\right|$$

where $\left|\mathcal{H}_K\left(x^n\right)\right|$ is the number of distinct contiguous K-tuples occurring in x^n, $\left|\mathcal{H}_{\mathcal{T}}\left(x^n\right)\right|$ is the number of states of m occurring in x^n and $W\left(K\right)$ is the number of computer words used to represent a context of memory K.

Given a template $\mathcal{T}$ and a sequence x^n, the evaluation cost for x^n using the K-WLSTM implied by $\mathcal{T}$ is:

$$c_{K-WLSTM(\mathcal{T})}\left(x^n\right) = a_1 \left|\mathcal{H}_K\left(x^n\right)\right| + a_2 W\left(K\right) \left|\mathcal{H}_K\left(x^n\right)\right| + a_4 \left|T_{\mathcal{T}}\left(x^n\right)\right|$$

where $\left|T_{\mathcal{T}}\left(x^n\right)\right|$ is the number of nodes in the tree structure before pruning.

Besides the improvements described in Section 5.3, DITO algorithms have two characteristics that make it efficient in the use of computational resources:

1. the weight of the evaluated templates starts from 0 and then successively increases. The evaluation of lighter templates tends to reduce $\left|\mathcal{H}_{\mathcal{T}}\left(x^n\right)\right|$ and $\left|T_{\mathcal{T}}\left(x^n\right)\right|$, although it depends on the data and the templates considered.[2]

2. they evaluate templates that are slight variations from the best of each stage, which makes them likely to be good ones. The relationship here is less direct, but it is explained by the fact that the KT estimator favors models with less occurring states in the given data unless the extra states give enough improvement in fitting the data to compensate the extra model cost. Thus, the evaluation of better models tends to reduce $\left|\mathcal{H}_{\mathcal{T}}\left(x^n\right)\right|$ and $\left|T_{\mathcal{T}}\left(x^n\right)\right|$ as well.

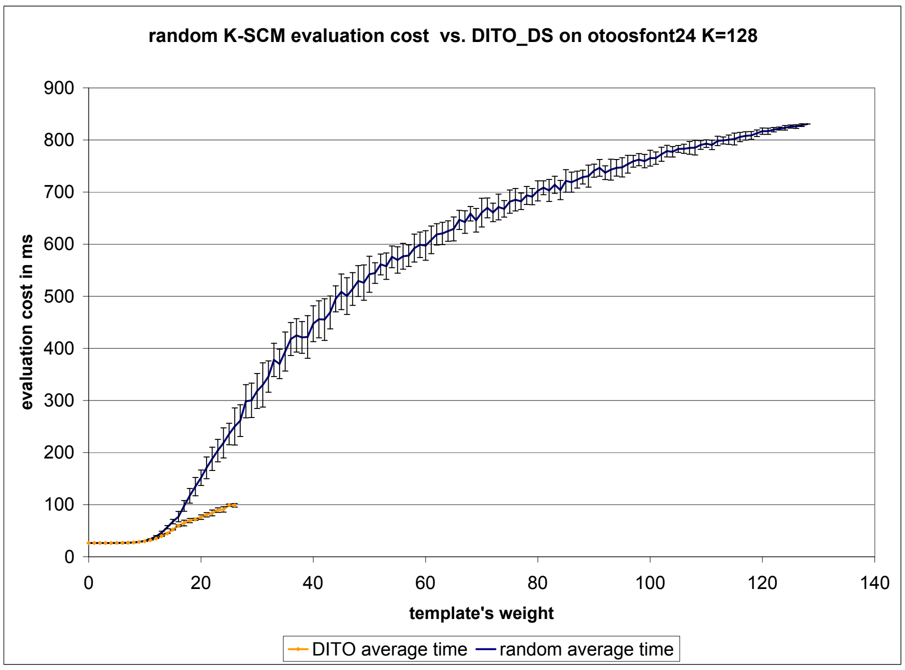
Figures 6.3 and 6.4 illustrate, using an example input image (with $K = 128$ for K-SCMs and $K = 1024$ for K-WLSTMs), these observations by comparing the results found at each stage by DITO (DITO$_{DS}$ for K-SCMs and DITO$_D$ for K-WLSTMs) against random templates. In the case of random templates, for K-SCMs, we consider templates of weight up to the window size since, in BRGTO, in the initial population, templates' weights are uniformly distributed in $1..K$ and, thus, templates of any weight can be evaluated. Figures 6.3a and 6.4a show, for K-SCMs and K-WLSTMs respectively, how evaluation cost increases with weight, as stated in the first observation. Nevertheless, we observe that in the case of random templates, the cost increases faster than for the templates found by DITO$_{DS}$. This explained, by the second observation, which is illustrated in Figures 6.3b and 6.4b, in which we observe a significant difference in code length between random templates and those found by DITO$_{DS}$, especially in the case of higher weights.

[2]Except when a template $\mathcal{T}$ represents a subset of the set represented by a template $\mathcal{T}'$. In this case, it is true that $\left|\mathcal{H}_{\mathcal{T}}\left(x^n\right)\right| \leq \left|\mathcal{H}_{\mathcal{T}'}\left(x^n\right)\right|$ and $\left|T_{\mathcal{T}}\left(x^n\right)\right| \leq \left|T_{\mathcal{T}'}\left(x^n\right)\right|$.

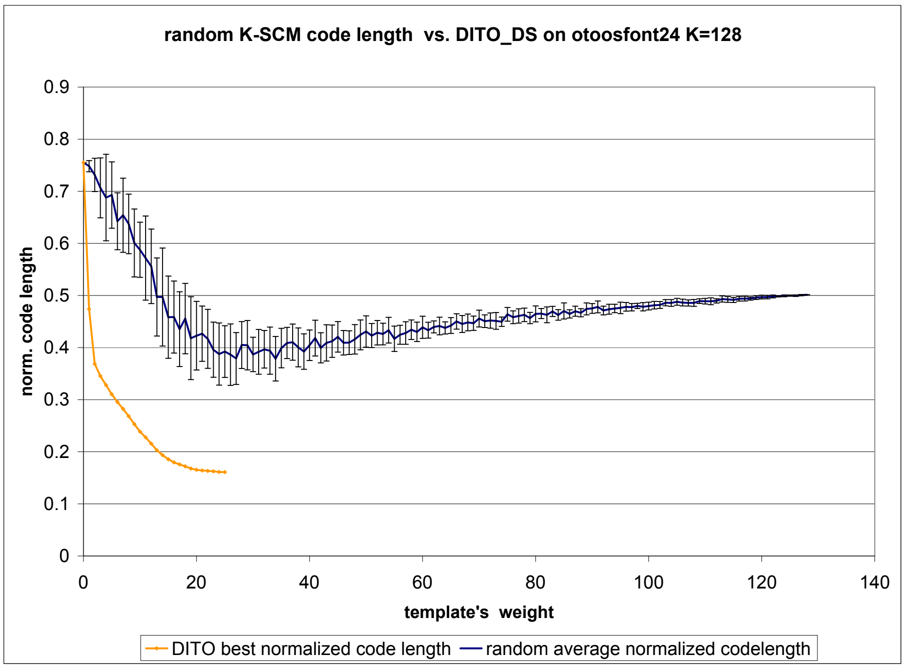
Another important issue when evaluating K-WLSTMs is the amount of memory necessary to represent the tree, which is proportional to $|T_{\mathcal{T}}(x^n)|$. Figure 6.5 shows, for the same image as before, the amount of memory necessary (in our implementation) to represent the trees used by the K-WLSTMs comparing random models against the ones evaluated by the greedy algorithm. We observe that the amount of memory could easily reach unpractical values if the templates were not carefully chosen.

6.4 Conclusions

In this chapter, we empirically demonstrated that the DITO$^{K-\mathrm{SCM}}$ performs generally well on K-SCM generated images. Nevertheless, DITO$^{K-\mathrm{SCM}}$ and DITO$^{K-\mathrm{WLSTM}}$ can considerably fail for other images. DITO$_D$ and DITO$_{DS}$ can give some improvements on some images but still fail on the same cases as DITO does. Therefore, the greedy algorithms proposed in this chapter can be simple and fast methods to obtain good solutions for some instances of the template optimization problem but cannot be considered reliable in every case. The evaluation cost analysis presented in this chapter suggests that, in order to have a reasonable computing performance, an heuristic designed for the sparse template problem ought to perform its search, when possible, by evaluating not too heavy and not too bad (in terms of code length) templates.

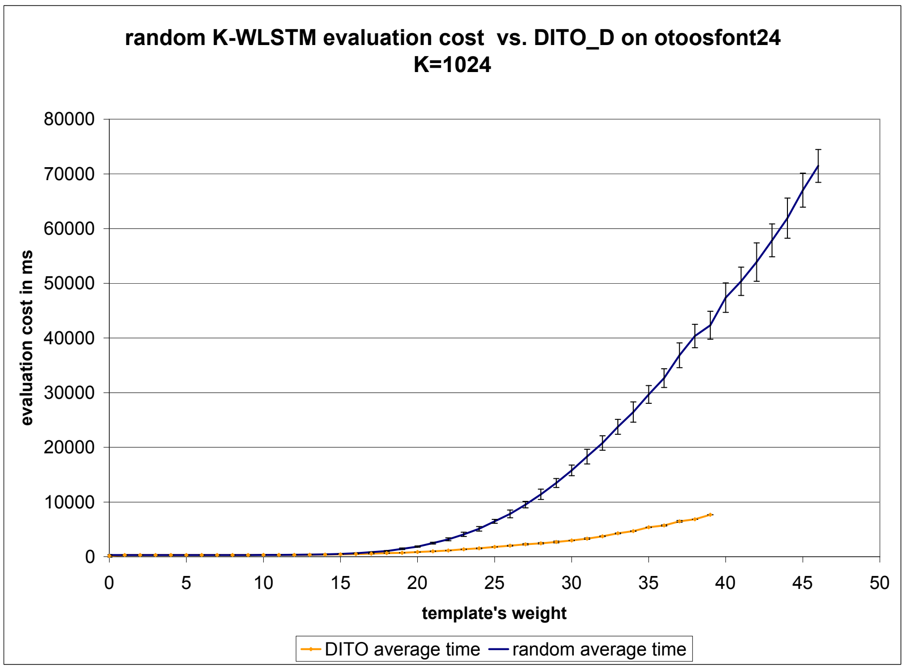


(a) Cost vs. weight

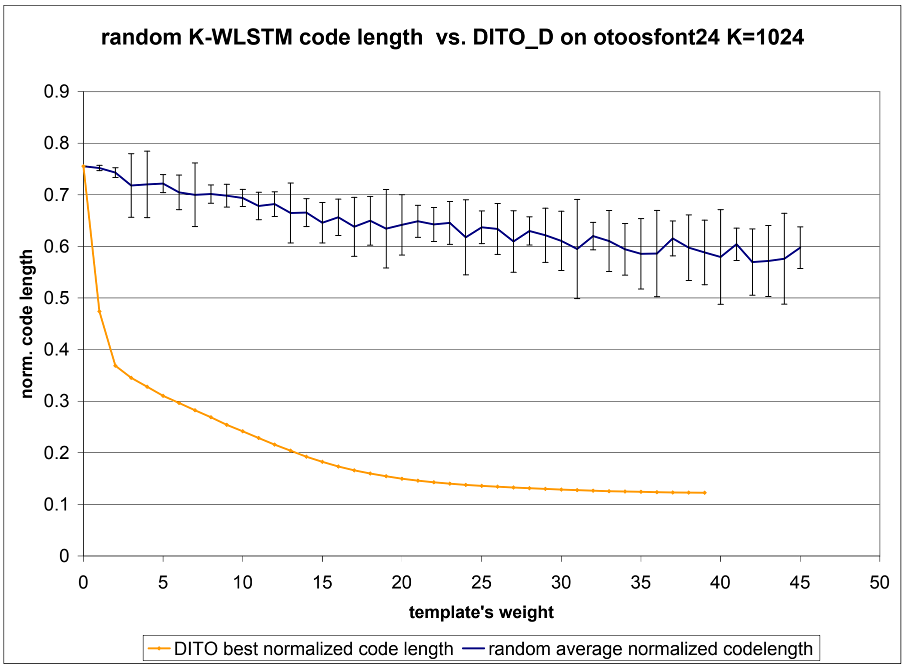


(b) Code length vs. weight

Figure 6.3: Relationships between evaluation cost and template weight and between evaluation cost and code length for K-SCMs using random templates and those found by DITO$_{DS}$. In the random case, 20 templates per weight were evaluated.



(a) Cost vs. weight



(b) Code length vs. weight

Figure 6.4: Relationships between evaluation cost and template weight and between evaluation cost and code length for K-WLSTMs using random templates and those found by the DITO$_D$. In the random case, 20 templates per weight were evaluated.

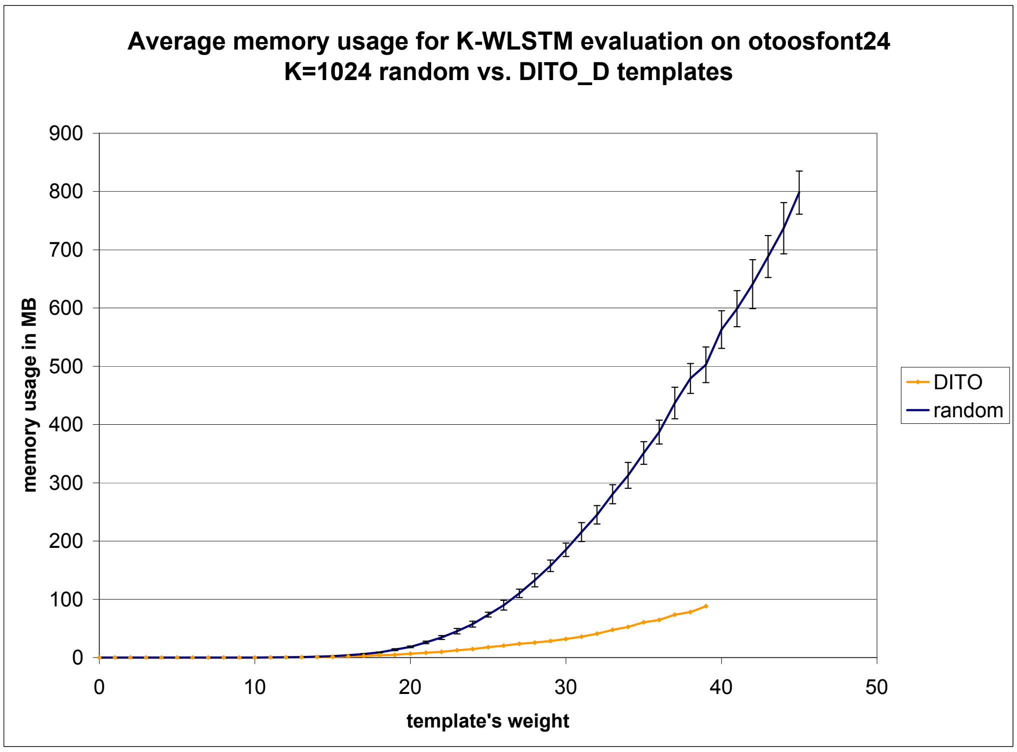


Figure 6.5: Relationship between template weight, KT code length and memory usage for K-WLSTMs. In the random case, 20 templates per weight were evaluated.

Chapter 7

ERGTO: an improved genetic algorithm

In this chapter, we study modifications to the genetic algorithm BRGTO described in Section 3.2. The goal of the modifications is to improve the computational efficiency of the algorithm. Although more efficient computations are a worthy goal in themselves, one of our main objectives is that the increased efficiency lead to improvements in compression performance, by enabling the practical use of larger context templates as well as modeling extensions as K-WLSTMs. We seek to apply some of the lessons learned from the greedy algorithms DITO described in Chapter 6, so that the modified genetic algorithm, which we call ERGTO (Enhanced Randomized Genetic Template Optimization), acquires some of the computational properties of DITO, while maintaining the randomness properties that help in preventing getting stuck in local minima. The modification will involve taking guidance from DITO in choosing parameter values for some of the components of BRGTO, and in some cases, changes in the choice of components themselves. In particular, we tested additional components from the genetic algorithms toolbox in the literature. In some cases, these additional components provided some advantages, while in others they did not, as will be detailed in the sequel. Overall, significant computational improvements were achieved, and, consequently, the desired improvements in compression performance were also obtained.

7.1 Optimization criteria and procedure

Some generic procedures for tuning the parameters of evolutionary algorithms have been proposed in the literature. These include Racing [BSPV02], Meta Evolutionary-Algorithms [Bäc96, Gre86] or a combination of the previous two [YG07]. These procedures would generally be expensive in the case of sparse template optimization because of the high cost of model evaluation analyzed in Chapter 5. Therefore, instead, we use a simpler method, somewhat inspired on Racing and based on the idea of emulating the DITO algorithms.

We aimed at optimizing the total evaluation cost (time) $\boldsymbol{T}$ required by the genetic algorithm to reach some appropriate code length value $L^*(x^n)$. The solution domain is defined by a set P of parameters and a set of possible values V_p for each parameter $p \in P$. For the optimization experiments, we used a synthetic image generated by a K-WSLTM M, since it gives us a known reference code length $L_M(x^n)$ to measure the genetic algorithm against.[1] The image, shown in Figure 7.1, was generated by a K-WLSTM with $K = 128$ optimized for the image "cmfugue1-0" (Figure A.12). The synthetic image was generated using the procedure described in Subsection 4.4.2. The image and the window size were chosen as a representative case for which K-WLSTMs perform well. For the target code length, we used $L^*(x^n) = 1.01 L_M(x^n)$ since we aim at optimizing computational complexity which is strongly determined by the weight and goodness of the evaluated templates (see Chapter 5) and we empirically observed that, at this code length level, the algorithm spends most of its time in evaluating templates whose weight is close to optimal.

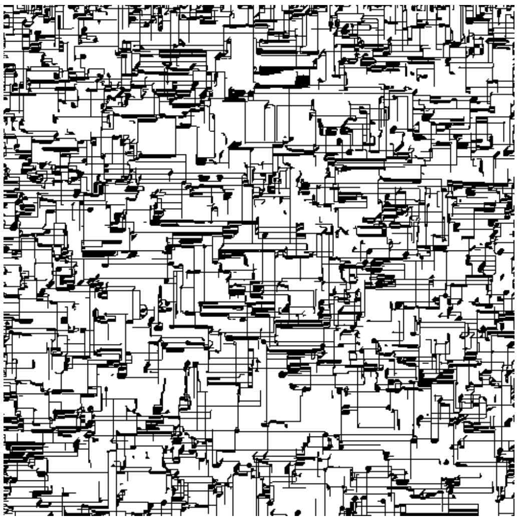


Figure 7.1: Generated image based on the best K-WLSTM found by DITO$_D$ for $K = 128$ on the image "cmfugue1-0" (Figure A.12).

Additionally, since genetic algorithms are of a random nature, we need to take into account the variance of the results and give an advantage to parameter values that yield a smaller variance in the results when different runs initialized by different PRNG seeds (see Section 3.1) are considered. For this reason, when evaluating $\boldsymbol{T}$ for each combination i of parameter values $\left(v_1^i, v_2^i, \ldots, v_{|P|}^i\right)$, we performed 10 runs of the algorithm with different seeds and we measured the evaluation cost T_s for each seed s. Therefore, we used the following cost function:

$$\boldsymbol{T}\left(v_1^i, v_2^i, \ldots, v_{|P|}^i\right) = \overline{T_s}\left(v_1^i, v_2^i, \ldots, v_{|P|}^i\right) + \overline{\sigma}_{T_s}\left(v_1^i, v_2^i, \ldots, v_{|P|}^i\right), \tag{7.1}$$

where $\overline{T_s}$ and $\overline{\sigma}_{T_s}$ are, respectively, the empirical average and the empirical standard deviation of T_s over the 10 runs. We set a maximum running time of the algorithm of approximately 1 hour in order to avoid wasted computing time, since we observed that this amount of time was sufficient in most cases to reach the objective value $L^*(x^n)$ with appropriate parameter values. In cases where the algorithm failed to reach $L^*(x^n)$ before the maximum running time, the truncation time,[2] which is in this case a lower bound for the required running time, was still used, in lieu of T_s, in the cost computation (7.1) (these cases are indicated in Figure 7.2 with parentheses).

The general idea in the optimization was to modify the genetic algorithm so that it starts

[1]Although, due to the complexity of the procedure, only one image was used in the optimization, the resulting optimized parameters and algorithm worked well for the variety of image types used in the test set of the results presented in Section 8.5.

[2]The truncation condition is evaluated after each generation is completely evaluated and, therefore, the truncation time can be greater than the maximum running time.

from a population of weight 1 and performs only slight mutations on combinations of the very best individuals, similarly to what the greedy algorithm does. When the weight of the optimal template is high, of course, the genetic algorithm must perform parts of its search in heavy populations, but our intuitive idea is to make it find most of the good combinations of template locations while searching in lighter populations instead of performing most of its search in populations whose weight is close to the optimal one (as observed in the implementation of [Ser04]). The optimization procedure is summarized in Algorithm 7.1.

Algorithm 7.1 Procedure for selecting the parameter values of ERGTO

```
Input: a set of parameters $P$ to be optimized,
a set of values $V_p$ and an initial value $v_p^{(0)} \in V_p$ for each $p \in P$
Output: a preferred value $v_p^*$ for each $p \in P$
for each parameter $p \in P$
  set $v_p := v_p^{(0)}$
end for
repeat as necessary
  for each parameter $p \in P$
    find $v_p^* \in V_p$ that minimizes $\boldsymbol{T}\left(v_1, v_2, \ldots, v_p^*, \ldots, v_{|P|}\right)$
    set $v_p$:=$v_p^*$
  end for
end repeat
```

Next, we enumerate (in the order in which the for loop of Algorithm 7.1 was executed) each parameter $p \in P$ and the corresponding values for V_p, $v_p^{(0)}$ and v_p^*. The discussion refers to the tables in Figure 7.2, which summarizes parameter values and timing results.

1. Selection strategy: when choosing parents for recombination, a probability of being selected is assigned to each individual. Two schemes were considered:
   - Windowing: the probability of selecting an individual i is proportional to

   $$\max_{w \in population} fitness\,(w) - fitness\,(i)$$

   (as in BRGTO, the fitness of an individual is the code length given by the template represented by it)
   - Ranking: the probability of selecting an individual i depends only on its position $r\,(i)$ in the population sorted by fitness. The following commonly used alternatives were evaluated as probability assignments: proportional to $1/(1+r(i))$, $1/(1+r(i))^2$ and $1/\sqrt{1+r(i)}$. These alternatives allow different levels of probability concentration on the best individuals. We also evaluated the same piecewise uniform geometric distribution described in Section 3.2 using $\gamma = 0.8$ and bins of size 10.

   An initial value is not needed in this case, since this is the first parameter optimized. The preferred value selected after the optimization was the ranking strategy with probability assignment proportional to $\frac{1}{(i+1)^2}$ (see Figure 7.2a).

2. Crossover type: In addition to the *averaging* crossover used in BRGTO and described in Section 3.2, we evaluated a uniform crossover (defined in Section 3.1). The uniform crossover was the initial value and also the preferred value selected after the optimization (see Figure 7.2b).

3. Expected number of flips per mutation: instead of specifying a mutation probability of each gene, we specify the expected number of mutations for the whole chromosome in order to keep this value fixed when varying K. As previously mentioned, the intention is to skew the genetic algorithm so that it spends more of its time searching in light populations. A mutation rate too high makes the algorithm rapidly reach a population of weight close to the optimal one and this slows down the search and eventually might make it impractical to reach a satisfactory solution. Therefore, we tested the values 0.5, 2 and 3. Since DITO$_D$ adds or deletes one location per evaluated template, here we chose 1 as the initial value, which turned out to be the preferred value selected after the optimization (see Figure 7.2c).

4. Expected number of swaps per mutation: for the same reasons as for the previous parameter, we specify the expected number of swaps for the whole chromosome. We tested the values 0 (i.e., no swaps),0.5, 2 and 3. The initial value used was 0 but, in this case, the selected value after the optimization was 0.5 (see Figure 7.2d).

5. Recombination type: two schemes were evaluated:
   - *not conservative*: this is the scheme used in the original genetic algorithm, in which the children undergo the mutation process after the crossover. A design decision we took was that, in the case that a uniform crossover is used, one child is mutated by flips and the other by swaps. When an averaging crossover is used, each child is mutated by flips first and then by swaps.
   - *conservative*: this type of reproduction keeps a copy of the result of each crossover in the new generation without undergoing any mutation process. Another copy of each child is mutated as in the previous scheme.

   The conservative type was the initial value and also the preferred value selected after the optimization (see Figure 7.2e).

6. Number of survivors: we evaluated how many distinct individuals should directly pass into the next generation. In order to guarantee a strictly decreasing evolution of fitness during the algorithm evolution, it is necessary to let survive at least the best individual of each generation. We tested the values 1, 2, 4, 8 and 16. Higher values for this parameter tend to concentrate the search on a fixed set of individuals, which can be bad for escaping from local minima. In the case of the DITO algorithms, the evaluated templates are slightly modified versions of the best of the previous stage. Therefore, we chose 1 as the initial value, which turned out to be the selected value after the optimization (see Figure 7.2f).

7. Hill Climbing strategy: *Hill climbing* consists of taking the best of each generation and trying to improve it by generating N new individuals by performing, in each case, one flip on the best individual at a time. We tested the values 0 (i.e., no hill climbing), 5, 10 and 20 for N. The results shown in Figure 7.2g suggest that no higher values are worth a try. The initial value used was 0, which turned out to be the preferred value selected after the optimization (see Figure 7.2g).

8. Population size: It is a known rule of thumb[3] in GAs that population size should be proportional to chromosome size. The tested values were $\frac{K}{4}$, $\frac{K}{2}$, $2K$, $4K$ and $8K$. Since DITO$_D$ evaluates K templates in each stage, we chose K as the initial value for population size which turned out to be the preferred value selected after the optimization (see Figure 7.2h).

[3]See, for example, http://eislab.gatech.edu/people/scholand/gapara.htm (as April 2009)

Figure 7.2 shows the value of the cost function $\boldsymbol{T}$ evaluated for each combination of parameter values resulting from the optimization procedure summarized in Algorithm 7.1.

Selection	Ranking: $\frac{1}{i+1}$	**Ranking:** $\frac{1}{(i+1)^2}$	Ranking: $\frac{1}{\sqrt{(i+1)}}$	Ranking: Geometric $\gamma = 0.8$, $binsize = 10$	Windowing
$\boldsymbol{T}$	58.4	**49.7**	(75.5)	(63.0)	(74.1)

(a)

Crossover type	**Uniform**$^{(0)}$	Averaging
$\boldsymbol{T}$	**49.7**	52.3

(b)

Expected flips	0.5	$\mathbf{1}^{(0)}$	2	3
$\boldsymbol{T}$	51.2	**49.7**	(68.5)	(76.2)

(c)

Expected swaps	$0^{(0)}$	**0.5**	1	2	3
$\boldsymbol{T}$	49.7	**34.8**	39.5	60.1	43.1

(d)

Recombination	**Conservative**$^{(0)}$	Non conservative
$\boldsymbol{T}$	**34.8**	39.5

(e)

Survivors	$\mathbf{1}^{(0)}$	2	4	8	16
$\boldsymbol{T}$	**34.8**	43.2	42.2	46.7	39.1

(f)

N (hill climbing)	$\mathbf{0}^{(0)}$	5	10	20
$\boldsymbol{T}$	**34.8**	76.0	(87.8)	(87.8)

(g)

Population size	$\frac{K}{4}$	$\frac{K}{2}$	$K^{(0)}$	$2K$	$4K$	$8K$
$\boldsymbol{T}$	(50.5)	40.1	**34.8**	46.4	(63.0)	(83.3)

(h)

Figure 7.2: Cost function value, according to (7.1), for each parameter value combination resulting from the optimization procedure. The cost function value of the cases where for some run the maximum allowed time was reached is enclosed in parentheses. The parameters are shown in the same order that were considered in the for loop of Algorithm 7.1. The initial values $v_p^{(0)}$ are indicated with a superscript $^{(0)}$ and the preferred values v_p^* are displayed in boldface. Although it would be natural to perform another iteration of the algorithm since the preferred value for swaps differed from its initial value, it was not performed because of the high computing cost of the iterations.

In general, we observe in Figure 7.2 that there is a high sensitivity of the evaluation cost $\boldsymbol{T}$ to the different parameter values, which helps explain the ultimate usefulness of the optimization.

In addition to the strategies and parameter values previously discussed, a particularity in the design of ERGTO for K-WLSTMs (which we denote as ERGTO$^{K-\text{WLSTM}}$) is that, after the pruning step of the evaluation algorithm, if some locations of the template get no longer used (i.e., they become holes), then the corresponding genes are removed from the chromosome. This helps in reducing the weight of the evaluated individuals.

Finally, we tested the idea of including the result of DITO$_D$ in the initial population of the genetic algorithm for K-WLSTMs. It was observed that in some cases where DITO$_D$ gets stuck in a local minimum, including this "bad" solution in the initial population can make the genetic algorithm get stuck (or spend a lot of time) in the same local minimum, so the idea was not adopted.

7.2 Comparison with BRGTO

In this section, we compare, in the setting of K-SCMs, ERGTO (which we denote as ERGTO$^{K-\mathrm{SCM}}$ in this case) and BRGTO in terms of overall computing performance in order to assess the optimizations detailed in Sections 5.3 and 7.1. We ran both algorithms for $K = 128$, on the image "albert2D" (Figure A.5). In the case of BRGTO, we used the best set of parameter values reported in [Ser04] for that specific image and K, which are presented in Table 7.1.

M	m	μ_r	μ_m	γ	B
300	3	0.002	0.5	0.75	7

Table 7.1: Parameters values used for BRGTO in the comparison experiment.

The algorithm ERGTO$^{K-\mathrm{SCM}}$ was run with the parameter values resulting from our optimization. Table 7.2 summarizes the results.

We observe that ERGTO$^{K-\mathrm{SCM}}$ largely outperforms BRGTO with gains in running time larger than 32.2×. In addition, it takes less than 60% of the number of generations to reach better code length values with less than half the number of individuals per generation of BRGTO.

This example, which is quite typical of the overall comparison on binary images, demonstrates that our improvements were highly effective in reducing computing time by improving the convergence rate of the algorithm, by reducing the computing time for each individual (as shown in Chapter 5) and by generally selecting individuals that were faster to evaluate. These improvements in computing efficiency enabled the study of K-WLSTMs and of larger window sizes.

Algorithm	Code length value reached vs. best known K-SCM	Number of generations	Time in s	Time ratio
BRGTO	+1.98%	29	4669	
ERGTO	+1.78%	17	106	**44.0**
BRGTO	+1.07%	51	5151	
ERGTO	+0.81%	29	160	**32.2**

Table 7.2: Performance comparison between ERGTO and BRGTO. We measured the time and the number of generations taken by both algorithms to reach a code length close to +1% and +2% the code length given by the best known K-SCM for the input image with $K = 128$. The best result found by BRGTO in 2000 generations was within +1.06% of the best known result and was found in 115 generations. Differences with the code length given by the best known template $\mathcal{T}$ are calculated as $(L_{\{\mathrm{B}|\mathrm{E}\}\mathrm{RGTO}} - L_{\mathcal{T}})/L_{\mathcal{T}}$, expressed as a percentage.

7.3 Stopping criterion

The last criterion to be determined was the one that tells the algorithm when to stop. The criteria of setting a time limit or a maximum number of generations is very dependent of the input data and, therefore, difficult to adjust. A more flexible alternative is to set a maximum number *maxstall* of consecutive generations without improvements in the objective function, called *stalling* generations. In other words, if the evolution gets stuck over a number of generations we assume that there is little chance of further improvements.

Varying the number of stalling generations offers a trade-off between evaluation cost and quality of the solutions. As an example, we evaluated this trade-off over the same generated image that we used for the other parameters and we ran the algorithm with 20 different seeds. Four values of *maxstall* were evaluated, namely 1,2,3 and 4. The results are shown in Figure 7.3. One curve shows the average evaluation cost + 1 estimated standard deviation versus average fitness of the best individual (normalized by the generator's fitness) + 1 estimated standard deviation. The other curve shows the worst case over the 20 runs for both dimensions.

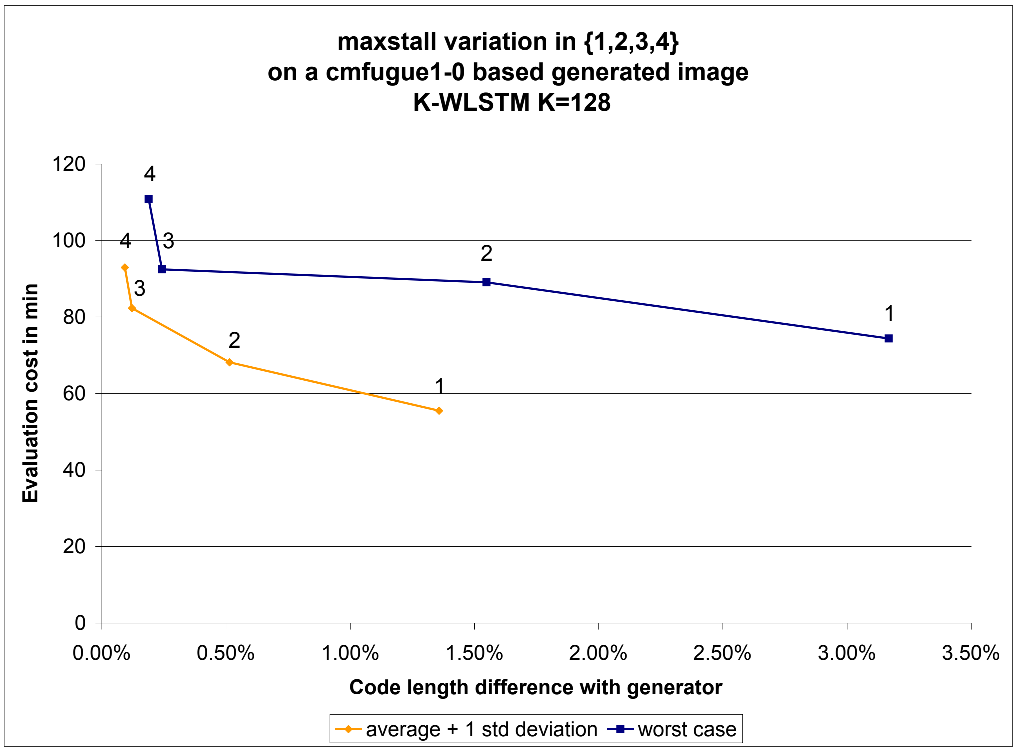


Figure 7.3: Trade-off between evaluation cost and quality of the solution when varying the maximum number of stalling generations allowed. The curve points represent the results for values 1,2,3 and 4 from right to left in each curve.

We observe that 3 stalling generations may be a good choice for this trade-off since the slope of the curves considerably increases in the last segment and the corresponding difference with the fitness given by the generating template for this value in both curves falls below +0.25% which is quite small. Nevertheless, in cases where we are willing to spend more computation time in exchange for an even better solution, the value of *maxstall* can be increased.

7.4 Conclusions

We described ERGTO, an improved version of the genetic algorithm BRGTO of [Ser04], described in Section 3.2. ERGTO$^{K-\mathrm{SCM}}$ shows very significant computational gains over BRGTO, enabling the study of K-WLSTMs and of much larger window sizes. ERGTO was optimized for K-WLSTMs, since these models offer the best promise of compression performance, and at the same time have the heaviest computational requirements.

Chapter 8

Compression performance of sparse models and their algorithms

In this chapter, we further explore compression-complexity trade-offs quantitatively addressing the following questions:

- How significant are the compression performance benefits of K-WLSTMs over K-SCMs (which are cheaper to evaluate)?
- How significant are the compression performance benefits of augmenting the window size of sparse models (which increases optimization difficulty)?
- How significant are the compression performance benefits of sparse models over the best contiguous model within the same window size?

Additionally, in order to illustrate how our sparse modeling algorithms "learn" the structure of binary images, we show some templates, found by the algorithms, that strongly track the patterns of the input images.

Finally, in Section 8.5, we present the main practical results of this work: a comparison of the compression performance of ERGTO$^{K-\text{WLSTM}}$ against that of standard compression methods on binary images. The comparison, which was run over the test set described in Section 4.5, shows that ERGTO$^{K-\text{WLSTM}}$ (for $K = 1024$) outperforms, in most general cases, the standard compression methods, and, in some cases, by significant margins.

8.1 Variable vs. fixed length conditioning (K-WLSTMs vs. K-SCMs)

In this section, we compare the performance of K-WLSTMs against that of K-SCMs. Because of the tree pruning step and the usually higher weight of the optimum templates (see Appendix A), K-WLSTMs are usually significantly more expensive to be evaluated (see Chapter 5). Nevertheless, the differences in compression rate as shown in Table 8.1 for K in $\{32, 64, 128, 256, 512\}$ for the test set, demonstrate that K-WLSTMs have a greater compression potential in comparison to K-SCMs.

Image	$K = 32$	$K = 64$	$K = 128$	$K = 256$	$K = 512$
1.1.01M	**-0.1%**	**-0.1%**	**-0.1%**	**-0.1%**	**-0.1%**
1.1.13M	**-0.4%**	**-0.4%**	**-0.4%**	**-0.4%**	**-0.4%**
1.5.02M	**-4.3%**	**-5.3%**	**-4.5%**	**-4.1%**	**-0.5%**
A-fixedwidth6and8	**-2.4%**	**-25.3%**	**-44.4%**	**-44.4%**	**-44.4%**
albert2D	**-4.0%**	**-3.0%**	**-5.0%**	**-6.0%**	**-7.2%**
amb	**-3.0%**	**-3.3%**	**-4.1%**	**-4.1%**	**-4.1%**
Bach_CPE-Sonata_flauto_...	**-6.2%**	**-7.5%**	**-8.0%**	**-8.0%**	**-8.0%**
Bobbys_letter_page_1	**-2.5%**	**-2.7%**	**-2.8%**	**-3.0%**	**-3.2%**
ccitt4small	**-2.6%**	**-3.0%**	**-4.5%**	**-5.4%**	**-5.4%**
ccitt7small	**-1.9%**	**-2.6%**	**-2.8%**	**-2.6%**	**-2.4%**
chinese_text	**-0.7%**	**-1.0%**	**-1.4%**	**-1.4%**	**-1.4%**
cmfugue1-0	**-6.3%**	**-8.6%**	**-9.1%**	**-9.1%**	**-9.1%**
flakes006-inca-100dpi...	**-0.3%**	**-0.4%**	**-0.4%**	**-0.4%**	**-0.4%**
HALFTONE	**-1.7%**	**-1.7%**	**-2.5%**	**-1.2%**	**-1.9%**
Halftone2	**-1.1%**	**-2.1%**	**-2.2%**	**-2.5%**	**-2.2%**
Halftone3	**-0.4%**	**-2.0%**	**-2.9%**	**-3.2%**	**-3.2%**
hamilton_bw	**-1.4%**	**-2.5%**	**-2.5%**	**-2.5%**	**-2.5%**
hamilton_ed	**-0.7%**	**-1.0%**	**-1.2%**	**-1.2%**	**-1.2%**
hieroglyph	**-1.6%**	**-1.7%**	**-2.6%**	**-2.6%**	**-3.0%**
leeleter	**-2.0%**	**-2.6%**	**-3.0%**	**-3.1%**	**-3.1%**
lena_j	+0.5%	+0.4%	+0.4%	+0.4%	+0.4%
otoosfont12	**-10.0%**	**-14.8%**	**-16.1%**	**-16.2%**	**-16.5%**
otoosfont24	**-13.2%**	**-21.1%**	**-23.5%**	**-23.5%**	**-23.8%**
pep_j	+0.3%	+0.3%	+0.3%	+0.3%	+0.3%
texmos1.p512M	**-0.2%**	**-0.2%**	**-0.2%**	**-0.2%**	**-0.2%**
wallpaper003-inca-100dpi...	**-0.7%**	**-0.8%**	**-0.8%**	**-0.8%**	**-0.8%**
wallpaper004-inca-100dpi...	**-0.1%**	**-0.3%**	**-0.5%**	**-0.1%**	**-0.1%**
wallpaper010-inca-100dpi...	**-1.8%**	**-1.9%**	**-2.0%**	**-1.6%**	**-1.4%**
writing	**-2.9%**	**-3.2%**	**-3.2%**	**-3.2%**	**-3.2%**

Table 8.1: Difference in compression rate of K-WLSTMs vs. K-SCMs. Differences are calculated as $(L_{K-\text{WLSTM}} - L_{K-\text{SCM}})/L_{K-\text{SCM}}$, expressed as a percentage. Negative numbers represent gains in compression rate, and larger magnitude numbers represent larger gains.

In general, we observe that K-WLSTMs give better results, as expected, since they are a superset of K-SCMs. In exchange for this class generalization, the tree description cost (described in Subsection 2.2.2) has to be paid. Therefore, it is possible to have a worse compression rate with a K-WLSTM if it actually represents a tree that is close to a K-SCM (i.e., when the tree is close to balanced), since we would be paying the extra cost of describing the tree with little or no benefit in code length in exchange. However, even in this case, the tree does not need to be complete, since not all the states necessarily appear. This is the case of the halftoned images "lena_j" (Figure A.21) and "pep_j" (Figure A.24), where the compression rates given by K-WLSTMs are slightly worse than those given by K-SCMs, for every K in $\{32, 64, 128, 256, 512\}$. Figure 8.1 shows the tree description cost for both images and $K = 512$ and confirms that the tree description overhead is the cause of K-WLSTMs being worse in these cases.

The results for the image "1.5.02M" (Figure A.3) are particularly interesting. The circles in the image are evenly distributed and there are only slight irregularities in each circle. In Figure 8.2, we show the templates of the best K-WLSTM and K-SCM found for K in $\{256, 512\}$. For $K = 512$, both types of sparse models are able to fully capture the distance between the centers of the circles

$L_{K-\mathrm{SCM}}(x^n)$	$L_{K-\mathrm{WLSTM}}(x^n)$	$\lvert T'\rvert$	$\lvert T'\rvert / L_{K-\mathrm{SCM}}(x^n)$
168941	169612	2761	1.6%

(a) Case of the image "lena_j" $K = 512$.

$L_{K-\mathrm{SCM}}(x^n)$	$L_{K-\mathrm{WLSTM}}(x^n)$	$\lvert T'\rvert$	$\lvert T'\rvert / L_{K-\mathrm{SCM}}(x^n)$
161939	162346	2611	1.6%

(b) Case of the image "pep_j" $K = 512$.

Figure 8.1: Tree description cost $|T'|$ in cases where K-WLSTMs perform worse that K-SCMs.

(horizontally and vertically). For smaller windows, the models can only "see" an alternation of situations and cannot capture the full structure. It is reasonable to think that, in these cases, K-WLSTMs should be beneficial thanks to the flexibility given by variable length conditioning. This is what happens for $K < 512$ since there is a benefit in using a K-WLSTM while for $K = 512$, there is almost no difference.

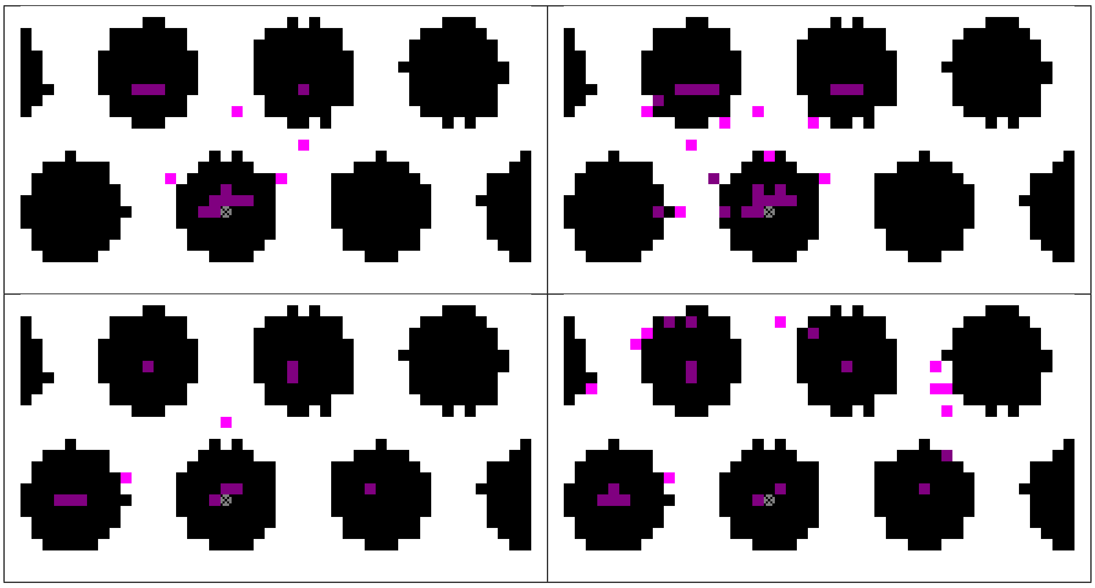


Figure 8.2: Best sparse templates found for "1.5.02M" (Figure A.3) displayed on a portion of the image. On top: $K = 256$, on bottom: $K = 512$, on left: K-SCM, on right: K-WLSTM. The current sample is marked with a cross. Shaded pixels represent template locations.

Another example of such a situation is the case of the image "A-fixedwidth6and8" (Figure A.4) where there are two perfectly periodic zones each one with a different period and with repeated symbols also different. Again, K-WLSTMs take advantage of this situation and give gains in compression rate up to 44.4%.

Other cases where K-WLSTMs take a significant advantage over K-SCMs are, for example:

- text images like "otoosfont24" (Figure A.23) which have different regular structures like character shapes and distances between characters
- score images like "cmfugue1-0" (Figure A.12) which have different regular structures like lines, notes, distance between lines, etc.
- the halftoned image "albert2D" (Figure A.5) which has a large white contiguous zone in the hair part and a very periodic zone in the sweater part.

On the other hand, there are images like, for example, the texture image "1.1.01M" (Figure A.1) which lack those kinds of regularities and make both types of models capture only the close contiguous dependencies.

8.1.1 K-WLSTMs based on K-SCM optimized templates

A simple idea that can be used to improve, at a reduced computational cost, the results found by any K-SCM optimization algorithm is to optimize a context tree based on the template $\mathcal{T}_{K-\text{SCM}}$ obtained by that algorithm, resulting in a K-WLSTM that is used for coding. Since variable length conditioning should allow K-WLSTMs to use more locations in the templates, this simple method would not let us use the full potential of K-WLSTMs. We tested this idea on some of the images of the test set for which K-WLSTMs give significant improvements over K-SCMs. The results are given in Table 8.2.

Image	Weight of $\mathcal{T}_{K-\text{SCM}}$	Weight of the best K-WLSTM template	K-WLSTM based on $\mathcal{T}_{K-\text{SCM}}$ vs. best K-SCM	Best K-WLSTM vs. K-WLSTM based on $\mathcal{T}_{K-\text{SCM}}$
albert2D	17	31	-2.9%	-2.1%
1.5.02M	16	27	-1.7%	-2.9%
cmfugue1-0	16	31	-3.6%	-5.7%
otoosfont12	21	47	-4.2%	-12.4%

Table 8.2: Comparison of code length given by K-WLSTMs based on $\mathcal{T}_{K-\text{SCM}}$ with the best found K-SCMs and K-WLSTMs. Differences in the fourth and fifth columns are calculated as, respectively, $\left(L_{K-\text{WLSTM}(\mathcal{T}_{K-\text{SCM}})} - L_{K-\text{SCM}}\right)/L_{K-\text{SCM}}$ and $\left(L_{K-\text{WLSTM}} - L_{K-\text{WLSTM}(\mathcal{T}_{K-\text{SCM}})}\right)/L_{K-\text{WLSTM}(\mathcal{T}_{K-\text{SCM}})}$, expressed as a percentage. Negative numbers represent gains in compression rate, and larger magnitude numbers represent larger gains.

Although some improvement over K-SCMs is achieved by this method, in the last two images we see a significant difference in advantage of the best K-WLSTM, which confirms our intuition. Also, we observe that, in every case, the best K-WLSTM templates include much more locations than the best K-SCM template, as expected.

8.2 Sparse models vs. contiguous models

By using sparse models, which are a superset of contiguous models, we expect to improve compression performance in cases better suited for them but at the same time to obtain a good performance in cases better suited for contiguous models. Table 8.3 shows the difference in compression rate of sparse models vs. contiguous models optimized for the same window sizes ($K \in \{32, 64\}$) on the test set.

We see that in every case sparse models are better than contiguous models as expected. Sometimes the gains are small as in the case of the order dithered image "Halftone2" (Figure A.15) where halftone patterns are 2 pixels away and, therefore, this distance can be captured by a contiguous model without incurring in a great model cost. However, the order dithered image "HALFTONE" presents a different situation in which halftone patterns are 5 pixels away. In this case, although the distance between patterns can be fully captured within a window size $K = 64$ (the best contiguous tree model is found at $\tilde{K} = 43$ as shown in Figure 8.3), the contiguous model gives a

	K-SCM vs. fixed length contiguous model		K-WLSTM vs. tree model	
Image	$K=32$	$K=64$	$K=32$	$K=64$
1.1.01M	**-0.2%**	**-0.2%**	**-0.1%**	**-0.1%**
1.1.13M	**-0.3%**	**-0.3%**	**-0.1%**	**-0.1%**
1.5.02M	**-12.2%**	**-25.7%**	**-2.1%**	**-9.9%**
A-fixedwidth6and8	**-2.8%**	**-33.8%**	**-1.1%**	**-17.3%**
albert2D	**-4.7%**	**-20.7%**	**-0.2%**	**-4.5%**
amb	**-4.9%**	**-10.6%**	**-1.2%**	**-4.9%**
Bach_CPE-Sonata_flauto_solo_La_min-fl	**-1.3%**	**-3.6%**	**-0.7%**	**-3.0%**
Bobbys_letter_page_1	**-1.3%**	**-1.3%**	**-0.6%**	**-0.8%**
ccitt4small	**-3.6%**	**-6.1%**	**-1.8%**	*
ccitt7small	**-1.9%**	**-2.7%**	**-0.8%**	*
chinese_text	**-1.2%**	**-1.2%**	**-0.7%**	**-0.9%**
cmfugue1-0	**-4.3%**	**-5.4%**	**-1.9%**	**-4.9%**
flakes006-inca-100dpi-00M	**-0.3%**	**-0.3%**	**-0.3%**	*
HALFTONE	**-21.8%**	**-37.5%**	**-9.9%**	**-26.7%**
Halftone2	**-2.6%**	**-3.1%**	**-1.3%**	**-2.5%**
Halftone3	**-10.5%**	**-10.4%**	**-4.3%**	**-5.7%**
hamilton_bw	**-4.5%**	**-6.5%**	**-1.9%**	**-4.0%**
hamilton_ed	**-0.4%**	**-0.9%**	**-0.3%**	*
hieroglyph	**-1.5%**	**-1.5%**	**-0.8%**	**-0.9%**
leeleter	**-1.7%**	**-1.8%**	**-0.4%**	*
lena_j	**-9.4%**	**-9.4%**	**-10.0%**	*
otoosfont12	**-5.2%**	**-6.0%**	**-0.5%**	**-1.9%**
otoosfont24	**-2.0%**	**-2.6%**	**-0.8%**	**-3.7%**
pep_j	**-9.7%**	**-9.7%**	**-11.3%**	*
texmos1.p512M	**-0.2%**	**-0.2%**	**-0.1%**	*
wallpaper003-inca-100dpi-00M	**-1.2%**	**-1.2%**	**-0.8%**	*
wallpaper004-inca-100dpi-00M	**-0.9%**	**-1.4%**	**-0.7%**	*
wallpaper010-inca-100dpi-00M	**-0.7%**	**-1.0%**	**-0.2%**	*
writing	**-0.8%**	**-0.7%**	**-0.6%**	**-0.9%**

Table 8.3: Difference in compression rate of sparse models vs. contiguous models for K in $\{32, 64\}$, in both fixed and variable length conditioning cases. Differences are calculated as $^{(L_{sparse}-L_{contiguous})}/_{L_{contiguous}}$, expressed as a percentage. *For some images, contiguous tree models with $K = 64$ could not be evaluated (because of the computer memory requirements which are $O\left(2^K\right)$) and therefore these results are not shown (see Appendix D). Negative numbers represent gains in compression rate, and larger magnitude numbers represent larger gains.

poor compression rate in comparison to the sparse model result, since it is forced to include many context locations that increase model cost without having enough improvement in fitting the data to compensate.

In the case of images like "1.1.01M" (Figure A.1) in which dependencies are taken among contiguous pixels, we observe similar results between sparse and contiguous models. The only overhead that sparse models have is the template description that is compressed using an adaptive memoryless model as explained in Appendix C. Since these templates are usually sparse, they can be greatly compressed and their description length (which is independent of n) turns out to be negligible in most cases. For instance, Figure 8.4 shows the template description cost for the best K-WLSTM found for K in $\{64, 1024\}$ for "1.1.01M" (which is a small image in relation to the set's size average), in relation to the code length given by the best tree model for $K = 64$.

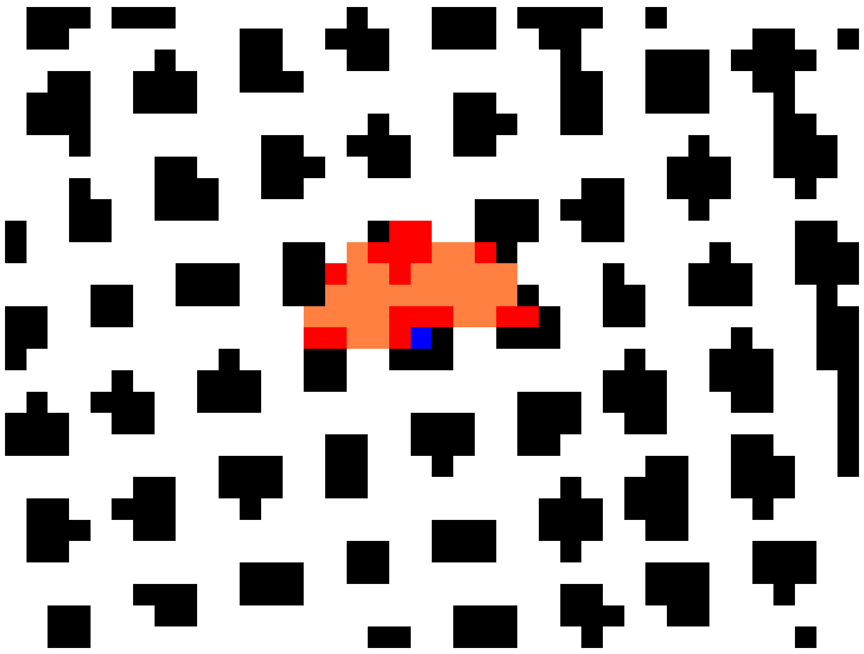


Figure 8.3: Best contiguous tree model for the image "HALFTONE" for $K = 64$, found at $\tilde{K} = 43$.

$L_{TreeK64}(x^n)$	$L_{K-WLSTM}(x^n)$	$L_{memoryless}(\mathcal{T})$	$L_{memoryless}(\mathcal{T})/L_{TreeK64}(x^n)$
167825	167710	34	0.02%

(a) Best K-WLSTM found for $K = 64$.

$L_{TreeK64}(x^n)$	$L_{K-WLSTM}(x^n)$	$L_{memoryless}(\mathcal{T})$	$L_{memoryless}(\mathcal{T})/L_{TreeK64}(x^n)$
167825	167761	141	0.08%

(b) Best K-WLSTM found for $K = 1024$.

Figure 8.4: Template description cost $L_{memoryless}(\mathcal{T})$ for the image "1.1.01M", a case where the best K-WLSTM performs similarly to the best contiguous tree model.

8.3 Evaluation of the benefit of increasing window size

In order to assess the benefit of increasing the window size K when using sparse models, we focus on the K-WLSTM case since its flexibility makes it more likely of taking advantage of larger windows if the image has some distant dependencies. When increasing the window size, the overhead in template description is generally quite small since it normally becomes sparser and, therefore, more compressible. Nevertheless, complexity usually increases significantly as the search space increases exponentially and the templates are more expensive to be evaluated (see Chapter 5). Table 8.4 shows the difference in code length when doubling K, on the test set.

For example, we notice that for "lena_j" (Figure A.21) there is almost no benefit in using $K > 32$ which is reasonable since the error-diffusion structures are fully captured with $K = 32$. On the other hand, for the halftoned image "albert2D" (Figure A.5), a significant benefit is obtained when using $K \geq 128$ in comparison to the cases of smaller window sizes. The texture image "flakes006-inca-100dpi-00M" (Figure A.13) is strongly characterized by its contiguous color zones which makes sparsity and larger windows less useful. In the case of the text image "otoosfont12" (Figure A.22) and the score image "cmfugue1-0" (Figure A.12), we notice that, although there could be some additional regularities captured by the model when increasing K, in fact, there is only a

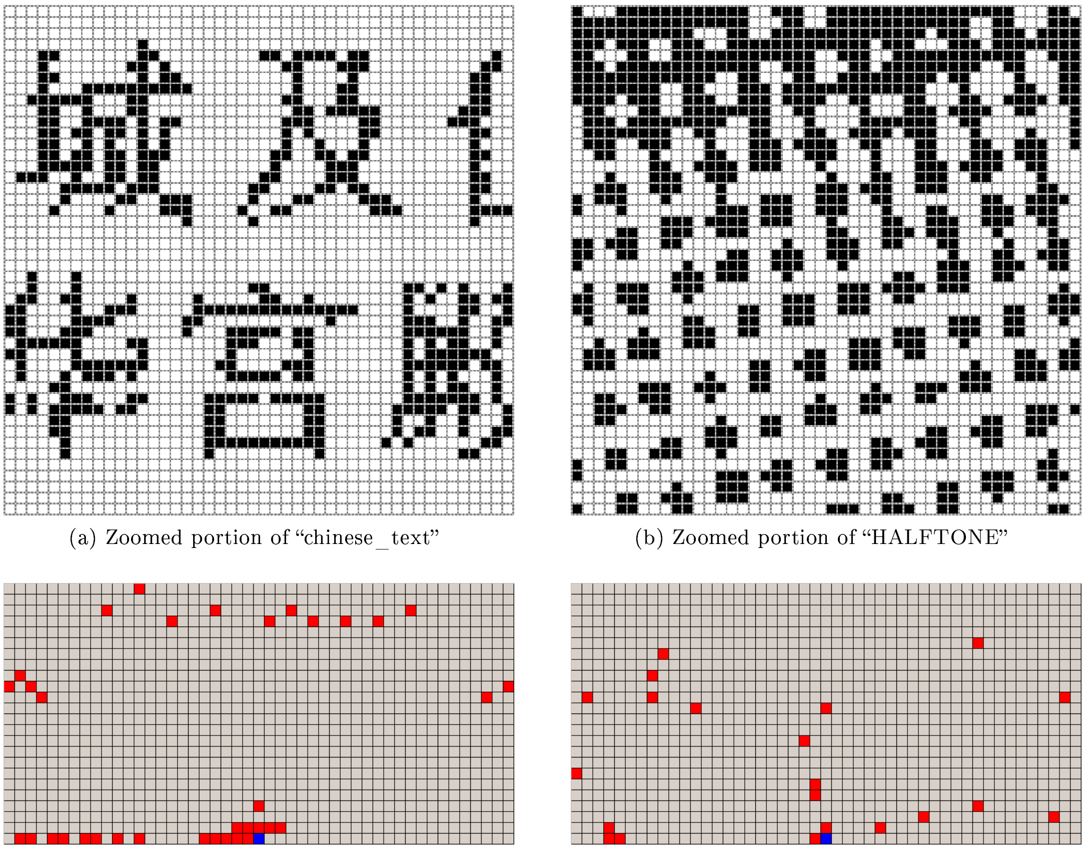
	K			
Image	64 vs. 32	128 vs. 64	256 vs. 128	512 vs. 256
1.1.01M	0.0%	0.0%	0.0%	0.0%
1.1.13M	0.0%	0.0%	0.0%	0.0%
1.5.02M	**-16.3%**	**-3.8%**	**-4.3%**	**-13.7%**
A-fixedwidth6and8	**-90.5%**	**-25.6%**	0.0%	0.0%
albert2D	**-16.0%**	**-5.5%**	**-1.0%**	**-2.1%**
amb	**-6.4%**	**-0.8%**	0.0%	0.0%
Bach_CPE-Sonata_flauto...	**-3.8%**	**-2.1%**	0.0%	0.0%
Bobbys_letter_page_1	**-0.3%**	**-0.1%**	**-0.1%**	**-0.2%**
ccitt4small	**-3.0%**	**-1.9%**	**-1.1%**	0.0%
ccitt7small	**-1.6%**	**-0.5%**	**-0.4%**	**-0.3%**
chinese_text	**-0.3%**	**-0.4%**	0.0%	0.0%
cmfugue1-0	**-3.7%**	**-0.6%**	0.0%	0.0%
flakes006-inca-100dpi...	0.0%	0.0%	0.0%	0.0%
HALFTONE	**-20.1%**	**-1.9%**	**-5.3%**	**-2.1%**
Halftone2	**-1.5%**	**-0.1%**	**-0.3%**	0.0%
Halftone3	**-1.6%**	**-1.2%**	**-0.3%**	0.0%
hamilton_bw	**-3.2%**	**-0.3%**	0.0%	0.0%
hamilton_ed	**-0.8%**	**-0.2%**	0.0%	0.0%
hieroglyph	**-0.2%**	**-0.9%**	0.0%	**-0.4%**
leeleter	**-0.8%**	**-0.4%**	**-0.1%**	0.0%
lena_j	**-0.1%**	0.0%	0.0%	0.0%
otoosfont12	**-6.0%**	**-1.6%**	**-0.1%**	**-0.4%**
otoosfont24	**-9.7%**	**-5.0%**	0.0%	**-0.4%**
pep_j	**-0.1%**	0.0%	0.0%	0.0%
texmos1.p512M	0.0%	0.0%	0.0%	0.0%
wallpaper003-inca-100dpi...	**-0.1%**	0.0%	0.0%	0.0%
wallpaper004-inca-100dpi...	**-0.7%**	**-0.3%**	**-0.6%**	0.0%
wallpaper010-inca-100dpi...	**-0.3%**	**-0.1%**	**-0.2%**	**-0.1%**
writing	**-0.3%**	0.0%	0.0%	0.0%

Table 8.4: Difference in compression rate of K-WLSTMs with window size K in $\{64, 128, 256, 512\}$ vs. K-WLSTMs with window size $K/2$. Differences are calculated as $(L^{2K}_{K\text{-WLSTM}} - L^{K}_{K\text{-WLSTM}})/L^{K}_{K\text{-WLSTM}}$, expressed as a percentage. Negative numbers represent gains in compression rate, and larger magnitude numbers represent larger gains.

slight improvement and the greatest improvement occurs when capturing closer dependencies like the symbol structure. This is not the case for the image "1.5.02M" (Figure A.3) since its almost perfect periodicity can be fully captured with $K = 512$ as we observed in Section 8.1 and, therefore, an important gain in compression rate (of 13.7%) over $K = 256$ is achieved. In the case of the image "A-fixedwidth6and8" (Figure A.4), doubling the window size, from $K = 32$ to $K = 64$ and from $K = 64$ to $K = 128$, gives great gains (of 90.5% and 25.6%, respectively) which is consistent with the size and the spacing of the characters in the image (see Figure A.4 for a detailed view).

8.4 Resemblance of sparse templates with binary image structures

In addition to the case of the image "1.5.02M" already shown in Figure 8.2, Figures 8.5 and 8.6 show cases of K-WLSTM templates found by our algorithms that present patterns quite similar to structures found in the input image. These similarities illustrate how these models capture the structure of these types of binary images.


(a) Zoomed portion of "chinese_text" (b) Zoomed portion of "HALFTONE"

(c) Best K-WLSTM for "chinese_text" (d) Best K-WLSTM for "HALFTONE"

Figure 8.5: Resemblance between best K-WSLTM templates and image structures.

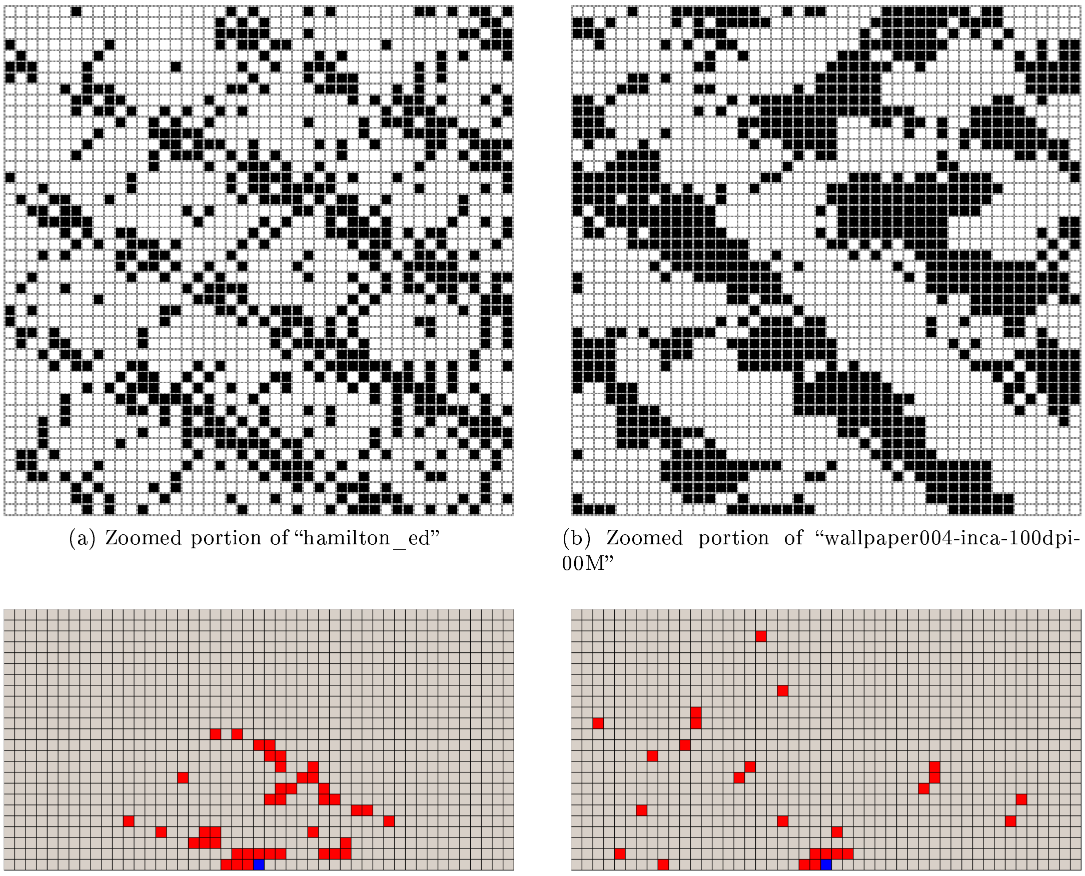
In the case of the image "chinese_text", we observe that the best K-WLSTM template found captures the horizontal and vertical distances between characters. Locations in the best K-WLSTM template for the image "HALFTONE" are arranged in a very similar way to how halftone patterns are in the input image. The template for "hamilton_ed" presents a striking resemblance with the drawing patterns of the image. In the case of "wallpaper004-inca-100dpi-00M", the template seems to capture the right angles and the distance between parallel edges.

8.5 Comparison of ERGTO$^{K-WLSTM}$ against popular standard methods

In this section, we show that ERGTO$^{K-WSLTM}$ with $K = 1024$ generally outperforms standard compression methods. The standard methods that we are going to compare with are JBIG, JBIG2, DjVu[1] in their lossless compression modes.

[1]We used the following implementations:

- JBIG: imagemagick (http://www.imagemagick.org, as of April 2009).
- JBIG2: Power JBIG-2 Coder, Signal Processing and Multimedia Group of the University of British Columbia and Image Power, Inc.


(a) Zoomed portion of "hamilton_ed" (b) Zoomed portion of "wallpaper004-inca-100dpi-00M"

(c) Best K-WLSTM for "hamilton_ed" (d) Best K-WLSTM for "wallpaper004-inca-100dpi-00M"

Figure 8.6: Resemblance between best K-WSLTM templates and image structures (cont.).

In Section 8.2, we compared sparse models against contiguous models in order to assess the effect of sparsity within the same window size. An important benefit of our sparse modeling algorithms is to allow much larger window sizes than those that can be practically used with contiguous tree models. Nevertheless, contiguous tree models with small window sizes (up to 64, see Appendix D) still give good results in comparison to standard methods. Therefore, we also include these results in the comparisons of this section.

8.5.1 Results on non-synthetic images

The results on the non-synthetic images of the test set are shown in Table 8.5.

As explained in Section 4.3, JBIG2 and DjVu use specialized pattern matching algorithms in text regions and different algorithms in other regions, while our sparse modeling algorithms are generic. Nevertheless, the table shows that ERGTO$^{K-WLSTM}$ outperforms JBIG and JBIG2 in *every* tested case. DjVu only beats ERGTO$^{K-WLSTM}$ in 3 cases of text documents. In particular, we see a great advantage for DjVu in the case of "otoosfont24" (Figure A.23). Nevertheless, in the lower resolution case of "otoosfont12" (Figure A.22), ERGTO$^{K-WLSTM}$ beats DjVu by a significant margin (of 9%).

- DjVu: DjVuLibre (http://djvu.sourceforge.net, as of April 2009)

Image	Image size (in bits, at 1 bit/pixel)	ERGTO normalized code length $K = 1024$ (in bits/pixel)	vs. contiguous tree $K \approx 64$	vs. DITO$_D$ $K = 1024$	vs. jbig	vs. jbig2	vs. djvu
1.1.01M	262144	0.640	0.0%	**+0.0%**	-4.6%	-11.6%	-13.4%
1.1.13M	262144	**0.543**	0.0%	0.0%	-4.9%	-9.6%	-11.4%
1.5.02M	262144	0.146	-28.4%	**+0.6%**	-51.0%	-47.8%	-24.3%
A-fixedwidth6and8	238128	**0.00319**	-27.6%	-43.4%	-85.1%	-89.7%	-97.8%
albert2D	1039360	0.0716	-12.3%	**+0.1%**	-42.5%	-34.0%	-67.3%
amb	960000	0.123	-5.5%	**+0.1%**	-40.9%	-33.8%	-67.2%
Bach_CPE-Sonata...	513744	**0.0754**	-4.7%	-0.4%	-21.2%	-18.8%	-20.5%
Bobbys_letter...	1144800	0.124	-1.0%	**+0.2%**	-10.2%	-9.7%	-12.3%
ccitt4small	1026432	0.187	-5.4%	+1.0%	-12.0%	-10.4%	**+5.9%**
ccitt7small	1026432	0.191	-2.5%	+0.2%	-8.1%	-10.6%	**+0.9%**
chinese_text	220604	0.364	-1.7%	**+0.2%**	-5.6%	-10.0%	-7.0%
cmfugue1-0	248832	0.107	-4.3%	**+0.7%**	-20.7%	-22.1%	-20.7%
flakes006...	401348	0.506	-0.2%	**+0.0%**	-4.5%	-9.5%	-6.0%
HALFTONE	223776	0.201	-32.7%	**+1.3%**	-59.7%	-55.4%	-62.1%
Halftone2	259081	0.121	-2.5%	**+0.0%**	-24.3%	-14.4%	-26.3%
Halftone3	259081	0.103	-6.4%	**+0.0%**	-38.6%	-37.6%	-60.6%
hamilton_bw	360000	**0.320**	-4.1%	0.0%	-10.9%	-15.9%	-17.3%
hamilton_ed	360000	0.629	-1.0%	**+0.1%**	-4.7%	-9.8%	-13.8%
hieroglyph	225970	0.341	-1.9%	**+0.2%**	-5.8%	-12.8%	-12.0%
leeleter	1389660	0.177	-1.3%	**+0.3%**	-10.1%	-8.3%	-10.1%
lena_j	262144	0.648	-10.0%	**+0.0%**	-22.3%	-13.9%	-24.6%
otoosfont12	484704	0.191	-3.4%	**+0.3%**	-46.5%	-30.7%	-9.0%
otoosfont24	484704	0.123	-8.4%	0.0%	-51.0%	-39.7%	**+56.1%**
pep_j	262144	**0.620**	-11.3%	-0.2%	-22.8%	-13.7%	-29.9%
texmos1.p512M	262144	**0.643**	0.0%	-0.2%	-4.4%	-11.1%	-8.9%
wallpaper003...	401348	**0.484**	-0.9%	-0.4%	-4.0%	-9.1%	-13.3%
wallpaper004...	401348	**0.515**	-2.3%	-0.1%	-5.6%	-10.7%	-17.6%
wallpaper010...	401348	**0.481**	-0.7%	-0.7%	-3.0%	-7.8%	-14.5%
writing	249500	**0.434**	-0.5%	-0.2%	-4.7%	-10.3%	-10.2%

Table 8.5: ERGTO$^{K-WLSTM}$ code length results for $K = 1024$ on the non-synthetic binary images of the test set in comparison to other compression methods. Differences with each scheme C are calculated as $(L_{ERGTO}-L_C)/L_C$, expressed as a percentage. DITO$_D$ stands for DITO$_D^{K-WLSTM}$. Negative numbers represent gains in compression rate, and larger magnitude numbers represent larger gains.

In the case of halftoned images, we see that in every case, our sparse modeling algorithms are superior and sometimes give significant gains in code length. This is consistent with what authors observe in [SIH01]. We also observe that for music score images, there are great differences in favor of sparse models, which is explained by the ability of K-WLSTMs to capture the alternation of different regular structures. As shown before, because of the great regularity of image "1.5.02M" (Figure A.3) that can be captured with some distant dependency locations, the best found K-WLSTM gives large gains in compression rates: 47.8% in comparison to JBIG2 and 28.4% in comparison to contiguous trees.

In the case of other textures and thresholded images considered, sparsity is generally less useful since most of the dependencies are found in close locations, except in the case of "hamilton_bw" (Figure A.17) and "hamilton_ed" (Figure A.18) where K-WLSTMs capture the structure of the evenly spaced lines and patterns.

8.5.2 Results on generated images

As an empirical consistency check, we compare the performance of ERGTO$^{K-WLSTM}$ on generated images based on trained K-WLSTMs against the code length given by the generating template and the results given by other compression methods. Table 8.6 shows that the results of ERGTO$^{K-WLSTM}$ are quite close to the code length given by the generating template and are significantly better than the results given by other methods as expected for this type of images specifically generated by K-WLSTMs.

Image	Image size (in bits, at 1 bit/pixel)	ERGTO normalized code length (in bits/pixel)	K for ERGTO	vs. generating template	vs. contiguous tree $K \approx 64$	vs. jbig	vs. jbig2	vs. djvu
GEN_ A-fixedwidth6and8	250000	**0.0773**	128	-1.1%	-43.4%	-64.5%	-60.4%	-78.3%
GEN_ cmfugue	250000	0.186	1024	**+0.3%**	-8.6%	-18.4%	-26.6%	-19.5%
GEN_ otoosfont12	250000	0.302	512	**+0.6%**	-3.7%	-33.7%	-32.7%	-48.3%

Table 8.6: ERGTO$^{K-WLSTM}$ code length results on binary images generated by K-WLSTMs trained on binary images (see Subsection 4.4.2) in comparison to other compression methods and to the code length given by the generating template. Differences with each scheme C are calculated as $(L_{ERGTO}-L_C)/L_C$, expressed as a percentage. Negative numbers represent gains in compression rate, and larger magnitude numbers represent larger gains.

8.6 Conclusions

In this chapter, we demonstrated the great compression potential of K-WLSTMs with large window sizes over smaller subclasses of models, i.e., contiguous models, K-SCMs and K-WLSTMs with smaller window sizes. We observed that these types of class generalizations carry, at most, a small overhead in code length and, in most cases, it is negligible. Finally, we showed that this increased compression potential in combination with our improved algorithm ERGTO$^{K-WSLTM}$ generally outperforms standard compression methods and sometimes by significant margins.

Chapter 9

Conclusions and future work

In this work, we proposed a greedy algorithm DITO and an enhanced genetic algorithm ERGTO for modeling and coding with sparse context models. DITO is quite efficient even for large window sizes and also in the case of K-WLSTMs, which are more complex to be evaluated. Nevertheless, we found that, due to its deterministic nature, it is prone to getting stuck in some bad local minima.

ERGTO is an improvement of the algorithm BRGTO of [Ser04] (originally only for K-SCMs) and was designed by taking guidance from some properties that make the efficiency of DITO while keeping the randomness properties that help in preventing getting stuck in local minima. Although not as fast as DITO, ERGTO runs significantly faster (usually, more than 30$\times$) than BRGTO for K-SCMs but also is practical for large values of K and for K-WLSTMs.

The efficiency improvements of ERGTO$^{K-WLSTM}$ lead to better compression results that outperform those of tree models with gains sometimes larger than 30% and, in many cases, those of standard methods with gains sometimes larger than 55% (for the three competitors considered).

It is important to emphasize that the algorithms presented in this thesis for the sparse template optimization problem are not restricted to the binary images application. Additionally, the extensive experimentation over binary images gave clues of what type of structures in the data can be captured by these models. This can be useful for the search of other data types for which these models could suitable (even when the goal is not compression), which is not a trivial task since, in many common cases, the most important dependencies are located within a small contiguous window (e.g., continuous-tone natural images in which color intensities change in a smooth way).

This work was essentially of an experimental nature and, thus, future work directions could be oriented to further understanding of the theoretical properties of these models.

Bibliography

[Bäc96] T. Bäck, *Evolutionary algorithms in theory and practice*, Oxford University Press New York, 1996. 55

[Bar54] N.A. Barricelli, *Esempi numerici di processi di evoluzione*, Methodos **6** (1954), 21–22. 19

[BG+00] A.C. Bovik, J.D. Gibson, et al., *Al Bovik, Handbook of Image and Video Processing*, 2000. 30

[BHH+98] L. Bottou, P. Haffner, P.G. Howard, P. Simard, Y. Bengio, and Y. LeCun, *High quality document image compression with DjVu*, Journal of Electronic Imaging **7** (1998), 410–425. 31

[Bla09] P.E. Black, *metaheuristic*, `http://www.itl.nist.gov/div897/sqg/dads/HTML/metaheuristic.html`, 2009. 19

[BR04] P.Y. Bourguignon and D. Robelin, *Modèles de Markov parcimonieux: sélection de modèle et estimation*, Proceedings of JOBIM Congress, 2004. 4

[Bre57] L. Breiman, *The individual ergodic theorem of information theory*, The Annals of Mathematical Statistics (1957), 809–811. 33

[BRY98] A. Barron, J. Rissanen, and B. Yu, *The minimum description length principle in coding and modeling*, IEEE Transactions on Information Theory **44** (1998), no. 6, 2743–2760. 12

[BSPV02] M. Birattari, T. Stutzle, L. Paquete, and K. Varrentrapp, *A racing algorithm for configuring metaheuristics*, GECCO, 2002, pp. 11–18. 55

[BW99] P. Buhlmann and A.J. Wyner, *Variable length Markov chains*, Annals of Statistics (1999), 480–513. 2

[Cor01] T.H. Cormen, *Introduction to algorithms*, MIT press, 2001. 40

[CT06] T.M. Cover and J.A. Thomas, *Elements of information theory*, Wiley-Interscience, 2006. 11, 14, 33

[Dar59] C. Darwin, *On the origin of species*, John Murray, London, 1859. 19

[Doo53] J.L. Doob, *Stochastic processes*, John Wiley & Sons, New York, 1953. 33, 34

[FLSV] N. Fraiman, A. Lhéritier, G. Seroussi, and A. Viola, *Lossless compression for sparse finite memory sources*, In Information Theory and applications (ITA'08), San Diego, CA, USA, January 2008. ii, iv, 4

[Fra08] N. Fraiman, *Model selection techniques & sparse markov chains*, Master's thesis, Universidad de la República, 2008. ii, iv, 4, 32

[FSV08] N. Fraiman, G. Seroussi, and A. Viola, *Lossless source coding via sparse tree models*, unpublished, 2008. ii, iv, 4, 32

[FWA04] S. Forchhammer, X. Wu, and J.D. Andersen, *Optimal context quantization in lossless compression of image data sequences*, IEEE Transactions on Image Processing **13** (2004), no. 4, 509–517. ii, iv, 3

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability; a guide to the theory of np-completeness*, W.H. Freeman, 1979. 19

[Gre86] J.J. Grefenstette, *Optimization of control parameters for genetic algorithms*, IEEE Transactions on Systems, Man and Cybernetics **16** (1986), no. 1, 122–128. 55

[Hol75] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, 1975. 19

[Hsi04] P. Hsieh, *Superfasthash function*, `http://www.azillionmonkeys.com/qed/hash.html`, 2004. 40

[Int88] International Telecommunication Union, *Facsimile coding schemes and coding control functions for group 4 facsimile apparatus*, ITU-T Recommendation T.6, November 1988. 30

[Int96] ———, *Standardization of group 3 facsimile terminals for document transmission*, ITU-T Recommendation T.4, July 1996. 30

[Joi93] Joint Bi-level Image experts Group (JBIG), *Information technology — coded representation of picture and audio information — progressive bi-level image compression*, ISO/IEC 11544 and ITU-TRec. T. 82, 1993. 3, 28

[Joi01] ———, *Information technology — coded representation of picture and audio information — lossy/lossless coding of bi-level images*, ISO/IEC 14492 and ITU T.88, 2001. 3, 28

[Knu69] Donald E. Knuth, *The art of computer programming: Volume 2, seminumerical algorithms*, Addison-Wesley Publishing Company, 1969. 21

[Kra49] L.G. Kraft, *A device for quantizing, grouping and coding amplitude-modulated pulses*, Master's thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1949. 10

[KT81] R. Krichevsky and V. Trofimov, *The performance of universal encoding*, IEEE Transactions on Information Theory **27** (1981), no. 2, 199–207. 12

[Liz05] LizardTech, *Djvu reference v3*, `http://djvu.org/docs/DjVu3Spec.djvu`, 2005. 28

[Lub96] Michael Luby, *Pseudorandomness and cryptographic applications*, Princeton University Press, 1996. 21

[Mac03] D.J.C. MacKay, *Information theory, inference and learning algorithms*, Cambridge University Press, 2003. 10, 13

[Mar09] A. Martín, *Tree models: Algorithms and information theoretic properties*, Ph.D. thesis, Universidad de la República, 2009. 17

[McM53] B. McMillan, *The basic theorems of information theory*, The Annals of Mathematical Statistics (1953), 196–219. 33

[McM56] ______, *Two inequalities implied by unique decipherability*, Information Theory, IRE Transactions on **2** (1956), no. 4, 115–116. 10

[MF98] N. Merhav and M. Feder, *Universal prediction*, IEEE Transactions on Information Theory **44** (1998), 2124–2147. 15

[MSW04] Alvaro Martín, Gadiel Seroussi, and Marcelo J. Weinberger, *Linear time universal coding and time reversal of tree sources via FSM closure*, IEEE Transactions on Information Theory **50** (2004), no. 7, 1442–1468. 2, 16

[Noh93] R. Nohre, *Topics in descriptive complexity*, Ph.D. thesis, Technical University of Linkoping, 1993. 2, 15, 16

[Pas76] R.C. Pasco, *Source coding algorithms for fast data compression*, Ph.D. thesis, Citeseer, 1976. i, iii, 10

[PMLJA88] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon Jr, and R.B. Arps, *An overview of the basic principles of the Q-coder adaptive binary arithmetic coder*, IBM Journal of research and development **32** (1988), no. 6, 717–726. 30

[Ris76] J. Rissanen, *Generalized Kraft inequality and arithmetic coding*, IBM Journal of Research and Development **20** (1976), no. 3, 198–203. i, iii, 10

[Ris78] ______, *Modeling by shortest data description*, Automatica **14** (1978), 465–471. 1

[Ris83] ______, *A universal data compression system*, IEEE Transactions on information theory **29** (1983), no. 5, 656–664. 2, 15

[Ris84] ______, *Universal coding, information, prediction, and estimation*, IEEE Transactions on Information Theory **30** (1984), 629–636. i, iii, 1, 13

[Ris87] ______, *Stochastic complexity*, Journal of the Royal Statistical Society. Series B (Methodological) (1987), 223–239. 1

[Ris96] ______, *Fisher information and stochastic complexity*, IEEE Transactions on Information Theory **42** (1996), no. 1, 40–47. 1, 11

[RSP08] D. Rother, G. Sapiro, and V. Pande, *Statistical characterization of protein ensembles*, IEEE/ACM Trans. Comput. Biology Bioinform **5** (2008), no. 1, 42–55. ii, iv, 4, 43

[Ser04] G. Seroussi, *A genetic algorithm for optimizing context patterns*, draft and software implementation, 2004. ii, iv, 5, 7, 19, 22, 23, 40, 57, 60, 62, 75

[Sha48] C.E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27** (1948), 379–423 & 623–656. 1, 9, 10, 33

[Sht87] Y.M. Shtarkov, *Universal sequential coding of single messages*, Problemy Peredachi Informatsii **23** (1987), no. 3, 3–17. 11

[SIH01] H. Sakanashi, M. Iwata, and T. Higuchi, *A lossless compression method for halftone images using evolvable hardware*, Evolvable systems: from biology to hardware: 4th International Conference, ICES 2001, Tokyo, Japan, October 3-5, 2001: proceedings, Springer Verlag, 2001, p. 314. ii, iv, 3, 6, 31, 72

[Suz95] J. Suzuki, *A CTW scheme for some FSM models*, 1995 IEEE International Symposium on Information Theory, 1995. Proceedings., 1995. 4

[Uli87] R. Ulichney, *Digital halftoning*, MIT press, 1987. 28

[VW96] P.A.J. Volf and F.M.J. Willems, *Context-tree weighting for extended tree sources*, 17th Symp. on Information Theory in the Benelux (Enschede (NL)) (G. H. L. M. Heideman, ed.), Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL), May30-31 1996, pp. 95–102. 4

[Whi94] D. Whitley, *A genetic algorithm tutorial*, Statistics and computing **4** (1994), no. 2, 65–85. 19

[Wil98] F.M.J. Willems, *The context-tree weighting method: Extensions*, IEEE Transactions on Information Theory **44** (1998), no. 2, 792–798. 2, 15

[WNC87] I.H. Witten, R.M. Neal, and J.G. Cleary, *Arithmetic coding for data compression*, Communications of the ACM **30** (1987), no. 6, 520–540. 11

[WRF95] M.J. Weinberger, J. Rissanen, and M. Feder, *A universal finite memory source*, IEEE Transactions on Information Theory **41** (1995), no. 3, 643–652. 2, 15

[WST95] F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens, *The context-tree weighting method: Basic properties*, IEEE Transactions on Information Theory **41** (1995), no. 3, 653–664. 2, 15, 16

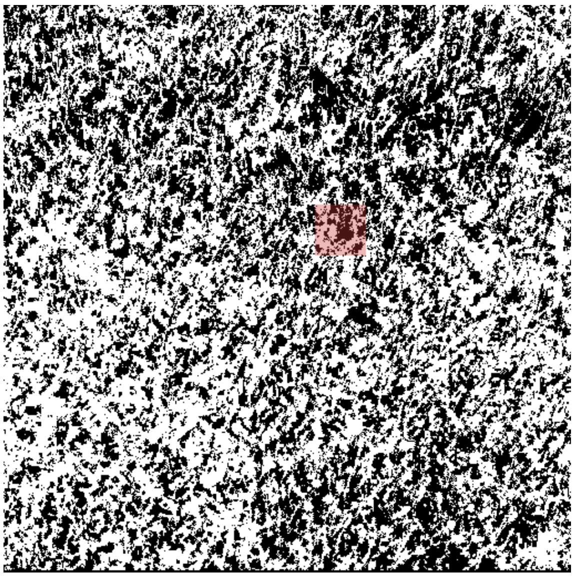
[YG07] B. Yuan and M. Gallagher, *Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks*, Parameter Setting in Evolutionary Algorithms, Series of Studies in Computational Intelligence (SCI) **54** (2007). 55

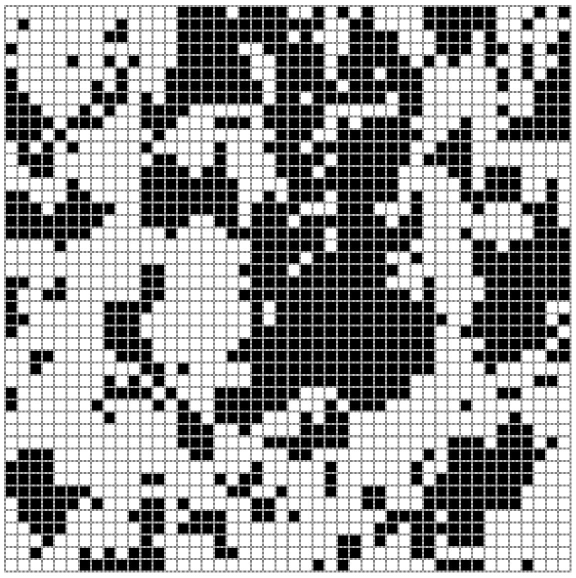
[ZHS05] X. Zhao, H. Huang, and T.P. Speed, *Finding short DNA motifs using permuted Markov models*, Journal of Computational Biology **12** (2005), no. 6, 894–906. 4

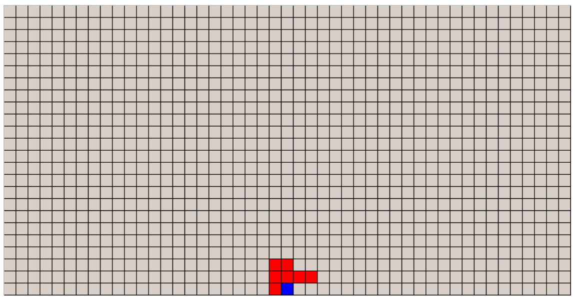
Appendix A

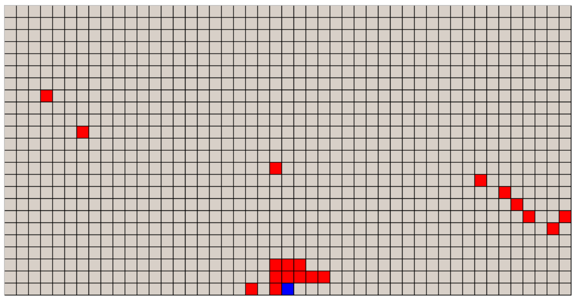
Image set and best sparse and contiguous results

In this appendix, we show each image of the test set described in 4.5. For each image, a zoomed portion (which is shaded on the full image) is shown. Additionally, we show the best sparse and contiguous (with $K \approx 64$, see Appendix D) results and the corresponding template characteristics. The darkest square in each template figure represents the current sample.


(a) Image


(b) Zoomed portion

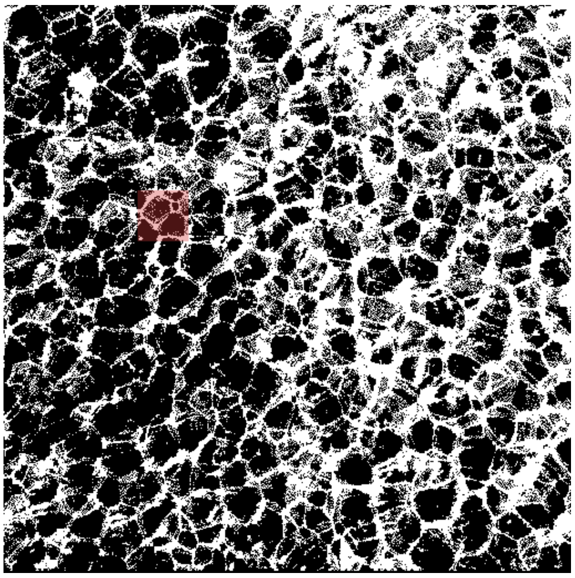

(c) Best K-SCM template


(d) Best K-WLSTM template

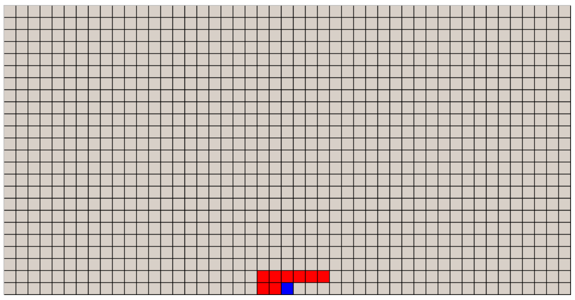
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	9	7	6.40E-01
K-WLSTM	975	19	6.40E-01
Fixed-length contiguous	8	8	6.42E-01
Tree model	10	10	6.40E-01

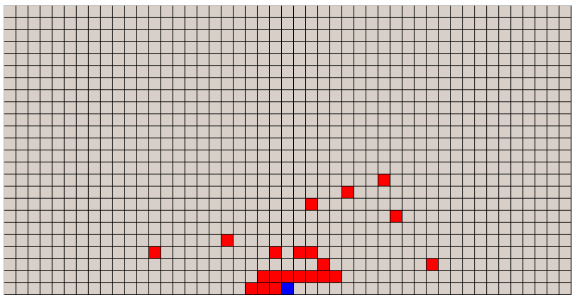
(e) Actual window size, weight and code length of the best templates

Figure A.1: Image "1.1.01M", 512 rows x 512 columns


(a) Image

(b) Zoomed portion

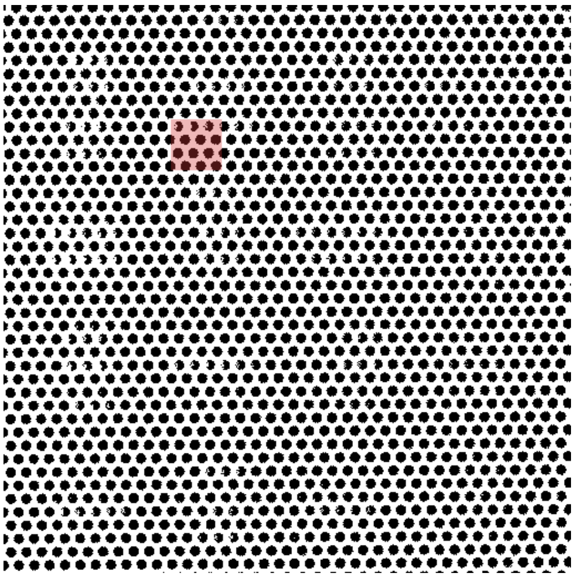

(c) Best K-SCM template

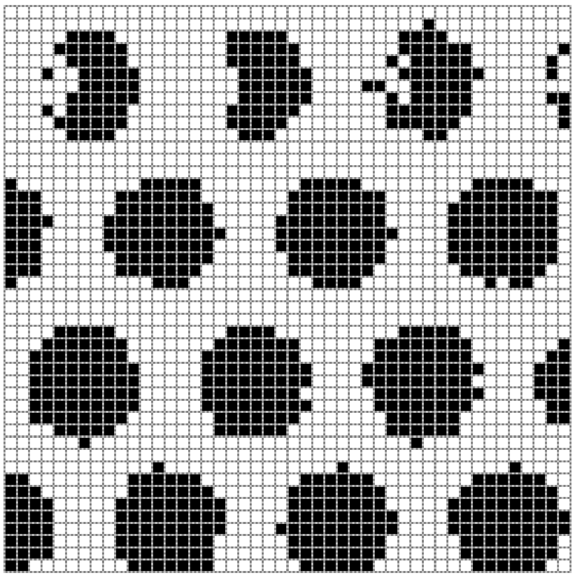

(d) Best K-WLSTM template

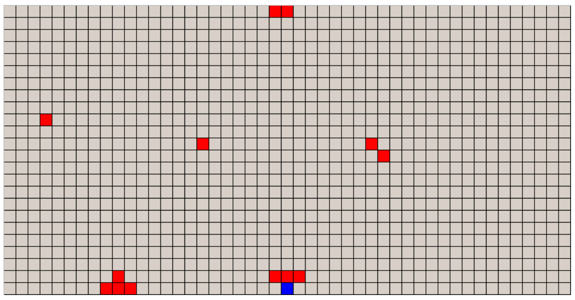
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	24	16	5.45E-01
K-WLSTM	234	21	5.43E-01
Fixed-length contiguous	8	8	5.46E-01
Tree model	26	26	5.43E-01

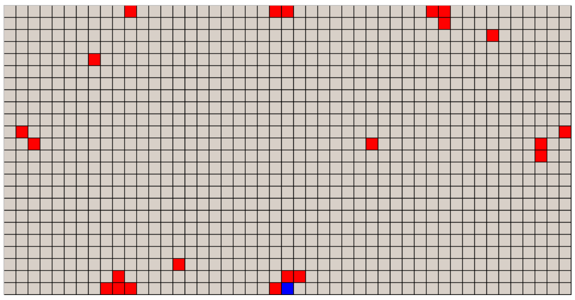
(e) Actual window size, weight and code length of the best templates

Figure A.2: Image "1.1.13M", 512 rows x 512 columns


(a) Image


(b) Zoomed portion

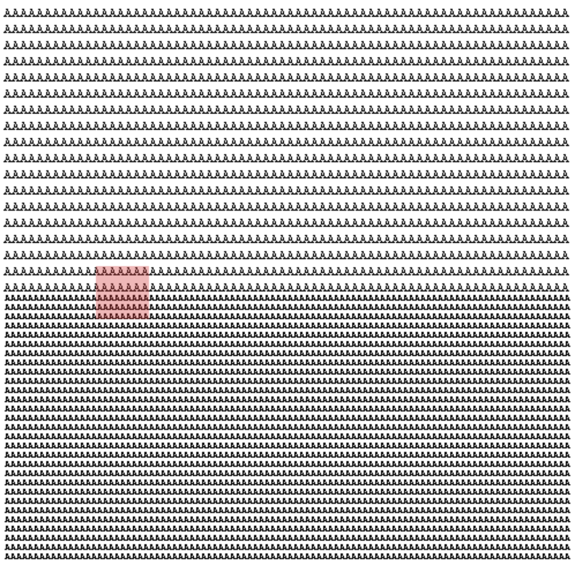

(c) Best K-SCM template

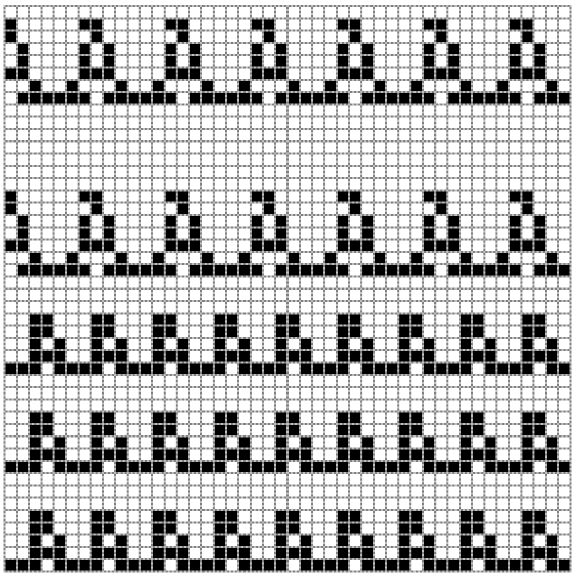

(d) Best K-WLSTM template

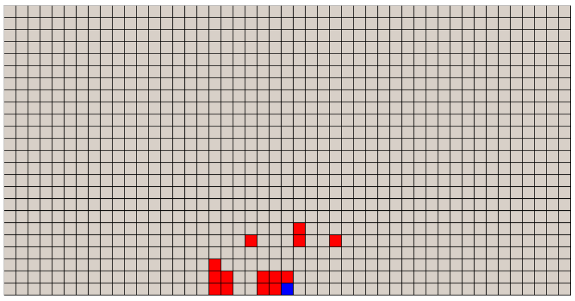
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	921	13	1.46E-01
K-WLSTM	1024	22	1.45E-01
Fixed-length contiguous	24	24	2.63E-01
Tree model	63	63	2.05E-01

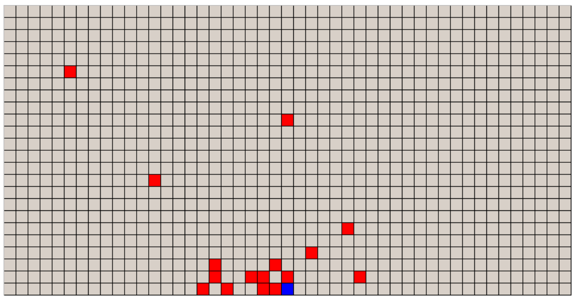
(e) Actual window size, weight and code length of the best templates

Figure A.3: Image "1.5.02M", 512 rows x 512 columns


(a) Image


(b) Zoomed portion

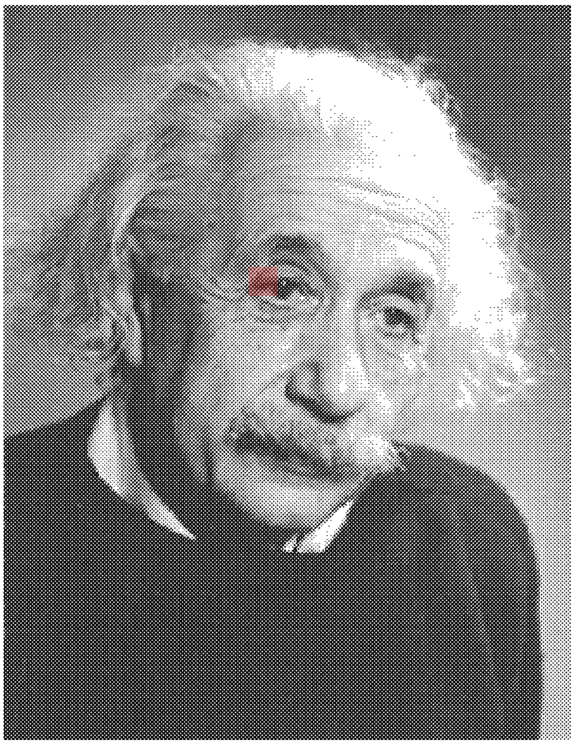

(c) Best K-SCM template

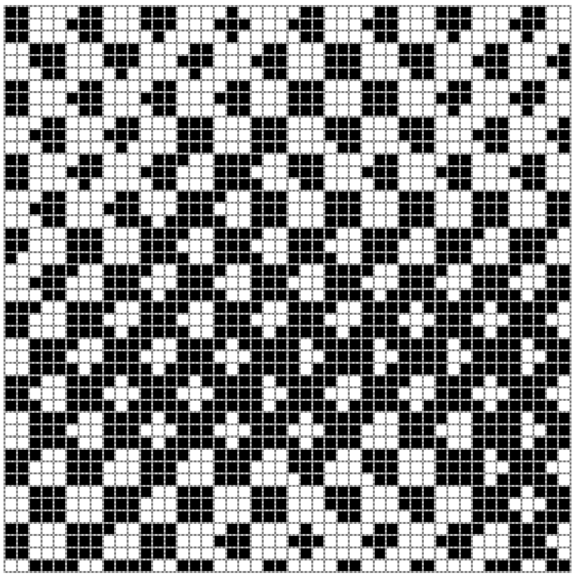

(d) Best K-WLSTM template

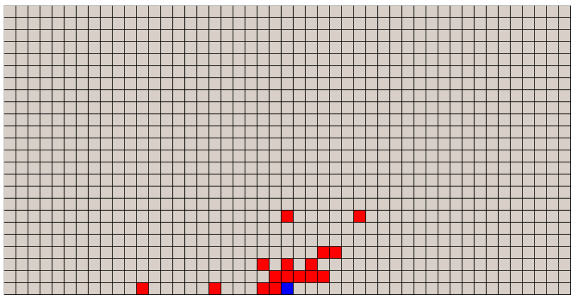
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	61	14	4.94E-03
K-WLSTM	961	16	3.19E-03
Fixed-length contiguous	55	55	7.48E-03
Tree model	56	56	4.68E-03

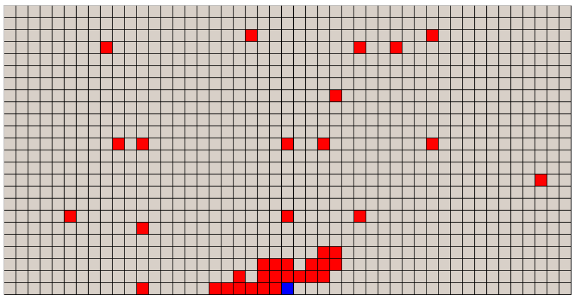
(e) Actual window size, weight and code length of the best templates

Figure A.4: Image “A-fixedwidth6and8”, 484 rows x 492 columns


(a) Image


(b) Zoomed portion


(c) Best K-SCM template

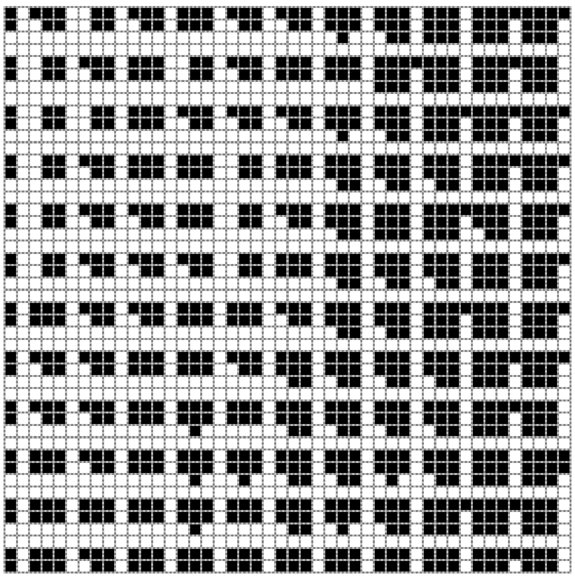

(d) Best K-WLSTM template

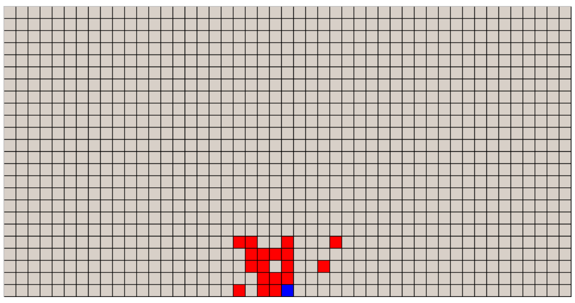
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	219	16	7.70E-02
K-WLSTM	947	38	7.15E-02
Fixed-length contiguous	30	30	1.01E-01
Tree model	62	62	8.17E-02

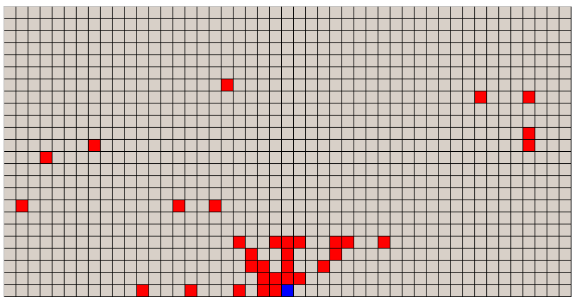
(e) Actual window size, weight and code length of the best templates

Figure A.5: Image "albert2D", 1160 rows x 896 columns

(a) Image


(b) Zoomed portion

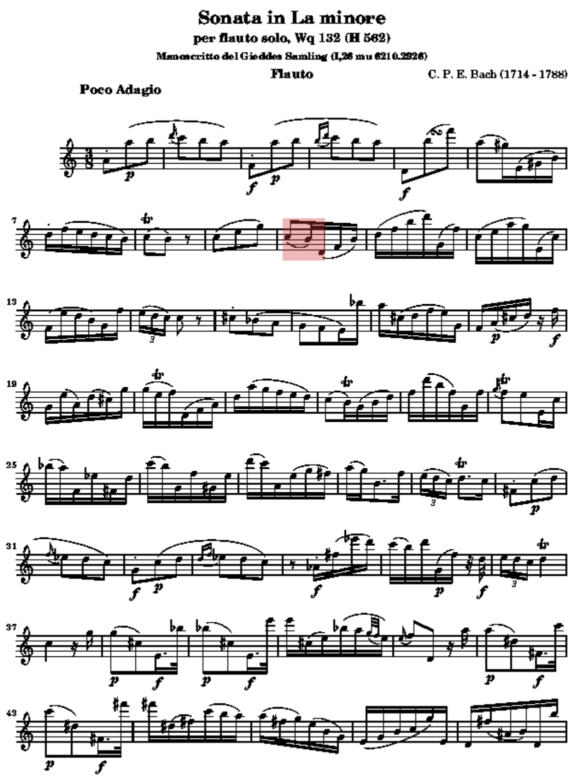

(c) Best K-SCM template


(d) Best K-WLSTM template

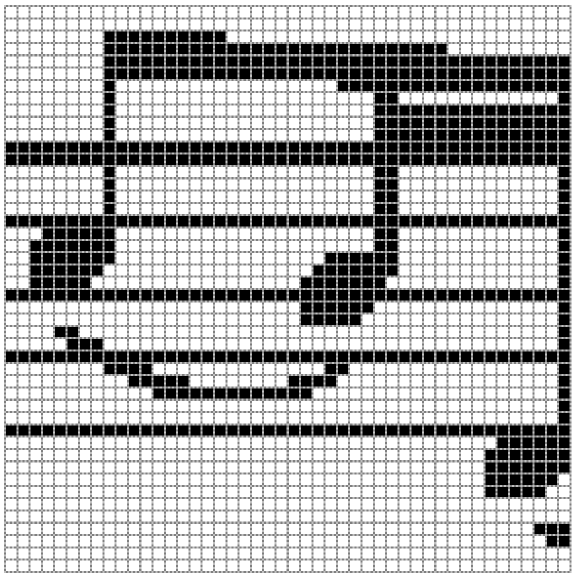
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	50	18	1.28E-01
K-WLSTM	976	33	1.23E-01
Fixed-length contiguous	24	24	1.43E-01
Tree model	52	52	1.30E-01

(e) Actual window size, weight and code length of the best templates

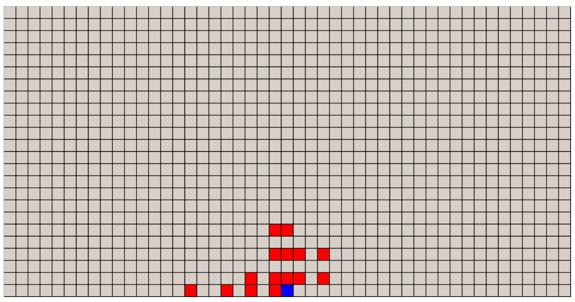
Figure A.6: Image “amb”, 1200 rows x 800 columns


Sonata in La minore
per flauto solo, Wq 132 (H 562)
Manoscritto del Giedde Samling (I,26 mu 6210.2926)
Flauto
C. P. E. Bach (1714 - 1788)
Poco Adagio

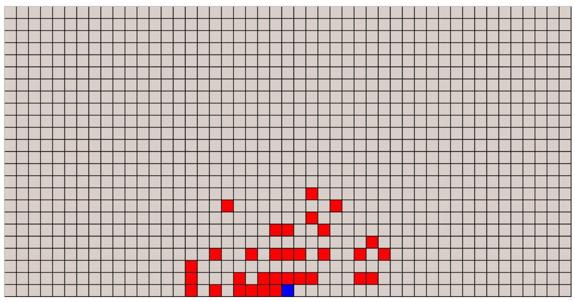
(a) Image



(b) Zoomed portion



(c) Best K-SCM template

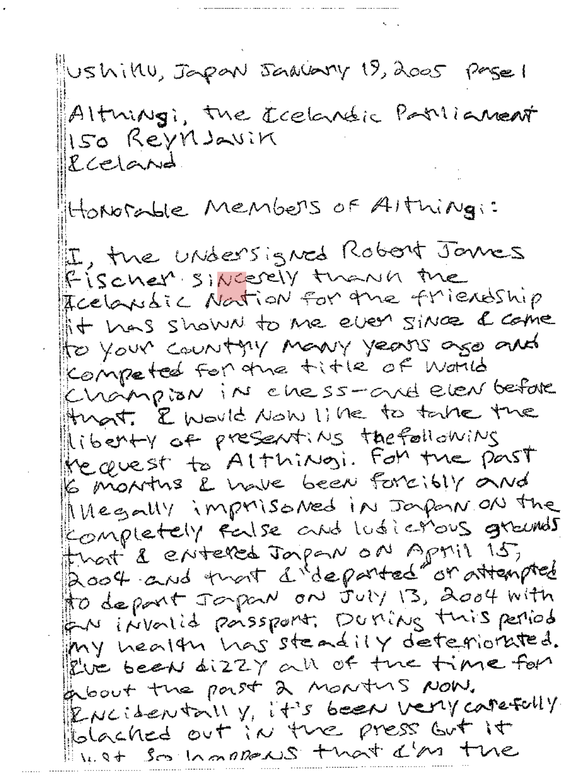


(d) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-SCM	97	15	8.18E-02
K-WLSTM	1013	34	7.54E-02
Fixed-length contiguous	18	18	8.63E-02
Tree model	61	61	7.93E-02

(e) Actual window size, weight and code length of the best templates

Figure A.7: Image “Bach_CPE-Sonata_flauto_solo_La_min-fl”, 834 rows x 616 columns



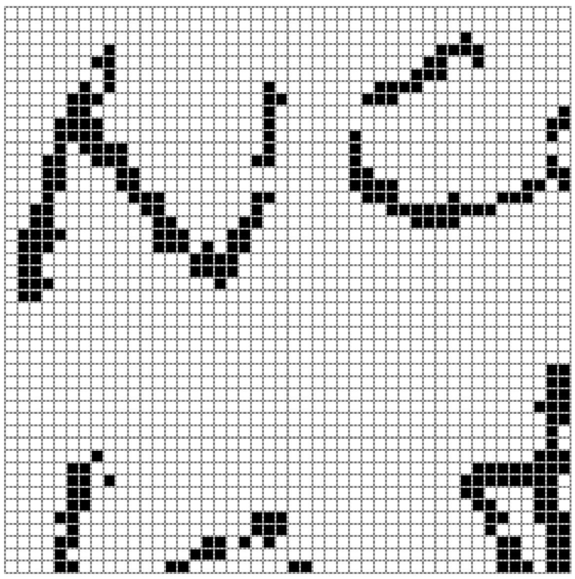
Ushiku, Japan January 19, 2005 Page 1

Althingi, the Icelandic Parliament
150 Reykjavik
Iceland

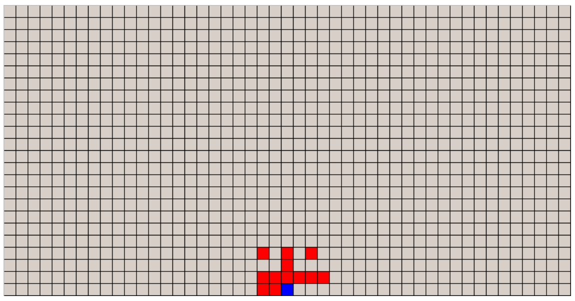
Honorable Members of Althingi:

I, the undersigned Robert James
Fischer sincerely thank the
Icelandic Nation for the friendship
it has shown to me ever since I came
to your country many years ago and
competed for the title of World
Champion in chess—and even before
that. I would now like to take the
liberty of presenting the following
request to Althingi. For the past
6 months I have been forcibly and
illegally imprisoned in Japan on the
completely false and ludicrous grounds
that I entered Japan on April 15,
2004 and that I "departed" or attempted
to depart Japan on July 13, 2004 with
an invalid passport. During this period
my health has steadily deteriorated.
I've been dizzy all of the time for
about the past 2 months now.
Incidentally, it's been very carefully
blacked out in the press but it
just so happens that I'm the

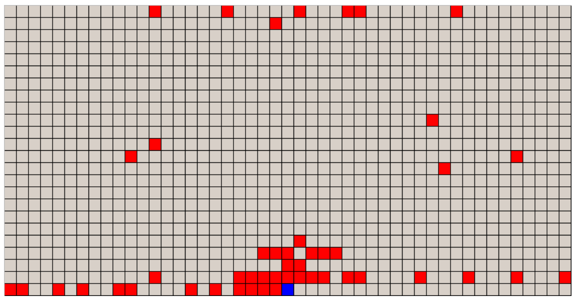
(a) Image



(b) Zoomed portion



(c) Best K-SCM template

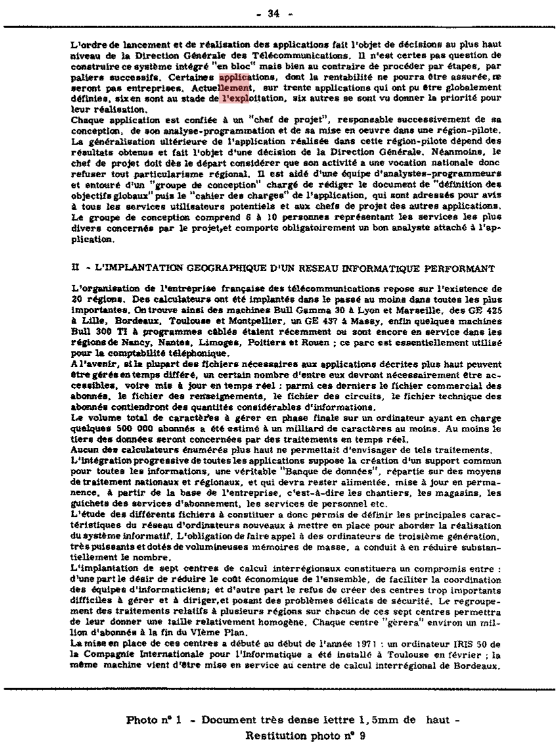


(d) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-SCM	22	12	1.27E-01
K-WLSTM	1020	48	1.23E-01
Fixed-length contiguous	10	10	1.29E-01
Tree model	51	51	1.25E-01

(e) Actual window size, weight and code length of the best templates

Figure A.8: Image "Bobbys_letter_page_1", 1272 rows x 900 columns



- 34 -

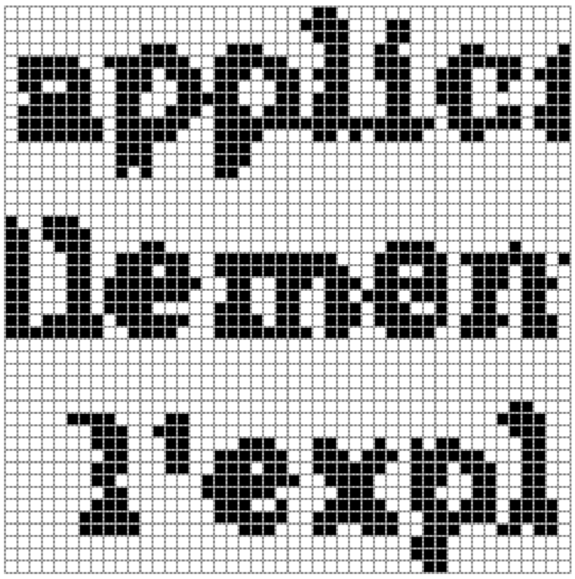
L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet, et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

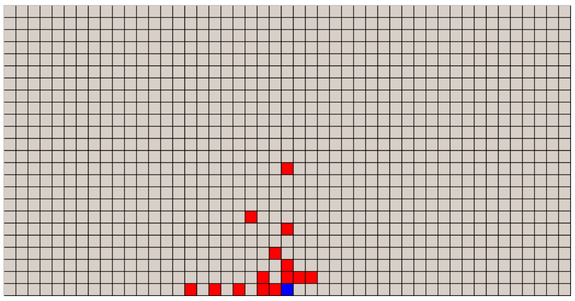
L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger, et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Photo n° 1 - Document très dense lettre 1,5mm de haut -
Restitution photo n° 9

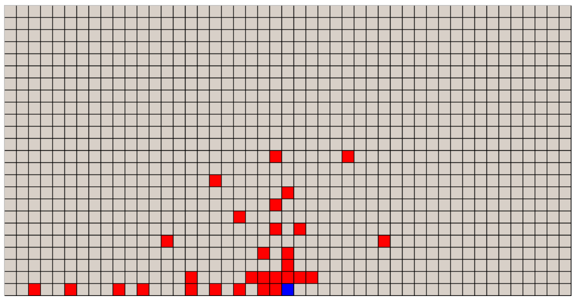
(a) Image



(b) Zoomed portion



(c) Best K-SCM template



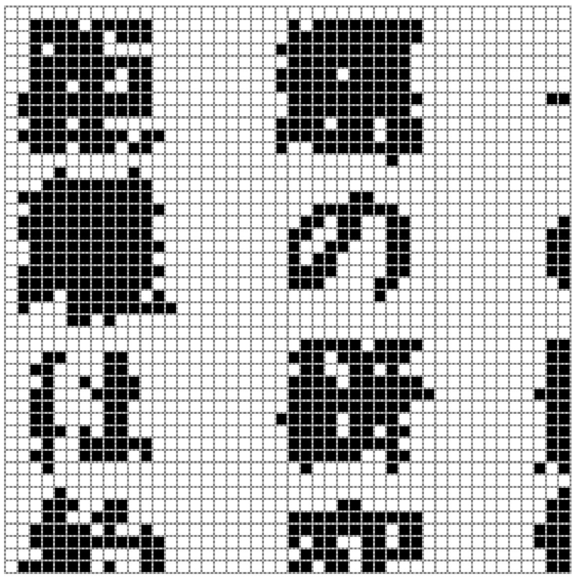
(d) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-SCM	158	14	1.96E-01
K-WLSTM	685	29	1.85E-01
Fixed-length contiguous	14	14	2.10E-01
Tree model	58	58	1.97E-01

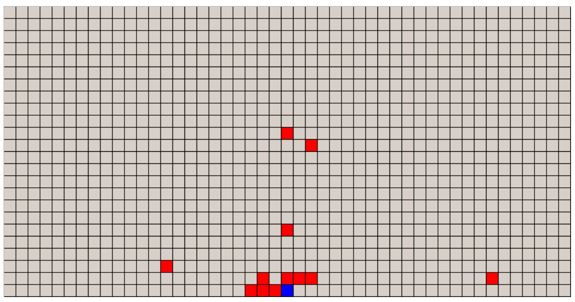
(e) Actual window size, weight and code length of the best templates

Figure A.9: Image "ccitt4small", 1188 rows x 864 columns

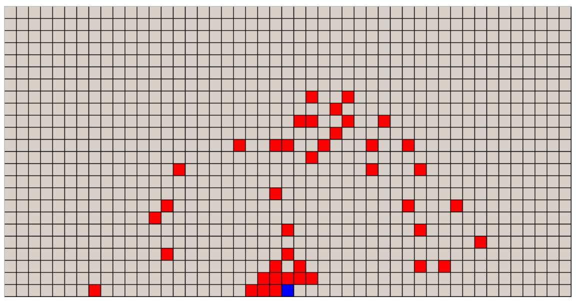
(a) Image



(b) Zoomed portion



(c) Best K-SCM template

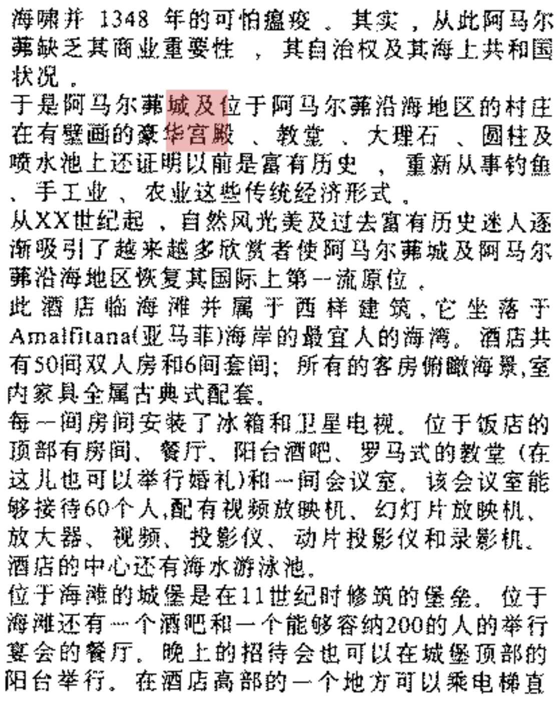


(d) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-SCM	452	12	1.95E-01
K-WLSTM	846	21	1.91E-01
Fixed-length contiguous	10	10	2.04E-01
Tree model	59	59	1.96E-01

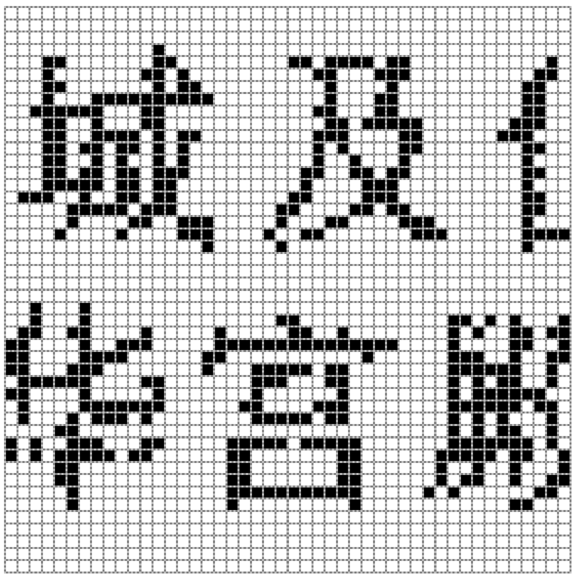
(e) Actual window size, weight and code length of the best templates

Figure A.10: Image “ccitt7small”, 1188 rows x 864 columns

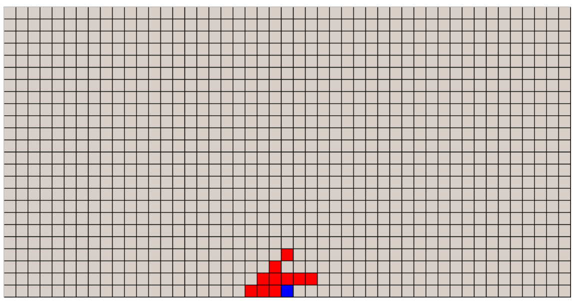


海啸并 1348 年的可怕瘟疫 。 其实 ，从此阿马尔菲缺乏其商业重要性 ， 其自治权及其海上共和国状况 。
于是阿马尔菲城及位于阿马尔菲沿海地区的村庄在有壁画的豪华宫殿 、 教堂 、 大理石 、 圆柱及喷水池上还证明以前是富有历史 ， 重新从事钓鱼 、 手工业 、 农业这些传统经济形式 。
从XX世纪起 ，自然风光美及过去富有历史迷人逐渐吸引了越来越多欣赏者使阿马尔菲城及阿马尔菲沿海地区恢复其国际上第一流原位 。
此 酒 店 临 海 滩 并 属 于 西 样 建 筑 , 它 坐 落 于 Amalfitana(亚马菲)海岸的最宜人的海湾。 酒店共有50间双人房和6间套间; 所有的客房俯瞰海景,室内家具全属古典式配套。
每一间房间安装了冰箱和卫星电视。 位于饭店的顶部有房间、 餐厅、 阳台酒吧、 罗马式的教堂 (在这儿也可以举行婚礼)和一间会议室。 该会议室能够接待60个人,配有视频放映机、 幻灯片放映机、 放大器、 视频、 投影仪、 动片投影仪和录影机。 酒店的中心还有海水游泳池。
位于海滩的城堡是在11世纪时修筑的堡垒。 位于海滩还有一个酒吧和一个能够容纳200的人的举行宴会的餐厅。 晚上的招待会也可以在城堡顶部的阳台举行。 在酒店高部的一个地方可以乘电梯直

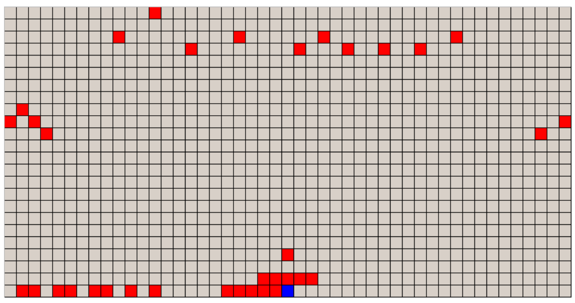
(a) Image



(b) Zoomed portion



(c) Best K-SCM template

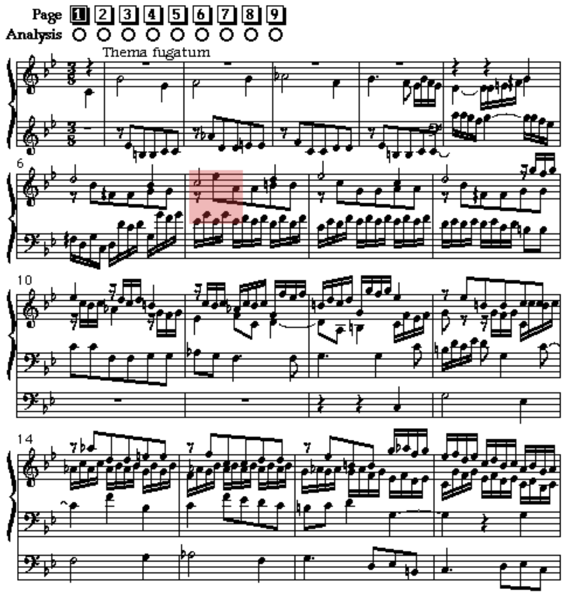


(d) Best K-WLSTM template

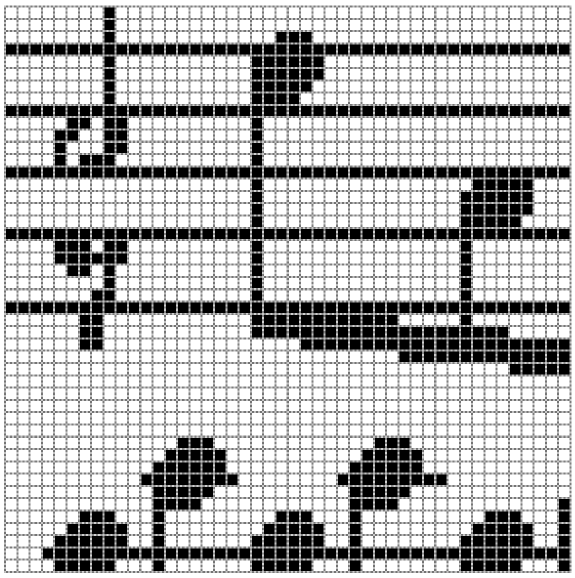
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	14	10	3.71E-01
K-WLSTM	1018	35	3.63E-01
Fixed-length contiguous	10	10	3.75E-01
Tree model	56	56	3.70E-01

(e) Actual window size, weight and code length of the best templates

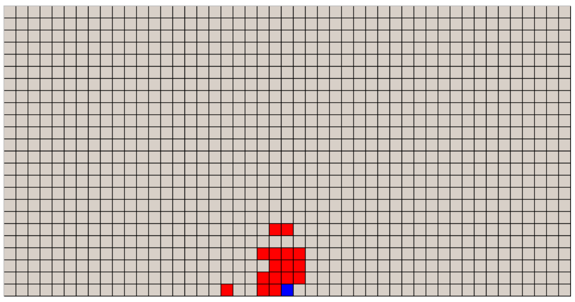
Figure A.11: Image “chinese_text”, 524 rows x 421 columns



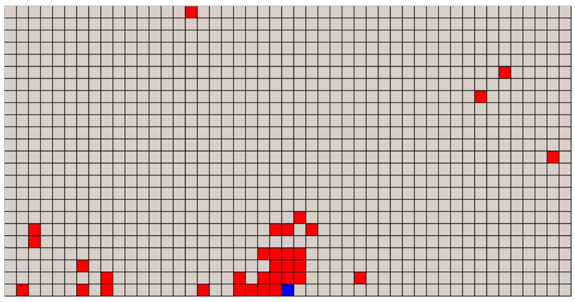
(a) Image



(b) Zoomed portion



(c) Best K-SCM template

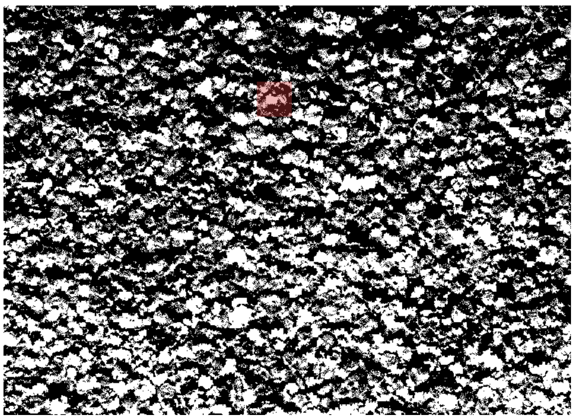


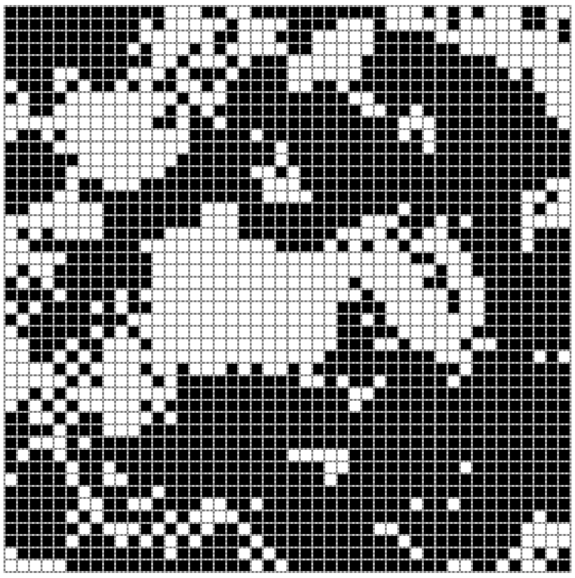
(d) Best K-WLSTM template

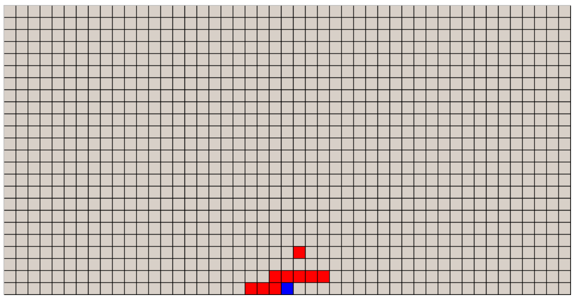
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	43	16	1.17E-01
K-WLSTM	962	33	1.06E-01
Fixed-length contiguous	18	18	1.24E-01
Tree model	56	56	1.12E-01

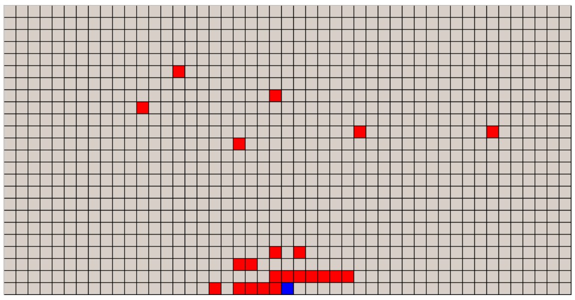
(e) Actual window size, weight and code length of the best templates

Figure A.12: Image “cmfugue1-0”, 512 rows x 486 columns


(a) Image


(b) Zoomed portion

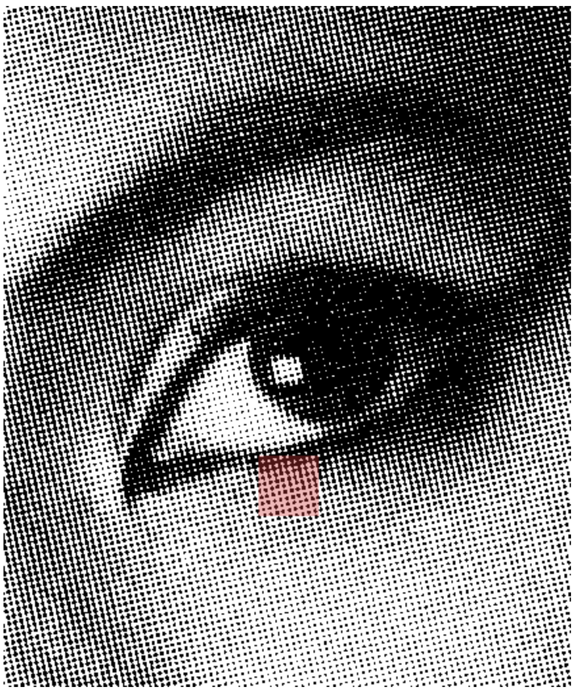

(c) Best K-SCM template

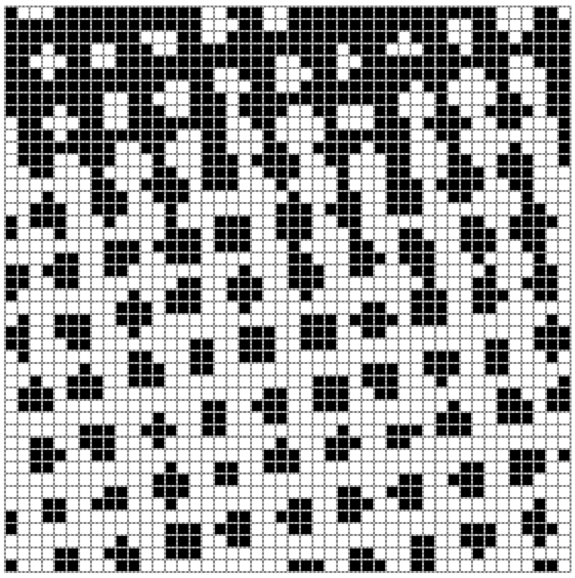

(d) Best K-WLSTM template

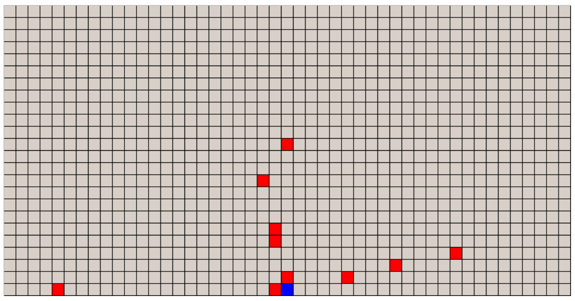
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	18	9	5.07E-01
K-WLSTM	722	22	5.06E-01
Fixed-length contiguous	8	8	5.09E-01
Tree model	45	45	5.07E-01

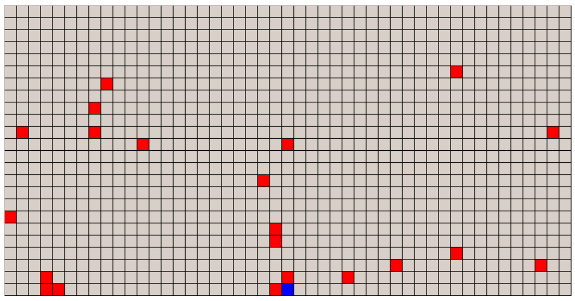
(e) Actual window size, weight and code length of the best templates

Figure A.13: Image "flakes006-inca-100dpi-00M", 538 rows x 746 columns


(a) Image


(b) Zoomed portion


(c) Best K-SCM template

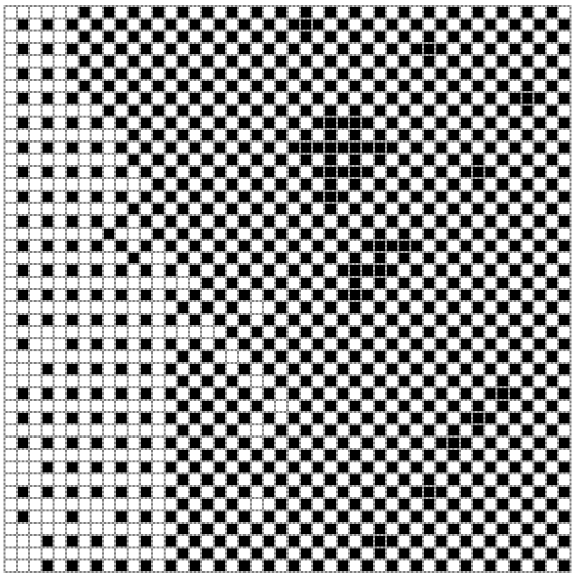

(d) Best K-WLSTM template

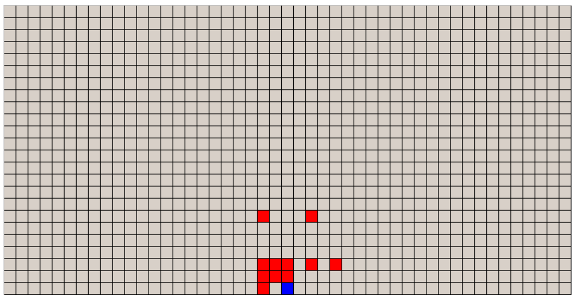
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	563	10	2.02E-01
K-WLSTM	972	21	1.98E-01
Fixed-length contiguous	27	27	3.56E-01
Tree model	43	43	2.99E-01

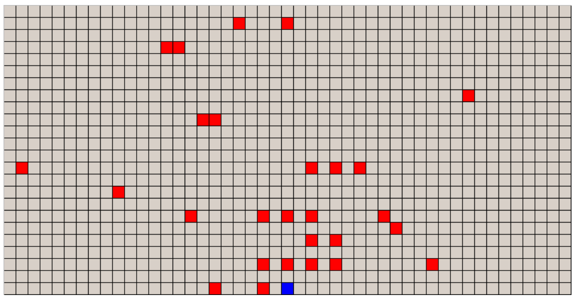
(e) Actual window size, weight and code length of the best templates

Figure A.14: Image "HALFTONE", 518 rows x 432 columns

(a) Image


(b) Zoomed portion


(c) Best K-SCM template

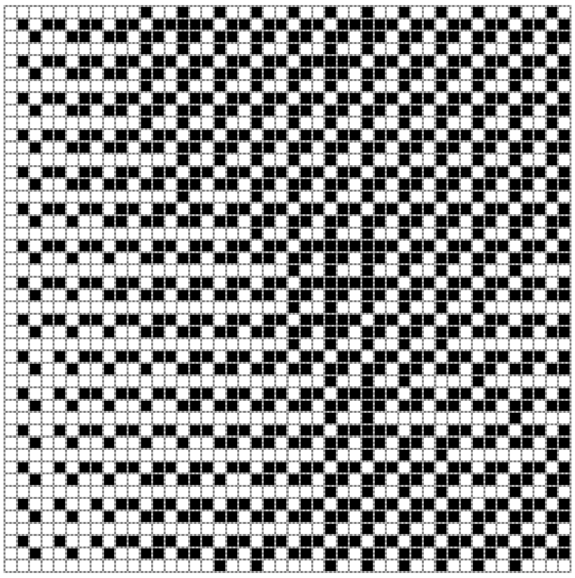

(d) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-SCM	64	11	1.23E-01
K-WLSTM	905	27	1.21E-01
Fixed-length contiguous	12	12	1.28E-01
Tree model	37	37	1.24E-01

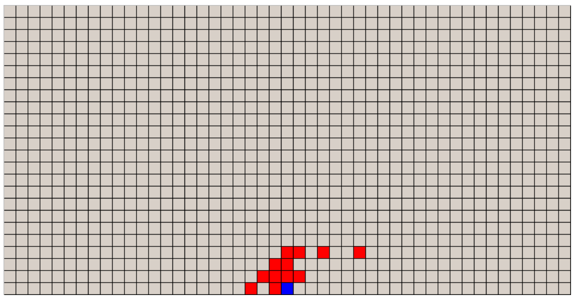
(e) Actual window size, weight and code length of the best templates

Figure A.15: Image “Halftone2”, 509 rows x 509 columns

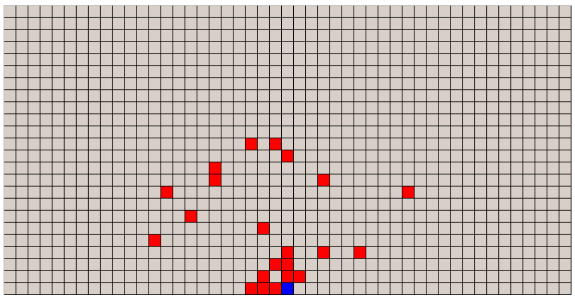
(a) Image



(b) Zoomed portion



(c) Best K-SCM template

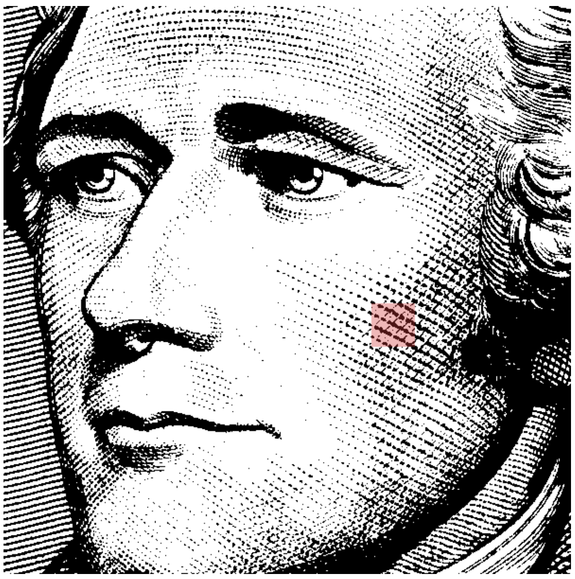


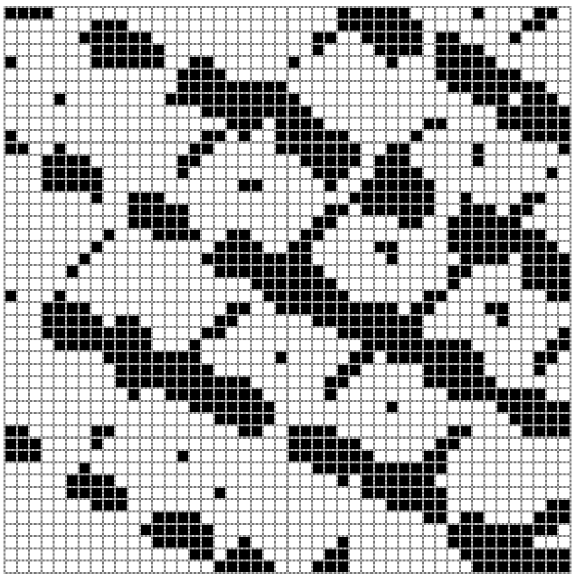
(d) Best K-WLSTM template

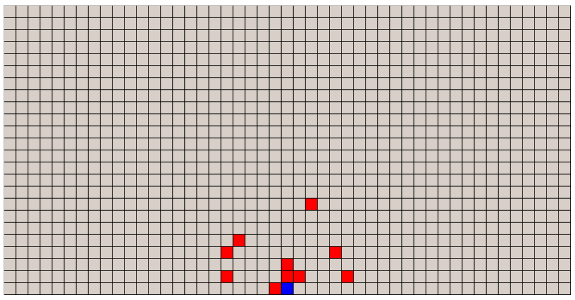
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	70	12	1.06E-01
K-WLSTM	256	22	1.02E-01
Fixed-length contiguous	14	14	1.18E-01
Tree model	31	31	1.10E-01

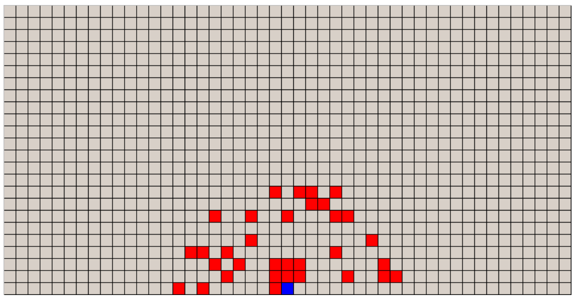
(e) Actual window size, weight and code length of the best templates

Figure A.16: Image "Halftone3", 509 rows x 509 columns


(a) Image


(b) Zoomed portion

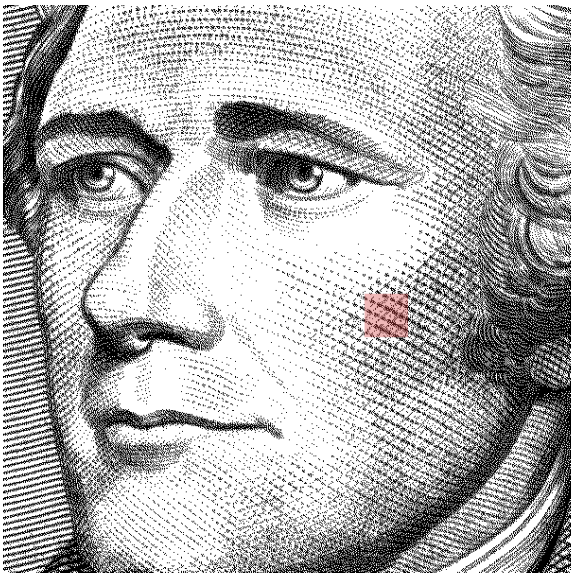

(c) Best K-SCM template

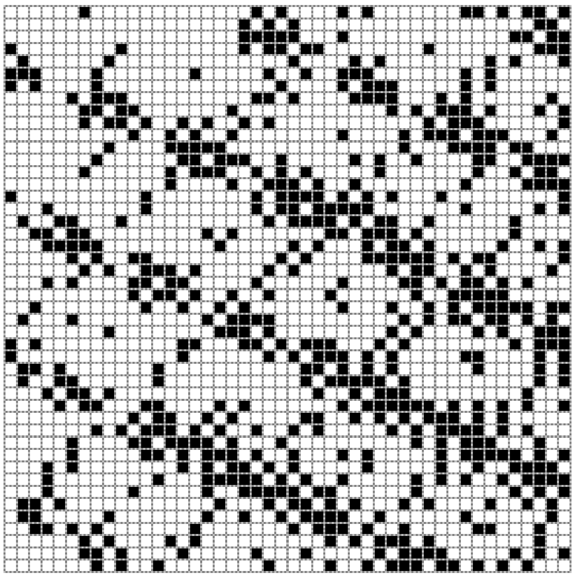

(d) Best K-WLSTM template

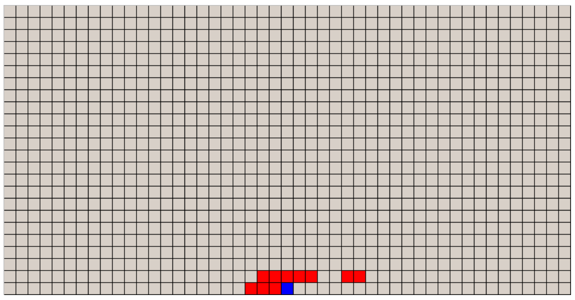
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	88	10	3.28E-01
K-WLSTM	128	33	3.20E-01
Fixed-length contiguous	10	10	3.52E-01
Tree model	61	61	3.34E-01

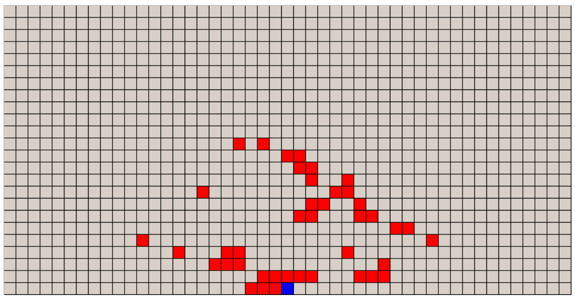
(e) Actual window size, weight and code length of the best templates

Figure A.17: Image “hamilton_bw”, 600 rows x 600 columns


(a) Image


(b) Zoomed portion

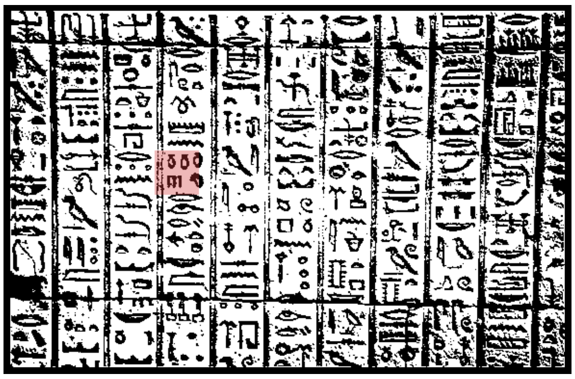

(c) Best K-SCM template

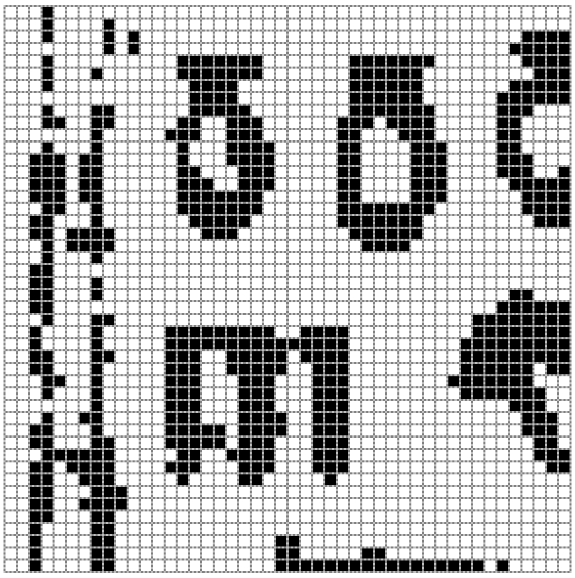

(d) Best K-WLSTM template

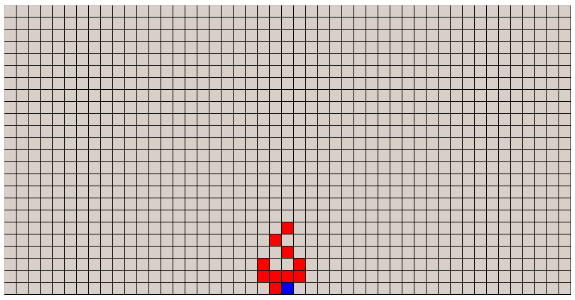
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	58	10	6.36E-01
K-WLSTM	251	41	6.28E-01
Fixed-length contiguous	10	10	6.41E-01
Tree model	52	52	6.36E-01

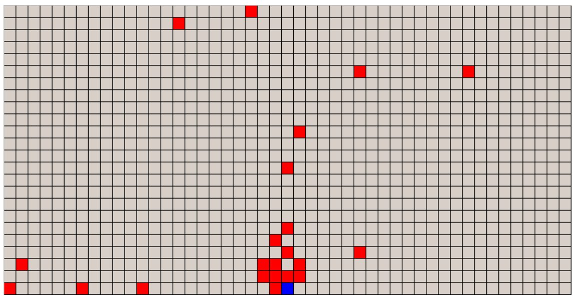
(e) Actual window size, weight and code length of the best templates

Figure A.18: Image "hamilton_ed", 600 rows x 600 columns


(a) Image


(b) Zoomed portion

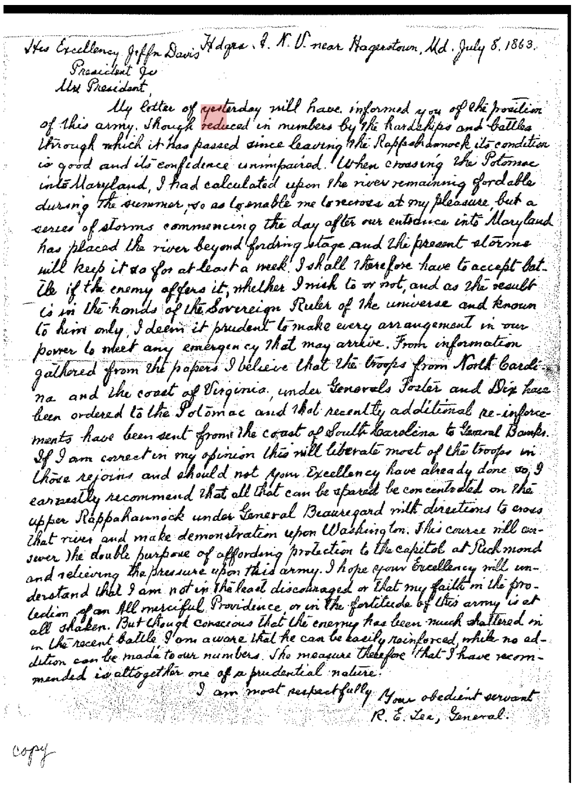

(c) Best K-SCM template


(d) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-SCM	40	10	3.50E-01
K-WLSTM	887	22	3.40E-01
Fixed-length contiguous	11	11	3.56E-01
Tree model	57	57	3.48E-01

(e) Actual window size, weight and code length of the best templates

Figure A.19: Image "hieroglyph", 383 rows x 590 columns



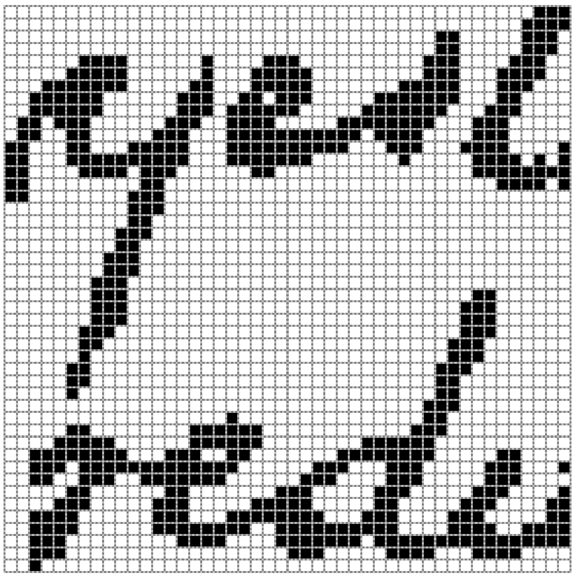
His Excellency Jeffn Davis Hdqrs. A. N. V. near Hagerstown, Md. July 8, 1863.
President &c
Mr President,

My letter of yesterday will have informed you of the position of this army. Though reduced in numbers by the hardships and battles through which it has passed since leaving the Rappahannock its condition is good and its confidence unimpaired. When crossing the Potomac into Maryland, I had calculated upon the river remaining fordable during the summer, so as to enable me to recross at my pleasure, but a series of storms commencing the day after our entrance into Maryland has placed the river beyond fording stage and the present storms will keep it so for at least a week. I shall therefore have to accept battle if the enemy offers it, whether I wish to or not, and as the result is in the hands of the Sovereign Ruler of the universe and known to him only, I deem it prudent to make every arrangement in our power to meet any emergency that may arrive. From information gathered from the papers I believe that the troops from North Carolina and the coast of Virginia, under Generals Foster and Dix have been ordered to the Potomac and that recently additional re-inforcements have been sent from the coast of South Carolina to General Banks. If I am correct in my opinion this will liberate most of the troops in those regions and should not Your Excellency have already done so, I earnestly recommend that all that can be spared be concentrated on the upper Rappahannock under General Beauregard with directions to cross that river and make demonstration upon Washington. This course will answer the double purpose of affording protection to the capitol at Richmond and relieving the pressure upon this army. I hope your Excellency will understand that I am not in the least discouraged or that my faith in the protection of an All merciful Providence or in the fortitude of this army is at all shaken. But though conscious that the enemy has been much shattered in the recent battle I am aware that he can be easily reinforced, while no addition can be made to our numbers. The measure therefore that I have recommended is altogether one of a prudential nature.

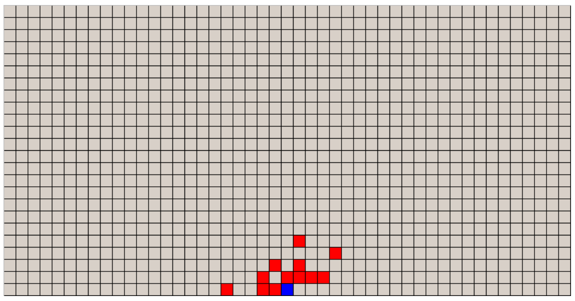
I am most respectfully Your obedient servant
R. E. Lee, General.

copy

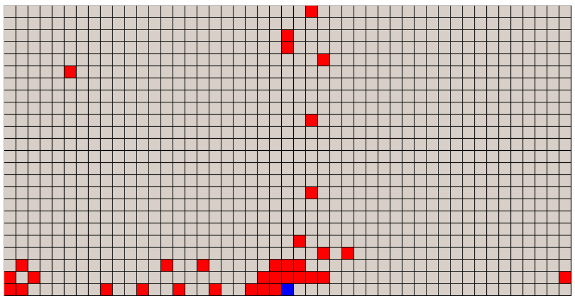
(a) Image



(b) Zoomed portion



(c) Best K-SCM template



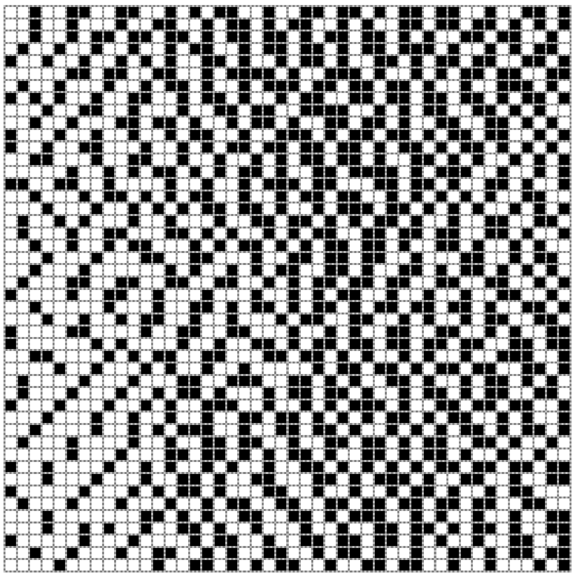
(d) Best K-WLSTM template

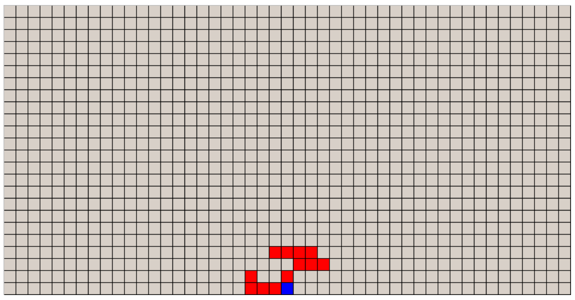
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	37	12	1.83E-01
K-WLSTM	961	34	1.77E-01
Fixed-length contiguous	13	13	1.86E-01
Tree model	55	55	1.80E-01

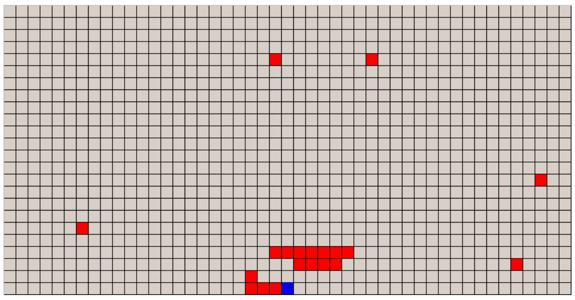
(e) Actual window size, weight and code length of the best templates

Figure A.20: Image “leeleter”, 1380 rows x 1007 columns

(a) Image


(b) Zoomed portion

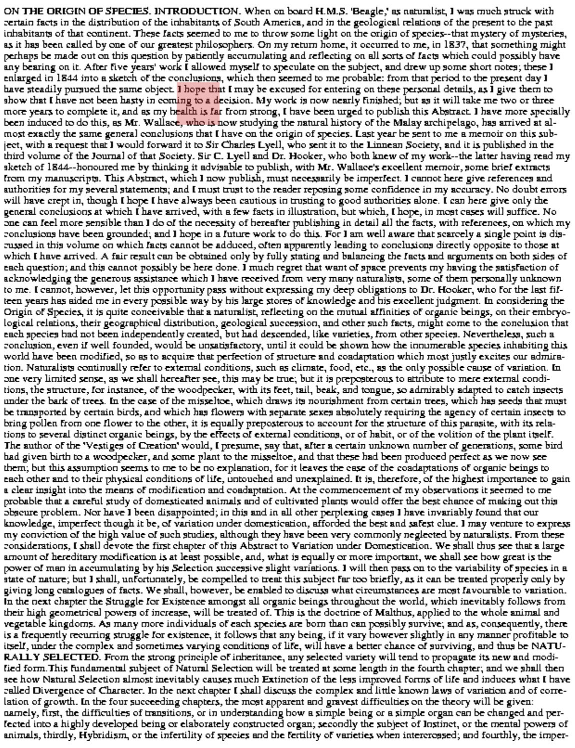

(c) Best K-SCM template


(d) Best K-WLSTM template

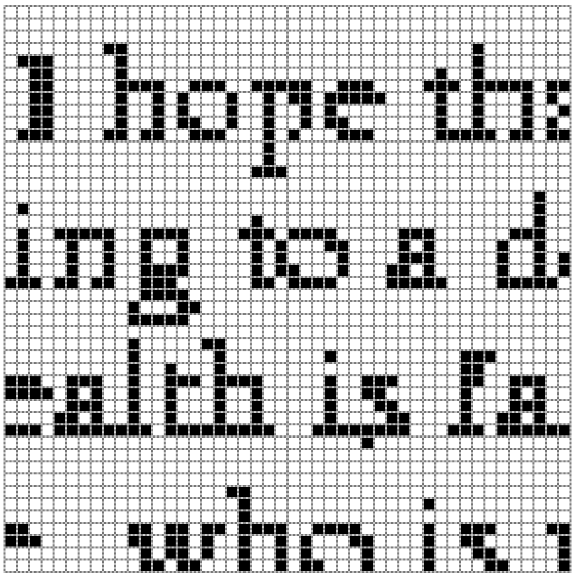
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	30	20	6.45E-01
K-WLSTM	822	20	6.48E-01
Fixed-length contiguous	15	15	7.12E-01
Tree model	25	25	7.20E-01

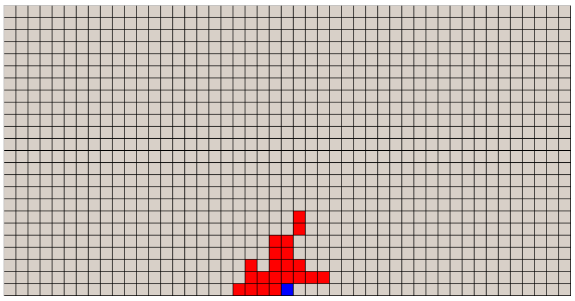
(e) Actual window size, weight and code length of the best templates

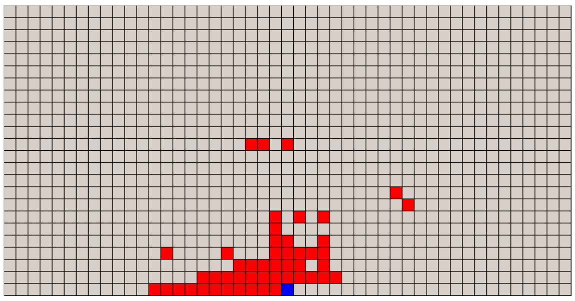
Figure A.21: Image "lena_j", 512 rows x 512 columns


ON THE ORIGIN OF SPECIES. INTRODUCTION. When on board H.M.S. 'Beagle,' as naturalist, I was much struck with certain facts in the distribution of the inhabitants of South America, and in the geological relations of the present to the past inhabitants of that continent. These facts seemed to me to throw some light on the origin of species--that mystery of mysteries, as it has been called by one of our greatest philosophers. On my return home, it occurred to me, in 1837, that something might perhaps be made out on this question by patiently accumulating and reflecting on all sorts of facts which could possibly have any bearing on it. After five years' work I allowed myself to speculate on the subject, and drew up some short notes; these I enlarged in 1844 into a sketch of the conclusions, which then seemed to me probable: from that period to the present day I have steadily pursued the same object. I hope that I may be excused for entering on these personal details, as I give them to show that I have not been hasty in coming to a decision. My work is now nearly finished; but as it will take me two or three more years to complete it, and as my health is far from strong, I have been urged to publish this Abstract. I have more specially been induced to do this, as Mr. Wallace, who is now studying the natural history of the Malay archipelago, has arrived at almost exactly the same general conclusions that I have on the origin of species. Last year he sent to me a memoir on this subject, with a request that I would forward it to Sir Charles Lyell, who sent it to the Linnean Society, and it is published in the third volume of the Journal of that Society. Sir C. Lyell and Dr. Hooker, who both knew of my work--the latter having read my sketch of 1844--honoured me by thinking it advisable to publish, with Mr. Wallace's excellent memoir, some brief extracts from my manuscripts. This Abstract, which I now publish, must necessarily be imperfect. I cannot here give references and authorities for my several statements; and I must trust to the reader reposing some confidence in my accuracy. No doubt errors will have crept in, though I hope I have always been cautious in trusting to good authorities alone. I can here give only the general conclusions at which I have arrived, with a few facts in illustration, but which, I hope, in most cases will suffice. No one can feel more sensible than I do of the necessity of hereafter publishing in detail all the facts, with references, on which my conclusions have been grounded; and I hope in a future work to do this. For I am well aware that scarcely a single point is discussed in this volume on which facts cannot be adduced, often apparently leading to conclusions directly opposite to those at which I have arrived. A fair result can be obtained only by fully stating and balancing the facts and arguments on both sides of each question; and this cannot possibly be here done. I much regret that want of space prevents my having the satisfaction of acknowledging the generous assistance which I have received from very many naturalists, some of them personally unknown to me. I cannot, however, let this opportunity pass without expressing my deep obligations to Dr. Hooker, who for the last fifteen years has aided me in every possible way by his large stores of knowledge and his excellent judgment. In considering the Origin of Species, it is quite conceivable that a naturalist, reflecting on the mutual affinities of organic beings, on their embryological relations, their geographical distribution, geological succession, and other such facts, might come to the conclusion that each species had not been independently created, but had descended, like varieties, from other species. Nevertheless, such a conclusion, even if well founded, would be unsatisfactory, until it could be shown how the innumerable species inhabiting this world have been modified, so as to acquire that perfection of structure and coadaptation which most justly excites our admiration. Naturalists continually refer to external conditions, such as climate, food, etc., as the only possible cause of variation. In one very limited sense, as we shall hereafter see, this may be true; but it is preposterous to attribute to mere external conditions, the structure, for instance, of the woodpecker, with its feet, tail, beak, and tongue, so admirably adapted to catch insects under the bark of trees. In the case of the misseltoe, which draws its nourishment from certain trees, which has seeds that must be transported by certain birds, and which has flowers with separate sexes absolutely requiring the agency of certain insects to bring pollen from one flower to the other, it is equally preposterous to account for the structure of this parasite, with its relations to several distinct organic beings, by the effects of external conditions, or of habit, or of the volition of the plant itself. The author of the 'Vestiges of Creation' would, I presume, say that, after a certain unknown number of generations, some bird had given birth to a woodpecker, and some plant to the misseltoe, and that these had been produced perfect as we now see them; but this assumption seems to me to be no explanation, for it leaves the case of the coadaptations of organic beings to each other and to their physical conditions of life, untouched and unexplained. It is, therefore, of the highest importance to gain a clear insight into the means of modification and coadaptation. At the commencement of my observations it seemed to me probable that a careful study of domesticated animals and of cultivated plants would offer the best chance of making out this obscure problem. Nor have I been disappointed; in this and in all other perplexing cases I have invariably found that our knowledge, imperfect though it be, of variation under domestication, afforded the best and safest clue. I may venture to express my conviction of the high value of such studies, although they have been very commonly neglected by naturalists. From these considerations, I shall devote the first chapter of this Abstract to Variation under Domestication. We shall thus see that a large amount of hereditary modification is at least possible, and, what is equally or more important, we shall see how great is the power of man in accumulating by his Selection successive slight variations. I will then pass on to the variability of species in a state of nature; but I shall, unfortunately, be compelled to treat this subject far too briefly, as it can be treated properly only by giving long catalogues of facts. We shall, however, be enabled to discuss what circumstances are most favourable to variation. In the next chapter the Struggle for Existence amongst all organic beings throughout the world, which inevitably follows from their high geometrical powers of increase, will be treated of. This is the doctrine of Malthus, applied to the whole animal and vegetable kingdoms. As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it vary however slightly in any manner profitable to itself, under the complex and sometimes varying conditions of life, will have a better chance of surviving, and thus be NATURALLY SELECTED. From the strong principle of inheritance, any selected variety will tend to propagate its new and modified form.This fundamental subject of Natural Selection will be treated at some length in the fourth chapter; and we shall then see how Natural Selection almost inevitably causes much Extinction of the less improved forms of life and induces what I have called Divergence of Character. In the next chapter I shall discuss the complex and little known laws of variation and of correlation of growth. In the four succeeding chapters, the most apparent and gravest difficulties on the theory will be given: namely, first, the difficulties of transitions, or in understanding how a simple being or a simple organ can be changed and perfected into a highly developed being or elaborately constructed organ; secondly the subject of Instinct, or the mental powers of animals, thirdly, Hybridism, or the infertility of species and the fertility of varieties when intercrossed; and fourthly, the imper-

(a) Image


(b) Zoomed portion

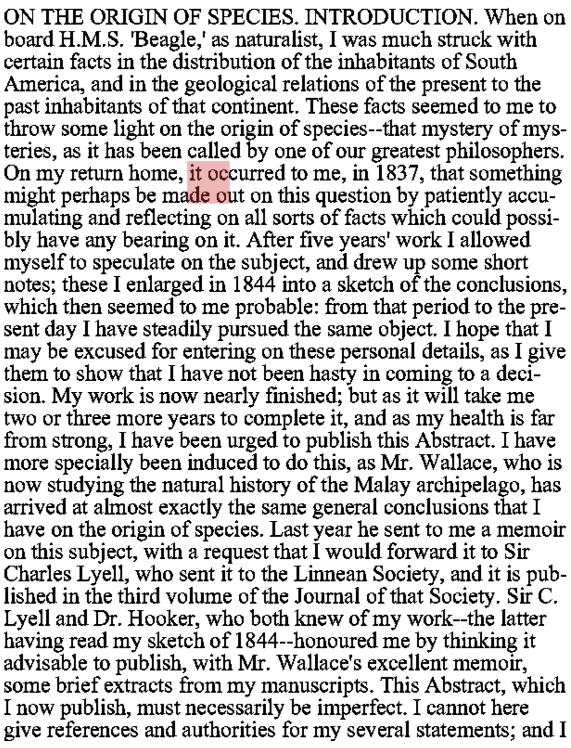

(c) Best K-SCM template


(d) Best K-WLSTM template

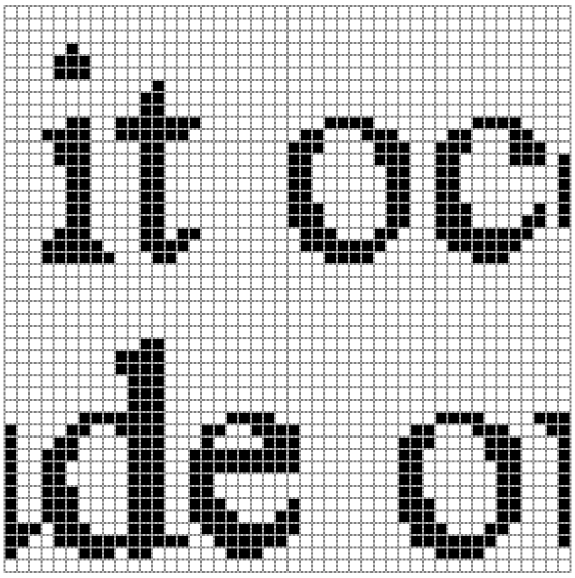
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	60	21	2.27E-01
K-WLSTM	243	49	1.90E-01
Fixed-length contiguous	19	19	2.42E-01
Tree model	64	64	1.97E-01

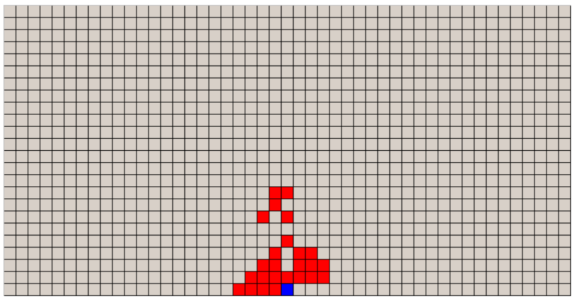
(e) Actual window size, weight and code length of the best templates

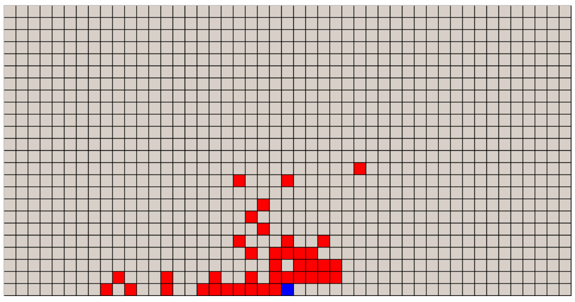
Figure A.22: Image “otoosfont12”, 792 rows x 612 columns


ON THE ORIGIN OF SPECIES. INTRODUCTION. When on
board H.M.S. 'Beagle,' as naturalist, I was much struck with
certain facts in the distribution of the inhabitants of South
America, and in the geological relations of the present to the
past inhabitants of that continent. These facts seemed to me to
throw some light on the origin of species--that mystery of mys-
teries, as it has been called by one of our greatest philosophers.
On my return home, it occurred to me, in 1837, that something
might perhaps be made out on this question by patiently accu-
mulating and reflecting on all sorts of facts which could possi-
bly have any bearing on it. After five years' work I allowed
myself to speculate on the subject, and drew up some short
notes; these I enlarged in 1844 into a sketch of the conclusions,
which then seemed to me probable: from that period to the pre-
sent day I have steadily pursued the same object. I hope that I
may be excused for entering on these personal details, as I give
them to show that I have not been hasty in coming to a deci-
sion. My work is now nearly finished; but as it will take me
two or three more years to complete it, and as my health is far
from strong, I have been urged to publish this Abstract. I have
more specially been induced to do this, as Mr. Wallace, who is
now studying the natural history of the Malay archipelago, has
arrived at almost exactly the same general conclusions that I
have on the origin of species. Last year he sent to me a memoir
on this subject, with a request that I would forward it to Sir
Charles Lyell, who sent it to the Linnean Society, and it is pub-
lished in the third volume of the Journal of that Society. Sir C.
Lyell and Dr. Hooker, who both knew of my work--the latter
having read my sketch of 1844--honoured me by thinking it
advisable to publish, with Mr. Wallace's excellent memoir,
some brief extracts from my manuscripts. This Abstract, which
I now publish, must necessarily be imperfect. I cannot here
give references and authorities for my several statements; and I

(a) Image


(b) Zoomed portion


(c) Best K-SCM template

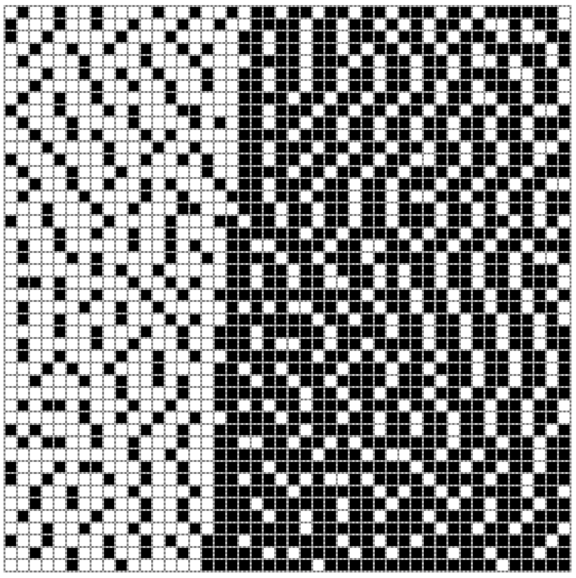

(d) Best K-WLSTM template

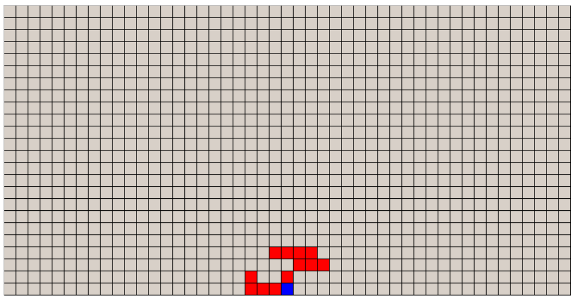
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	105	25	1.61E-01
K-WLSTM	349	39	1.23E-01
Fixed-length contiguous	27	27	1.69E-01
Tree model	64	64	1.35E-01

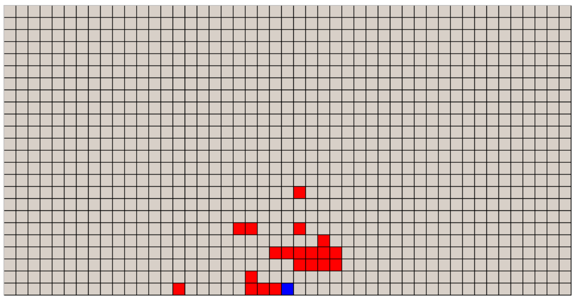
(e) Actual window size, weight and code length of the best templates

Figure A.23: Image “otoosfont24”, 792 rows x 612 columns

(a) Image


(b) Zoomed portion

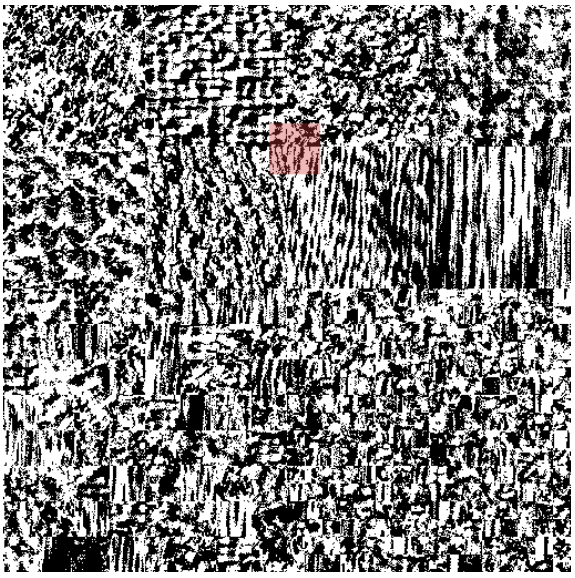

(c) Best K-SCM template

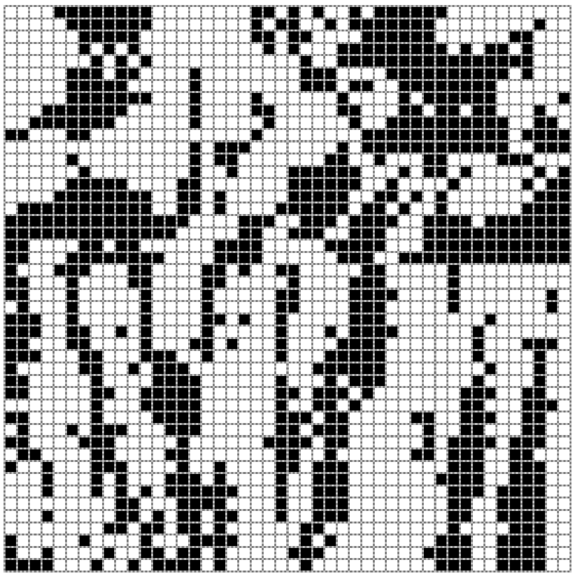

(d) Best K-WLSTM template

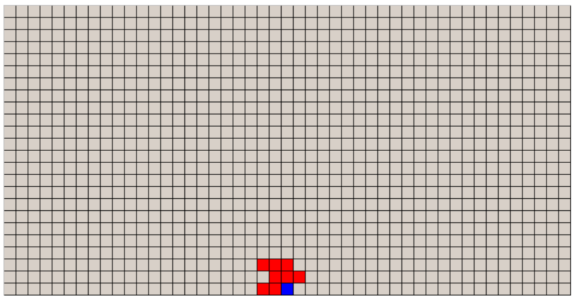
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	22	12	6.18E-01
K-WLSTM	801	21	6.20E-01
Fixed-length contiguous	15	15	6.85E-01
Tree model	24	24	7.00E-01

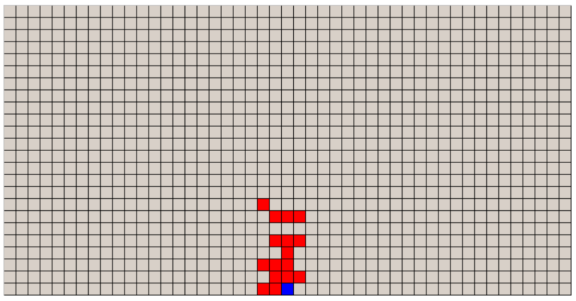
(e) Actual window size, weight and code length of the best templates

Figure A.24: Image “pep_j”, 512 rows x 512 columns


(a) Image


(b) Zoomed portion

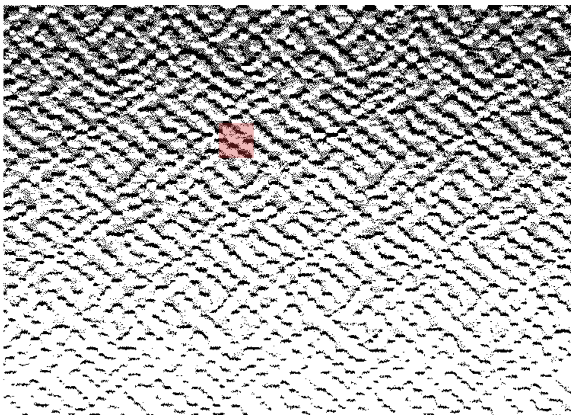

(c) Best K-SCM template

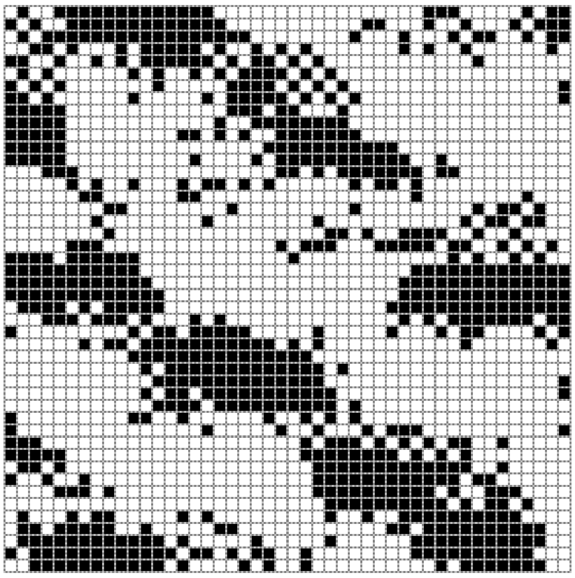

(d) Best K-WLSTM template

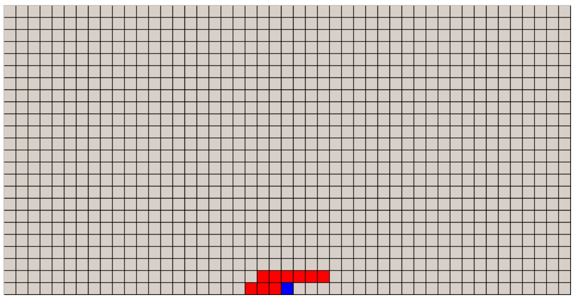
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	19	16	6.44E-01
K-WLSTM	87	16	6.43E-01
Fixed-length contiguous	7	7	6.46E-01
Tree model	13	13	6.44E-01

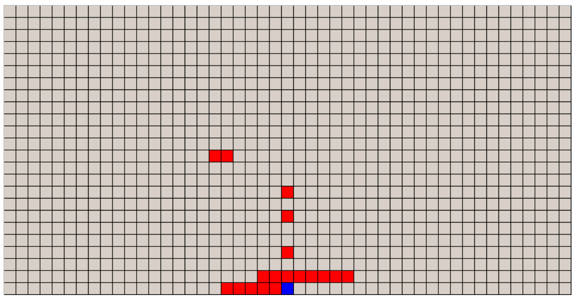
(e) Actual window size, weight and code length of the best templates

Figure A.25: Image "texmos1.p512M", 512 rows x 512 columns


(a) Image


(b) Zoomed portion

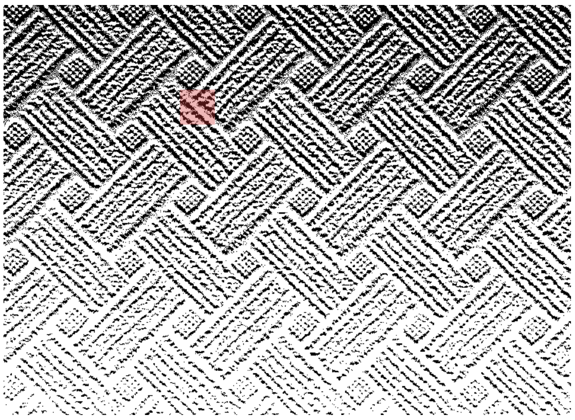

(c) Best K-SCM template

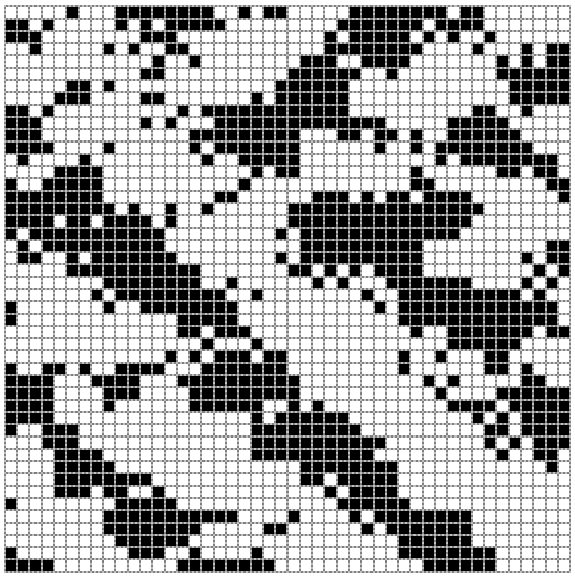

(d) Best K-WLSTM template

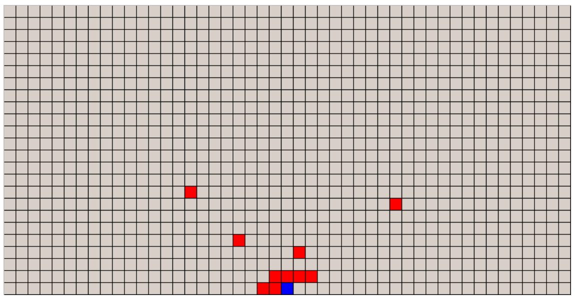
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	24	17	4.88E-01
K-WLSTM	988	27	4.84E-01
Fixed-length contiguous	9	9	4.94E-01
Tree model	42	42	4.88E-01

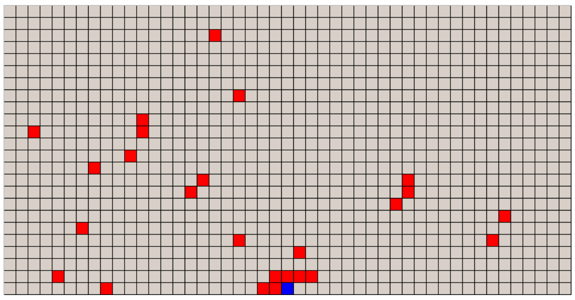
(e) Actual window size, weight and code length of the best templates

Figure A.26: Image "wallpaper003-inca-100dpi-00M", 538 rows x 746 columns


(a) Image


(b) Zoomed portion

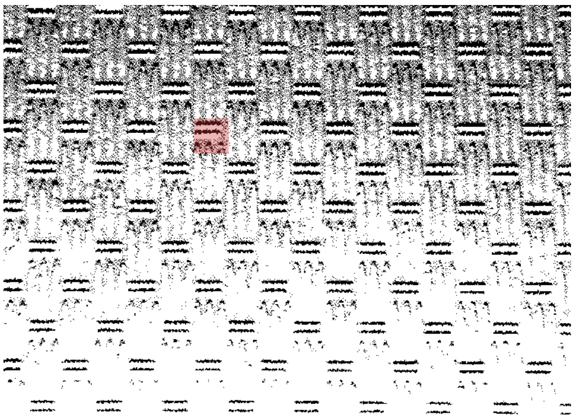

(c) Best K-SCM template

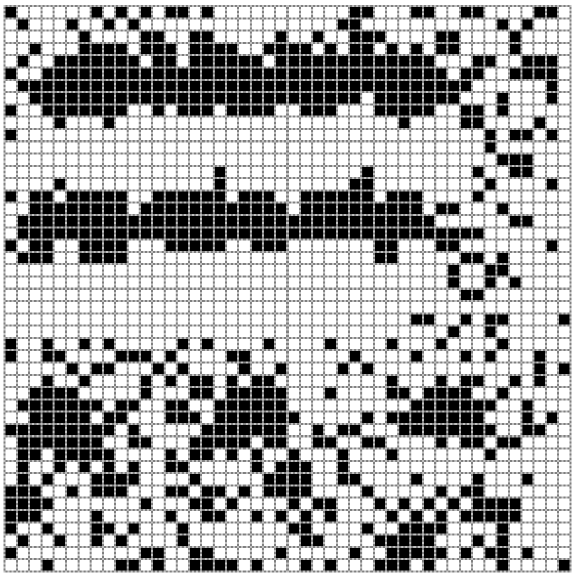

(d) Best K-WLSTM template

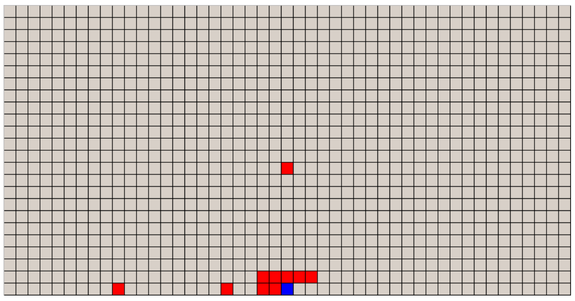
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	206	10	5.16E-01
K-WLSTM	931	25	5.15E-01
Fixed-length contiguous	8	8	5.30E-01
Tree model	19	19	5.28E-01

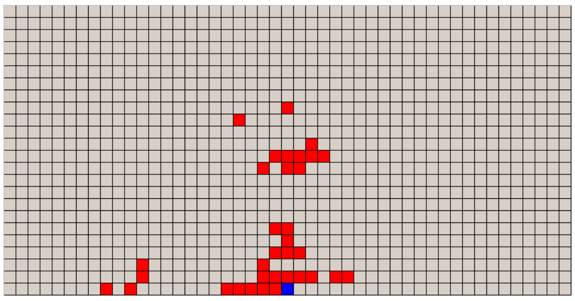
(e) Actual window size, weight and code length of the best templates

Figure A.27: Image "wallpaper004-inca-100dpi-00M", 538 rows x 746 columns


(a) Image


(b) Zoomed portion

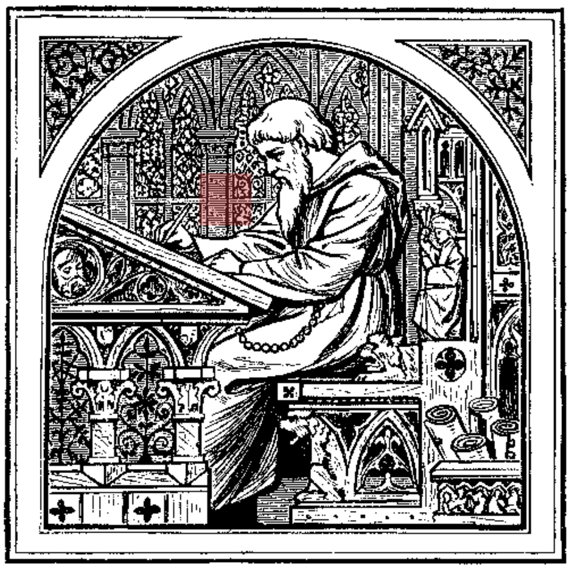

(c) Best K-SCM template

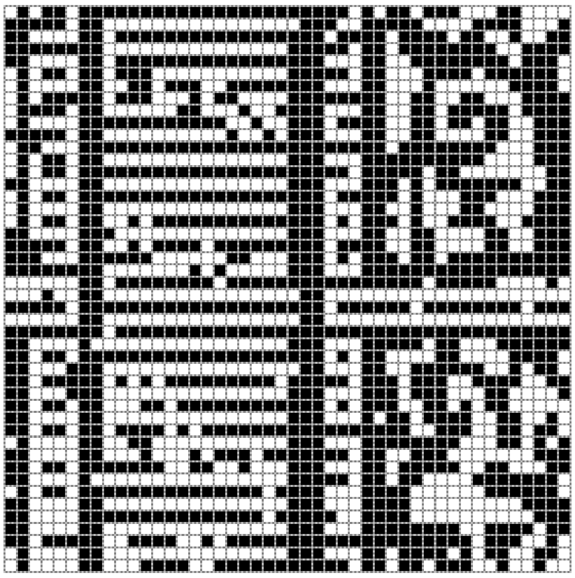

(d) Best K-WLSTM template

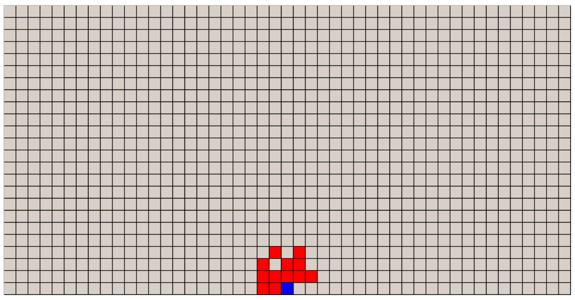
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	305	10	4.88E-01
K-WLSTM	877	33	4.81E-01
Fixed-length contiguous	9	9	4.97E-01
Tree model	44	44	4.85E-01

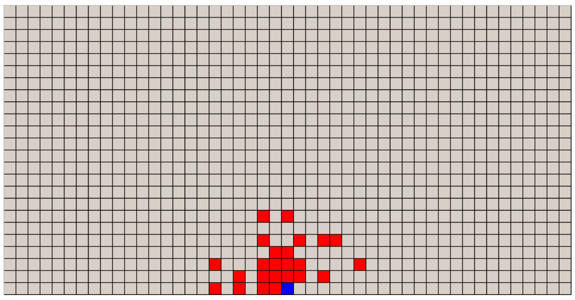
(e) Actual window size, weight and code length of the best templates

Figure A.28: Image "wallpaper010-inca-100dpi-00M", 538 rows x 746 columns


(a) Image


(b) Zoomed portion

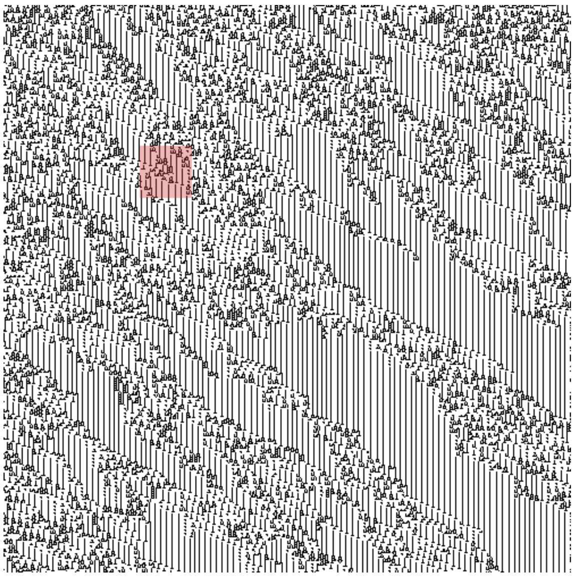

(c) Best K-SCM template

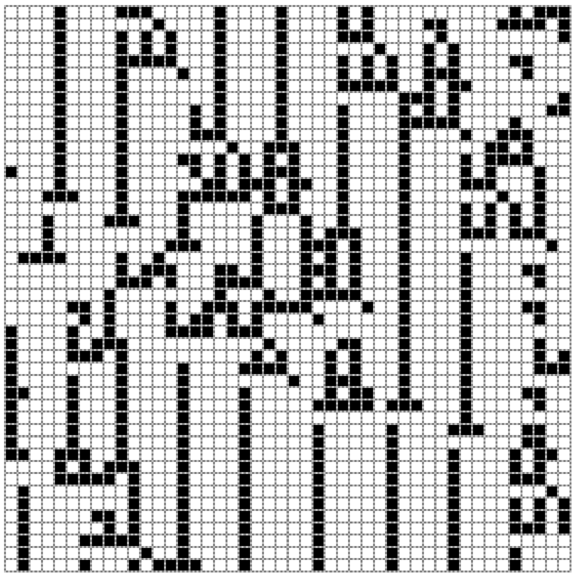

(d) Best K-WLSTM template

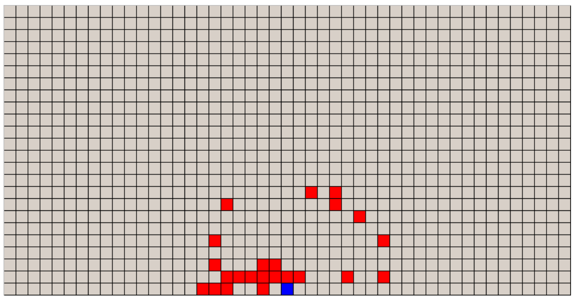
Model	$\tilde{K}$	k	Best normalized code length
K-SCM	18	12	4.47E-01
K-WLSTM	63	24	4.33E-01
Fixed-length contiguous	12	12	4.50E-01
Tree model	27	27	4.37E-01

(e) Actual window size and weight of the best templates

Figure A.29: Image "writing", 499 rows x 500 columns


(a) Image

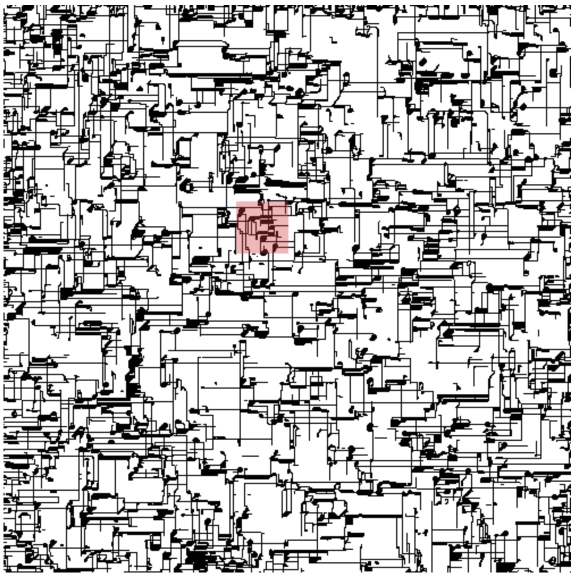

(b) Zoomed portion

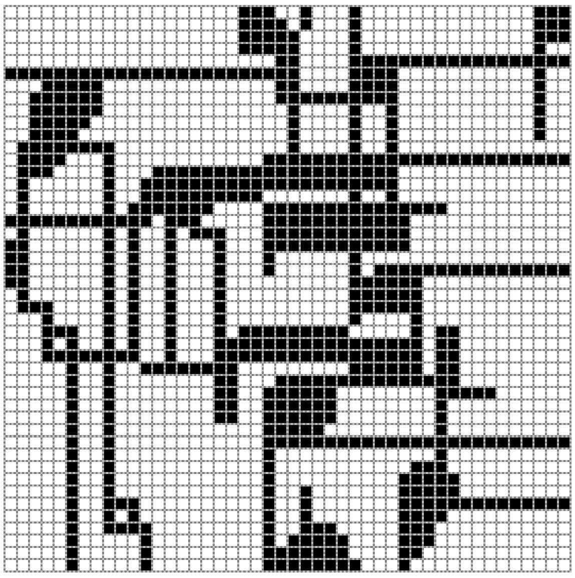

(c) Best K-WLSTM template

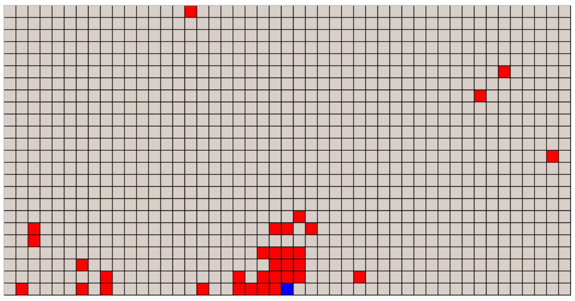
Model	$\tilde{K}$	k	Best normalized code length
K-WLSTM	124	23	7.73E-02
Tree model	42	42	1.37E-01

(d) Actual window size and weight of the best templates

Figure A.30: Image "GEN_A-fixedwidth6and8", 500 rows x 500 columns


(a) Image

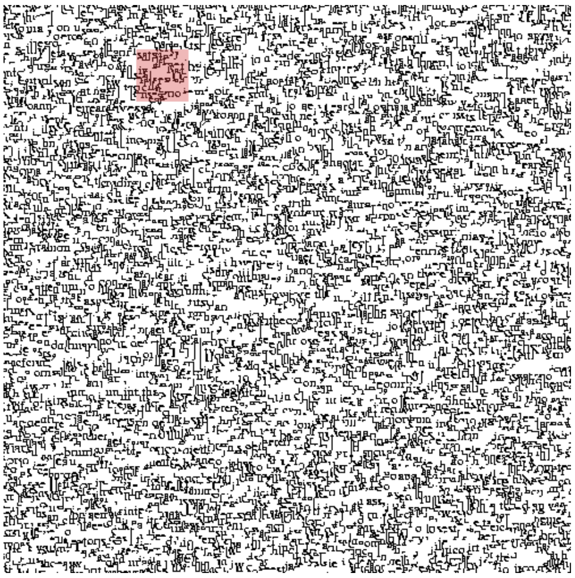

(b) Zoomed portion

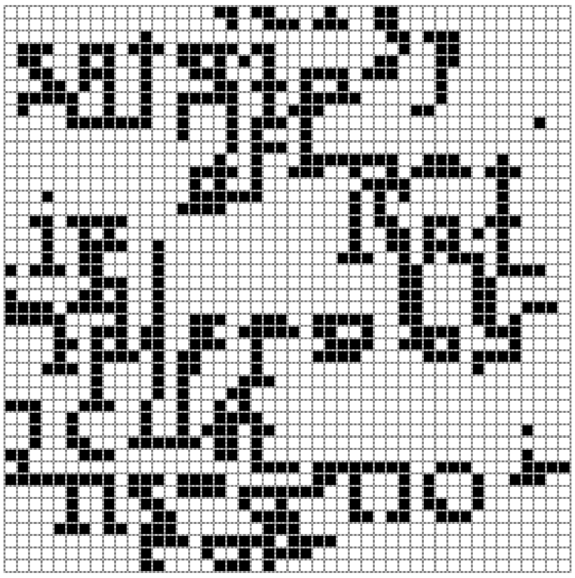

(c) Best K-WLSTM template

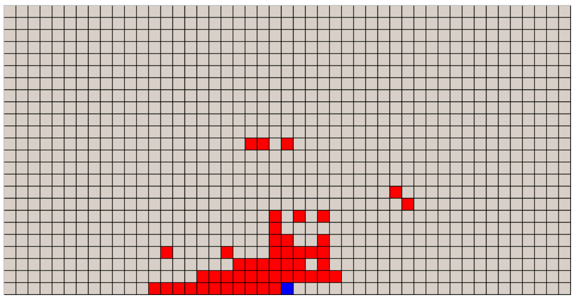
Model	$\tilde{K}$	k	Best normalized code length
K-WLSTM	962	33	1.86E-01
Tree model	28	28	2.04E-01

(d) Actual window size and weight of the best templates

Figure A.31: Image “GEN_cmfugue”, 500 rows x 500 columns


(a) Image


(b) Zoomed portion


(c) Best K-WLSTM template

Model	$\tilde{K}$	k	Best normalized code length
K-WLSTM	243	49	3.00E-01
Tree model	41	41	3.14E-01

(d) Actual window size and weight of the best templates

Figure A.32: Image “GEN_otoosfont12”, 500 rows x 500 columns

Appendix B

2D Context locations

Figure B.1 shows the 2D relative locations and their order (discussed in Section 4.1) for $K = 32, 64, 128, 256, 512, 1024$.

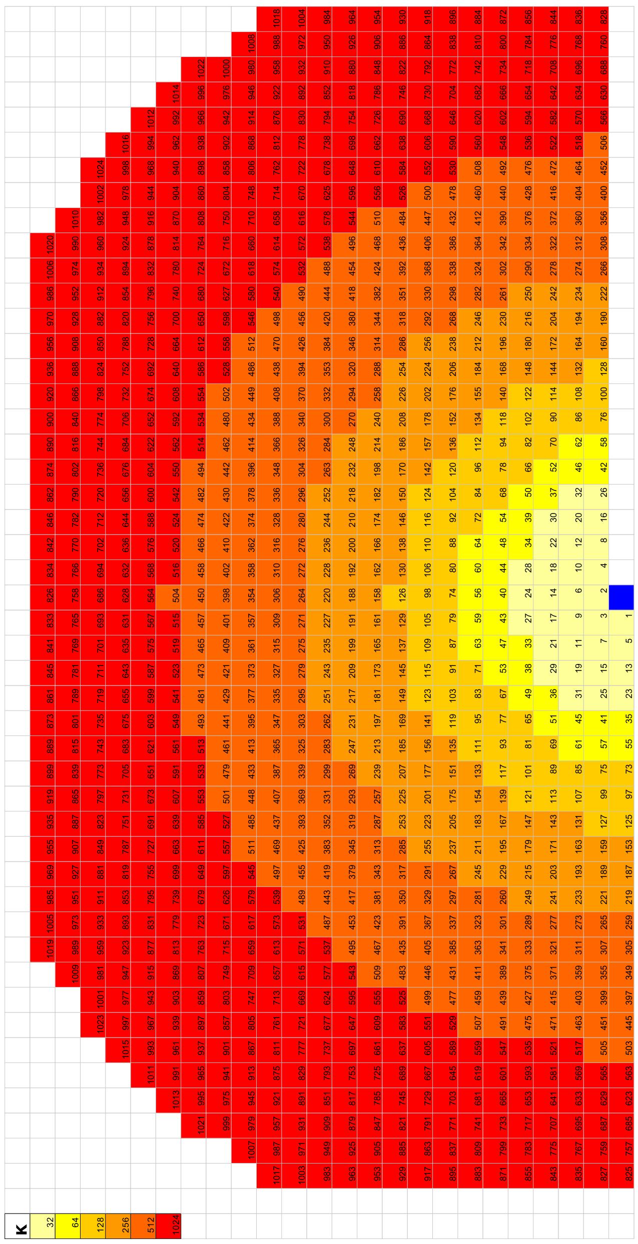


Figure B.1: 2D relative locations and their corresponding order for different K values. The darkest square represents the current sample.

Appendix C

Sparse encoder-decoder

The sparse encoder that was implemented for this thesis receives as input the image to be compressed and a sparse template $\mathcal{T}$ for the model to be used. Then, it produces as output a string of bits by the following steps:

- the dimensions of the image are encoded with a fixed number of bits (64 in our implementation)
- when using K-WLSTMs, the optimum tree, given $\mathcal{T}$, is found by the dynamic programming algorithm described in Subsection 2.2.2 and is described with one bit per node
- using an arithmetic encoder:[1]
  - the representation of $\mathcal{T}$ as a binary string is compressed by a memoryless model whose probabilities are given by the KT estimator
  - the image data is sequentially modeled and compressed using the sparse model implied by $\mathcal{T}$ and the KT estimator, while the border values are obtained through reflection as explained in Footnote 1 of Chapter 4.

The decoder, once it knows the sparse template and, in the case of K-WLSTMs, also the tree structure, using the arithmetic decoder, it sequentially recovers the original image data by estimating the probabilities as the encoder does, by performing the same reflection procedure for the border data and by using the dimensions information to arrange the data.

[1] In our implementation, we used A. Said's arithmetic coder called FastAC available at http://www.cipr.rpi.edu/~said/FastAC.html as of April 2009.

Appendix D

Details on contiguous tree optimization

Table D.1 shows which was the window size K considered for the contiguous tree evaluation of Section 8.5 for each image of the test set, since this evaluation requires a computer memory size $O\left(2^K\right)$. Context locations were considered in the order described in Appendix B.

Image	K
1.1.01M	58
1.1.13M	64
1.5.02M	64
A-fixedwidth6and8	64
albert2D	64
amb	64
Bach_CPE-Sonata_flauto_solo_La_min-fl	64
Bobbys_letter_page_1	64
ccitt4small	58
ccitt7small	59
chinese_text	64
cmfugue1-0	64
flakes006-inca-100dpi-00M	55
HALFTONE	64
Halftone2	64
Halftone3	64
hamilton_bw	64
hamilton_ed	52
hieroglyph	64
leeleter	55
lena_j	54
otoosfont12	64
otoosfont24	64
pep_j	56
texmos1.p512M	58
wallpaper003-inca-100dpi-00M	56
wallpaper004-inca-100dpi-00M	54
wallpaper010-inca-100dpi-00M	58
writing	64
GEN_A-fixedwidth6and8	64
GEN_cmfugue	62
GEN_otoosfont12	64

Table D.1: Window size for contiguous tree optimization