USING ALGORITHMIC INFORMATION THEORY AND STOCHASTIC MODELING TO IMPROVE CLASSIFICATION AND EVOLUTIONARY COMPUTATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
OF THE UNIVERSIDAD AUTONOMA DE MADRID
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Manuel Cebrián Ramos
June 2007

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Manuel Alfonseca Moreno, Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Alfonso Ortega de la Puente, Co-Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Francisco Saiz López, Reader

Approved for the Department of Computer Science Committee on Doctoral Studies.

Abstract

This thesis presents theoretical and practical contributions in Algorithmic Information Theory and (Algorithmic) Stochastic Modeling. Algorithmic Information Theory is the theory concerned with obtaining an absolute measure of the information contained in an object. Stochastic Modeling is a methodology to improve an algorithm's performance by means of the introduction of random elements in its logic.

One of the most interesting advances of Algorithmic Information Theory is the development of an absolute measure of similarity between objects. This measure can only be estimated, as it is incomputable by definition. The typical estimation relies on the use of data compression algorithms, being this estimation known as the compression distance. The two theoretical contributions of this thesis analyze the quality of this estimation. The first quantifies the estimation robustness when the information contained in the objects is noise-altered, concluding that it is considerably resistant to noise. The second studies the impact of the compression algorithm implementation on the estimation, yielding some practical recipes for making this choice.

We use variants of the compression distance to develop two applications for classification and one for evolutionary computation. The first application addresses the problem of detecting similarities in objects which have been generated by a predecessor common source, independently of whether they use or not the same coding scheme: this includes detecting document translation and reconstructing phylogenetic threes from genetic material. We make use of the already proved usefulness of compression based similarity distances for educational plagiarism detection to develop our second application: AC, an integrated source code plagiarism detection environment. The third application makes use of this distance as a fitness function, which is used by

evolutionary algorithms to automatically generate music in a given pre-defined style.

Another three new applications are derived using Stochastic Modeling, two for evolutionary computation and one for classification. Two of them are intimately related and make use of the presence of Heavy Tail probability distributions in the optimization processes involved in the generation of fractals by an evolutionary algorithm, and in the training process of a multilayer perceptron. This discovery is used to improve the performance of both algorithms by means of restart strategies. The last application presented in this thesis is a successful story of the use of a special randomized heuristic in a simple genetic algorithm to yield a state-of-the-art evolutionary algorithm for solving Constraint Satisfaction Problems.

Resumen

Esta tesis presenta contribuciones teóricas y prácticas de la Teoría de Información Algorítmica y del Modelado Stocástico (Algorítmico). La Teoría de Información Algorítmica es la teoría concerniente a la obtención de una medida absoluta de la cantidad información contenida en un objeto. El Modelado Estocástico es una metodología para la mejora del rendimiento de algoritmos mediante la introducción de elementos aleatorios en su lógica.

Una de las más interesantes aportaciones de la Teoría de Información Algorítmica es el desarrollo de una medida absoluta de similitud entre objetos. Esta medida sólo puede ser estimada, al ser no computable por definición. La estimación típica se basa en el uso de algoritmos de compresión de datos, siendo esta estimación conocida como la distancia de compresión. Las dos aportaciones teóricas de esta tesis analizan la calidad de esta estimación. La primera cuantifica la robustez de la estimación cuando la información contenida en los objetos ha sido alterada por ruido externo, concluyendo que ésta es considerablemente resistente al mismo. La segunda, estudia el impacto de la implementación del algoritmo de compresión sobre la estimación, obteniéndose algunas recetas prácticas para realizar dicha elección.

Usamos variantes de la distancia de compresión para desarrollar dos aplicaciones para clasificación y una para computación evolutiva. La primera aplicación considera el problema de la detección de similitudes entre documentos que han sido generados por una fuente común predecesora, independientemente de si estos usan o no la misma codificación: esto incluye la detección de traducciones de documentos y la reconstrucción de árboles filogenéticos a partir de material genético. Hacemos uso de la ya demostrada utilidad de las distancias de similitud basadas en compresión en

la detección de plagio (en el ámbito educacional) para desarrollar nuestra segunda aplicación: AC, un entorno integrado de detección de plagio en código fuente. La tercera aplicación hace uso de esta distancia como una función de *fitness*, que es usada por algoritmos evolutivos para generar de forma automática música con un estilo predefinido.

Otras tres nuevas aplicaciones derivan del uso de Modelado Estocástico, dos para computación evolutiva y una para clasificación. Dos de ellas están íntimamente relacionadas y hacen uso de la presencia de distribuciones de probabilidad de Cola Pesada en los procesos de optimización involucrados en la generación de fractales mediante un algoritmo evolutivo, y en el proceso de entrenamiento de un perceptrón multicapa. Este descubrimiento se usa para mejorar el rendimiento de ambos algoritmos mediante el uso de estrategias de recomienzo. La última aplicación presentada en esta tesis es una historia exitosa del uso de una heurística aleatoria especial en un algoritmo genético simple, obteniéndose un algoritmo que equivale al estado del arte para la resolución de Problemas de Satisfacción de Restricciones (CSPs).

Acknowledgements

I really want to thank everybody who has made this thesis possible. Especially my principal adviser, Manuel Alfonseca, and my co-adviser, Alfonso Ortega. Also, all the people with whom I have collaborated in research, yielding several publications: they are Iván Dotú, Kostadin Koroutchev, Álvaro del Val, Iván Cantador, Manuel Freire and Emilio de Rosal.

I also have to mention the people who have helped in this work by making their codes available or kindly giving them to us under our demand; they are Peter van Beek, Xinguang Chen, Stephen Roberts, Pekka Paalanen, Marina de la Cruz and Enrique Alfonseca, as well as the people who have helped with advice and/or discussions like Alberto Suárez, Jose Ramón Dorronsoro, Elka Koroutcheva, Gonzalo Martínez, Ignacio García, David Camacho, Francisco de Borja, Manuel García-Herranz and many others I am unable to remember now.

I would also like to thank the people in my department, specially Juana Calle, for every little detail. They have always looked for me as if I were their son.

I must mention the grants which have partially supported the research contained in this thesis: TIC 2000-0539 (MCyT), TIN 2004-07676-G01 (MEC) and TSI 2005-08225-C07-06 (MEC). I am indebted to Pilar Rodriguez as well, who kindly funded the initial stage of my research with his project TIC 2001-0685-C02-01 (CICYT). Last but not least, I thank the CAM for awarding me with the FPI scholarship, without which I wouldn't have been able to complete this work.

Finally, I must thank all the people who are around me every day: my family, friends and girlfriend.

Thank you all!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The research presented in this thesis is based on two ideas developed around the 1960s: Algorithmic Information Theory and (Algorithmic) Stochastic Modelling.

The first, Algorithmic Information Theory, is the theory concerned with obtaining an objective and absolute notion of information contained in an individual (non-stochastic) object. It was independently and simultaneously put forth by Kolmogorov [97, 96], Solomonoff [144] and Chaitin [31] in the sixties.

The second idea, Algorithmic Stochastic Modelling, deals with the design of Randomized Algorithms, which employ a degree of randomness as part of its logic to solve computational problems. The foundations of this idea can be attributed to Leew et al. [51] for with their pioneering work on probabilistic Turing machines in 1955.

Both concepts share similar historical development, which can be divided in three steps. Initially they were considered as purely theoretical results with limited applicability. On the one hand, the Kolmogorov Complexity, Algorithmic Information Theory's key concept, was an incomputable function due to the Turing's halting problem. On the other (Algorithmic Stochastic Modeling), the introduction of stochastic elements in the computer seemed rather arbitrary and lacking of motivation, not to talk about the practical impossibility of obtaining truly random number generators at those times. Nowadays it seems pretty ironical that concepts spawned withing the computation found their main obstacles in the computation itself.

This initial 'theoretical' stage lasted differently for both disciplines. It endured

from 1955 to 1976 for Algorithmic Stochastic Modeling, with Rabin's proposal of his famous probabilistic algorithm for testing primality [132] , the first proof of successful applicability of randomization. Much time did it took for Algorithmic Information Theory to become practical, from 1964 to 1997 when a real methodology for estimating and managing the Kolmogorov Complexity was rigorously presented by Li and Vitányi [105]

We can speak about a third 'exploitative' stage, where applications based on these two concepts have become the state-of-the-art in several fields.

Randomized algorithms have become prominent not only in the field of Number Theory, where they were originally used, but also in other areas of computer science such as Artificial Intelligence and Operations Research. In the recent years, stochastic local search algorithms such as Simulated Annealing, Tabu Search and Evolutionary Algorithms have been found to be very successful for solving NP-hard problems from a broad range of domains (e.g. [95] and [72]). But also a number of systematic search methods, like some modern variants of the Davis-Putnam algorithm for propositional satisfiability (SAT) problems, or backtracking-style algorithms for Constraint Satisfaction Problems (CSPs) and graph coloring problems make use of non-deterministic decisions (like randomized tie-breaking rules) and can thus be characterized as randomized algorithms

The amalgam of applications using Algorithmic Information Theory is not as large as the above mentioned due to its slower development. Despite this, ideas based on it had become the cutting-edge tools for several problems in the last ten years, including this: Clustering, Statistical Model Selection, Prediction, Average Case Analysis of Algorithms, Combinatorics and many others (see [105] for a comprehensive review).

This thesis applies ideas from Algorithmic Information Theory and Algorithmic Stochastic Modeling to improve current approaches to several open problems belonging to two major fields: Classification and Evolutionary Computation. The first is an umbrella term for the general problem of optimal grouping of objects. The second is a powerfull methodology for search and optimization which extracts inspiration from biological evolution.

In the next section we introduce the problems dealt with throughout the whole

research and we establish the boundaries of our work presenting its main contributions.

1.1 Problems addressed and contributions

The problems addressed in this thesis and their the resulting contributions are of both theoretical and practical nature. The theoretical part deals with holes in Algorithmic Information Theory, and has the whole chapter 3 devoted to it. The practical part presents new or improved applications for Classification and Evolutionary Computation derived using Algorithmic Information Theory (chapter 4) and Algorithmic Stochastic Modeling (chapter 5).

This section gives a summary of the research contained in this thesis using the just mentioned division.

1.1.1 Advances in Algorithmic Information Theory

Two advances in Algorithmic Information Theory, specially related to the the Normalized Compression Distance (NCD) are presented.

In the first (section 3.1) we analyze the influence of noise on the NCD, a measure based on the use of compressors to compute the degree of similarity of two files. This influence is approximated by a first order differential equation which gives rise to a complex effect, explaining the fact that the NCD may give values greater than 1, observed by other authors. The model is tested experimentally with good adjustment. Additionally, the influence of noise on the clustering of files of different types is explored, finding that the NCD performs well even in the presence of quite high noise levels.

In the second (section 3.2) we show that the compressors used to compute the normalized compression distance are not idempotent in some realistic scenarios, being strongly biased by the size of the objects and window size, and therefore causing a deviation in the identity property of the distance if we do not take care that the objects to be compressed fit the windows. The relationship underlying the precision of the

distance and the size of the objects is analyzed for several well-known compressors, and specially in depth for three cases, bzip2, gzip and PPMZ which are examples of the three main types of compressors: block-sorting, Lempel-Ziv, and statistic.

1.1.2 New applications of Algorithmic Information Theory

Three new applications of Algorithmic Information Theory are presented, which are specially related to the document similarity framework it provides: Common Source Data Detection, Source Code Plagiarism Detection and Music Generation.

Common Source Data Detection Compression based similarity distances have the main drawback of needing the same coding scheme for the objects to be compared. In some situations, there exists significant similarity with no literal shared information: text translations, different coding schemes, etc. To overcome this problem, we present a similarity measure in section 4.1 which compares the redundancy structure of the data extracted by means of a Lempel-Ziv compression scheme. Each text is represented as a graph in which vertices are text positions and edges represent shared information; two texts are similar with our measure if they have the same referential topology when compressed. In the same section we give empirical evidence and phenomenological explanation that this new measure is a robust indicator, detecting similarity between data coded in different languages. We also regard a textual data without any structure, but with a common source and find that we can detect such data and distinguish this situation from the previous one.

Source Code Plagiarism Detection On the one hand, Plagiarism detection in educational programming assignments is still a problematic issue in terms of resource waste, ethical controversy, legal risks, and technical complexity. On the other, the Normalized Compression Distance has proven to be a powerful source code similarity measure for copy-catching. We have enhanced the Normalized Compression Distance detection capability and usability by incorporating it into AC, a modular and open-source plagiarism detection program. The design is portable across platforms and assignment formats and provides easy extraction into the internal assignment

representation. Statistical analysis and several graphical visualizations aid in the interpretation of analysis results.

Music Generation Recent large scale experiments have shown that the Normalized Information Distance is among the best similarity metrics for melody classification. Section 5.2 proposes the use of this distance as a fitness function which may be used by genetic algorithms to automatically generate music in a given pre-defined style. The minimization of this distance of the generated music to a set of musical guides makes it possible to obtain computer-generated music which recalls the style of a certain human author. The recombination operator plays an important role in this problem and thus several variations are tested to fine tune the genetic algorithm for this application. The superiority of the relative pitch envelope over other musical parameters, such as the lengths of the notes, has been confirmed, bringing us to develop a simplified algorithm that nevertheless obtains interesting results.

1.1.3 New applications of Algorithmic Stochastic Modeling

We present three new applications of Algorithmic Stochastic Modelling, a methodology introduced in section 2.2. The first two are intimately related and make use of the presence of heavy tails in both the optimization process involved in the generation of fractals and in the training process of a multilayer perceptron.

Accelerated Generation of Fractals of a Given Dimension In a previous work, Ortega et al. [125] proposed a Grammatical Evolution algorithm to automatically generate Lindenmayer Systems which represent fractal curves with a predetermined fractal dimension. Section 5.1 gives strong statistical evidence that the probability distribution of the execution time of that algorithm exhibits a heavy tail with a hyperbolic probability decay for long executions, which explains the erratic performance of different executions of the algorithm. Three different restart strategies have been incorporated in the algorithm to mitigate the problems associated to heavy tail distributions: the first assumes full knowledge of the execution time probability distribution, the second and third assume no knowledge. These strategies exploit the

fact that the probability of finding a solution in short executions is non-negligible, yielding a severe reduction, both in the expected execution time (up to one order of magnitude) and in its variance, which is reduced from an infinite to a finite value.

Accelerated Training of Multilayer Perceptions The random initialization of weights of a multilayer perceptron makes it possible to model its training process as a Las Vegas algorithm, i.e. a randomized algorithm which stops when some required training error is obtained, and whose execution time is a random variable. This modelling is used in section 5.2 to perform a case study on a well-known pattern recognition benchmark: the CI Thyroid Disease Database. Empirical evidence is presented of the training time probability distribution exhibiting a heavy tail behavior, meaning a big probability mass of long executions. This fact is exploited to reduce the training time cost by applying two simple restart strategies. The first assumes full knowledge of the distribution yielding a 40% cut down in expected time with respect to the training without restarts. The second, assumes null knowledge, yielding a reduction ranging from 9% to 23%.

The third Algorithmic Stochastic Modeling application presented in this thesis is a success story of the use of a special randomized heuristic in a simple genetic algorithm to yield a state-of-the art evolutionary algorithm for solving Constraint Satisfaction Problems (CSPs).

GRASP-Evolution for Constraint Satisfaction Problems There are several evolutionary approaches for solving random binary Constraint Satisfaction Problems. In most of these strategies we find a complex use of information regarding the problem at hand. In section 5.3 we present a hybrid Evolutionary Algorithm that outperforms previous approaches in terms of effectiveness and compares well in terms of efficiency. Our algorithm is conceptual and simple, featuring a GRASP-like (GRASP stands for Greedy Randomized Adaptive Search Procedure) mechanism for genotype-to-phenotype mapping, and without considering any specific knowledge of the problem. Therefore, we provide a simple algorithm that harnesses generality while boosting

performance.

1.2 Publications

We present the main publications that this thesis yielded. We classify them into the chapters to which the research is related. The 'Under submission' sections refers to manuscripts which have been submitted for consideration for publication in a journal. Finally, the 'Others' section indicates papers published during the Ph.D. period which are not included in this thesis.

Advances in Algorithmic Information Theory

M. Cebrián, M. Alfonseca, A. Ortega. The normalized compression distance is resistant to noise. *IEEE Transactions on Information Theory*, 53(5):1895–1900, May 2007.

M. Cebrián, M. Alfonseca, A. Ortega. Common pitfalls using normalized compression distance: what to watch out for in a compressor. *Communications in Information and Systems*, 5(4):367–384, 2005.

New applications of Algorithmic Information Theory

M. Alfonseca, M. Cebrián, A. Ortega. A simple genetic algorithm for music generation by means of algorithmic information theory. To appear in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Singapore, September 25-28, 2007.

M. Cebrián, M. Alfonseca, A. Ortega. Automatic generation of benchmarks for plagiarism detection tools using grammatical evolution. To appear in *Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO)*, London, UK, July 7-11, 2007. arXiv:cs.NE/0703134.

K. Koroutchev, M. Cebrián. Detecting translations of the same text and data with common source. *Journal of Statistical Mechanics: Theory and Experiments*, P10009, October 2006.

K. Koroutchev, M. Cebrián. Detecting the same text in different languages. In *Proceedings of the IEEE Information Theory Workshop (ITW)*, pages 337–341, Chengdu, China, October 22-26, 2006.

M. Alfonseca, M. Cebrián, A. Ortega. A fitness function for computer-generated music using genetic algorithms. *WSEAS Transactions on Information Science and Applications*, 3(3):518–525, March 2006.

M. Alfonseca, M. Cebrián, A. Ortega. Testing genetic algorithm recombination strategies and the normalized compression distance for computer-generated music. In *Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED)*, pages 15–17, Madrid, Spain, February 15–17, 2006.

M. Alfonseca, M. Cebrián, A. Ortega. Evolving computer-generated music by means of the normalized compression distance. *WSEAS Transactions on Information Science and Applications*, 2(9):1367–1372, September 2005.

New applications of Algorithmic Stochastic Modeling

M. Cebrián, I. Dotú. GRASP-Evolution for constraint satisfaction problems. In *Proceedings of the 8th ACM Genetic and Evolutionary Computation Conference (GECCO)*, pages 531–538, Seattle, USA, July 8-12, 2006.

M. Cebrián, I. Dotú. A simple hybrid GRASP-Evolutionary algorithm for CSPs. In *Proceedings of the 2nd International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS)*, pages 2–15, Sitges, Spain, October 1st, 2005.

M. Cebrián, I. Cantador. Estrategias de recomienzo para el entrenamiento de perceptrones multicapa. En *Actas del 1er IEEE CIS Simposio de Inteligencia Computacional (SICO)*, páginas 55–62, Granada, España, 13-16 Septiembre, 2005.

M. Cebrián, A. Ortega, M. Alfonseca. Acceleration of a procedure to generate fractal curves of a given dimension through the probabilistic analysis of execution time. Volume 14 of *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 265–270. ASME Press, New York, USA, November 2004.

Under submission

M. Freire, M. Cebrián, E. del Rosal. AC: An integrated source code plagiarism detection environment. May 2007. arXiv:cs.IT/0703136.

M. Cebrián, M. Alfonseca, A. Ortega. Grammatical evolution with restarts for fast fractal generation. Submitted to *International Journal of General Systems*, April 2007.

M. Cebrián, I. Cantador. Exploiting heavy tails in training times of multilayer perceptrons. A case study with the UCI thyroid disease database Submitted to *IEEE Computational Intelligence Magazine*, April 2007. arXiv:0704.2725.

Others

I. Dotú, A. del Val, M. Cebrián. Redundant modeling for the quasigroup completion problem. Volume 2833 of *Lecture Notes in Computer Science*, pages 288–302, Springer-Verlag, Berlin/Heidelberg, Germany, September 2003.

I. Dotú, A. del Val, M. Cebrián. Channeling constraints and value ordering in the quasigroup completion problem. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1372–1373, Acapulco, Mexico, August 9-15,

2003.

Chapter 2

Basic concepts

In this chapter we introduce the basic concepts which provide the basement for the improvements presented in this thesis: Algorithmic Information Theory, Algorithmic Stochastic Modeling and Evolutionary Computation. We briefly describe the most important features and results of each and then we detail, with more attention, elements which are key to the advances coming in the following chapters.

2.1 Algorithmic Information Theory

Algorithmic Information Theory (AIT) is the theory concerned with obtaining an objective and absolute notion of information contained in an individual object. It was independently (and almost simultaneously) put forth by Andrey Kolmogorov [97, 96], Ray Solomonoff [144] and Gregory Chaitin [31] in the 1960s.

AIT principally studies Kolmogorov complexity and other complexity measures on strings (or other data structures) derived from it. Because most mathematical objects can be described in terms of strings, or as the limit of a sequence of strings, it can be used to study a wide variety of mathematical objects, including integers and real numbers. It is usual in AIT to consider without loss of generality that all strings are binary.

AIT is generally regarded as the deterministic converse of the, more classical, (Stochastic) Information Theory initiated by Shannon in [140], which is concerned

with measuring information quantities of random variables and whose central concept is the (stochastic) entropy. This measure depends on the probability distribution of the variable, while the algorithmic entropy is a measure for deterministic objects.

Here we only outline the basic notation along with some results used in our research. Reference [39] contains an excellent introduction. A very deep treatment with all details and mathematical proofs can be found in the the book by Li and Vitányi [105].

As mentioned earlier, the central concept in AIT is the *Kolmogorov complexity* or *Algorithmic Entropy* of a string x, $K(x)$, defined as the length of the shortest binary program for generating x in some Universal Turing Machine. Although the Turing formalism is the paradigmatic one, it would be equivalent to consider the length of the shortest binary program to generate x in some general purpose programming language such as LISP, C or Java. Formally stated:

Definition 2.1. The Kolmogorov complexity $K_{\mathcal{U}}(x)$ of a string x with respect to some universal machine $\mathcal{U}$ is defined as:

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p), \tag{2.1}$$

where $l(p)$ is the length of p and $\mathcal{U}(z)$ is the output of the machine $\mathcal{U}$ when fed with input z.

Intuitively, $K(x)$ represents the minimum amount of information required to generate x by means of an algorithm. Put it differently, if $\widetilde{x}$ denotes the shortest program which generates x (if there were more than one, the smallest and first in lexicographical order is chosen) then $K(x) = l(\widetilde{x})$.

We use the notation $K(x, y)$ for the length of the shortest program which generates x and y. The next theorem states that, although the function $K(\cdot)$ is defined in terms of a particular universal machine, it is independent on the computation model and has universal sense thanks to the ability of universal machines to simulate any other machine (and then execute any computable procedure).

Theorem 2.1. If $\mathcal{U}$ is a universal Turing machine, then for any other universal

machine $\mathcal{A}$,

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}} \tag{2.2}$$

Constant $c_{\mathcal{A}}$ is the length of the program which machine $\mathcal{U}$ needs to simulate $\mathcal{A}$ and can, in certain cases, be very large. For example, $\mathcal{A}$ can be a modern computer with a high number of built-in functions embedded in the system and, on the other hand, machine $\mathcal{U}$ can be the least powerful microprocessor in the market (but still universal). The program simulating machine $\mathcal{A}$ in $\mathcal{U}$ must contain all implementation details of all $\mathcal{A}$'s functions.

The crucial point is that the length of program needed to simulate $\mathcal{A}$ in $\mathcal{U}$ is independent of x, the string which is going to be generated. For x large enough, $c_{\mathcal{A}}$ becomes negligible, allowing us to use the Kolmogorov complexity to describe its inherent complexity and leave constants out. This is stated in the next theorem. If $\mathcal{A}$ and $\mathcal{U}$ are both universal, then we have

$$|K_{\mathcal{A}}(x) - K_{\mathcal{U}}(x)| < c$$

for all x. Therefore, from now onwards we will assume the use of a fixed machine $\mathcal{U}$ without loss of generality.

The second most important concept in this field is the *Conditional Kolmogorov Complexity* $K(x|y)$ of x conditioned to y, defined as the length of the shortest program generating x when the universal Turing machine is fed with y as an input for the computation.

Finally, we define the Algorithmic Mutual Information $I(x : y)$ between two strings x and y as $I(x : y) = K(x) - K(x|y)$. A very deep and (as we will see in the following subsection) useful result by Gacs [67] proves that, up to an additive constant term,

$$K(x) + K(y|x) = K(y) + K(x|y). \tag{2.3}$$

and therefore the algorithmic mutual information is symmetric.

$$I(x : y) = I(y : x).$$

Once we have presented the basics, we are prompted to talk about computability.

Any function which involves the calculation of the Kolmogorov complexity is incomputable. This is just a consequence of the non-existence of an algorithm for the halting problem: the only way to find the shortest program generating string x in general is to try all short programs and see which of them can do the job. However, at any time some of the short programs may not have halted and there is no effective way to tell whether they will halt or not and what they will print out. Hence, there is no effective way to find the shortest program to print a given string.

It turns out that this is not a drawback for using AIT in practical applications. This is because in spite of the fact that we are not able to find the shortest program, we can still find one which is very near in length to it by using a data compression algorithm: the compressed length $C(x)$ of a program is (for practical purposes) a good approximation of $K(x)$ [34]. Many reasonable compression algorithms like those in the core of GZIP, BZIP, PPMZ can do the job.

In the following subsection we present a similarity measure based on compressors which uses the theoretical framework of AIT.

2.1.1 Normalized Compression Distance

A natural measure of similarity assumes that two objects x and y are similar if the basic blocks of x are in y and vice versa. If this happens we can describe object x by making reference to the blocks belonging to y, thus the description of x will be very simple using the description of y.

This is partially what a compressor does to code the catenated xy sequence: a search for information shared by both sequences in order to cut back the redundancy of the whole sequence. If the result is small, it means that a lot of information contained in x can be used to code y, following the similarity conditions described in the previous paragraph. This was formalized by Rudi Cilibrasi and Paul Vitányi [34] using ideas presented in Section 2.1, giving rise to the concept of *Normalized Compression Distance* (NCD), which is based on the use of compressors to provide a measure of the similarity between the objects. This distance may then be used to

cluster those objects.

This idea is very powerful, because it can be applied in the same way to all kind of objects, such as music, texts or gene sequences. There is no need to use specific features of the objects to cluster. The only thing needed to compute the distance from one object x to another object y, is to measure the ability of x to turn the description of y simple and vice versa.

Cilibrasi and Vitányi have perfected this idea in two ways, by stating the conditions that a compressor must hold to be useful in the computation of the NCD, and by giving a formal expression to the quality of the distance in comparison with an ideal distance proposed by Li and others in [102].

The following definitions describe the conditions under which the NCD is a quasi-universal normalized admissible distance.

Definition 2.2. A compressor C is *normal* if it satisfies, up to an additive $O(\log n)$ term, with n the maximal binary length of an element involved in the (in)equality concerned, the following axioms:

1. *Idempotency:* $C(xx) = C(x)$, and $C(\lambda) = 0$, where λ is the empty string.
2. *Monotonicity:* $C(xy) \geq C(x)$.
3. *Symmetry:* $C(xy) = C(yx)$.
4. *Distributivity:* $C(xy) + C(z) \leq C(xz) + C(yz)$.

Definition 2.3. A distance $d(x, y)$ is a *normalized admissible distance* or *similarity distance* if it takes values in $[0, 1]$ and satisfies the following conditions for all objects x, y, z:

1. *Identity:* $d(x, y) = 0$ if $x = y$.
2. *Symmetry:* $d(x, y) = d(y, x)$.
3. *Triangular inequality:* $d(x, y) \leq d(x, z) + d(z, y)$.
4. *Density constraint* as in [34].

Definition 2.4. A normalized admissible distance f is *quasi-universal* if for every computable normalized admissible distance d and for all sequences x, y it satisfies the following condition:

$$f(x, y) = O(d(x, y)), \tag{2.4}$$

which means that two objects (of any kind) are similar (i.e. they have a small distance) with respect to a specific feature (pitch for music, sequence alignment for DNA, etc) when they are also similar with respect to a quasi-universal distance.

This universal (or quasi-universal) distance is the final goal for universal clustering, because in principle it will be as good as any other distance specialized in measuring some specific feature.

Reference [102] proposes an incomputable distance that fulfills that goal, the normalized information distance (NID):

$$\mathrm{NID}(x, y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}} \tag{2.5}$$

Inspired by this incomputable distance, the following normalized compression distance was designed [34], which would make the role of a quasi-universal distance:

$$\mathrm{NCD}(x, y) = \frac{\max\{C(xy) - C(x), C(yx) - C(y)\}}{\max\{C(x), C(y)\}} \tag{2.6}$$

$C(xy)$ is the compressed size of the catenation of x and y. NCD generates a non-negative number $0 \leq \mathrm{NCD}(x, y) \leq 1$. Distances near 0 indicate similarity between objects, while those near 1 reveal dissimilarity.

As we are assuming simmetry in the compressors, i.e. $C(xy) = C(yx)$, the equation 2.6 may be found in the literature as:

$$\mathrm{NCD}(x, y) = \frac{C(xy) - \min\{C(x), C(y\}}{\max\{C(x), C(y)\}} \tag{2.7}$$

If $x = y$, NCD becomes

$$\mathrm{NCD}(x, x) = \frac{C(xx) - C(x)}{C(x)} \tag{2.8}$$

The next lines summarize the most important results proved on this distance.

Theorem 2.2. If the compressor is normal, the NCD is a normalized admissible distance satisfying the metric inequalities.

Theorem 2.3. Let d be a normalized admissible distance and C be a normal compressor. Then, $\text{NCD}(x, y) \leq \alpha d(x, y) + \epsilon$ where α and ϵ are well-defined constants which depend on the distance between the compression-estimated Kolmogorov complexities and the real complexities [34].

The last theorem states that, if the compressor is chosen properly (i.e. normal and with good compression power), then the NCD may approximate the behavior of a quasi-universal normalized similarity distance.

2.2 Algorithmic Stochastic Modeling

One of the most amazing discoveries in the algorithm design area is that the incorporation of stochastic elements in a computation process can lead to a significant acceleration over purely deterministic methods. This insight has lead to a widespread effort to the stochastic modelling of algorithms, giving birth to the so-called *randomized algorithms*.

A randomized algorithm is an algorithm which employs a degree of randomness as part of its logic. In common practice, this means that the machine implementing the algorithm has access to a pseudo-random number generator. The algorithm typically uses the random bits as an auxiliary input to guide its behavior, in the hope of achieving a good performance in the *average case*. Formally, the algorithm's performance will be a random variable determined by the random bits, with (hopefully) good expected value; this expected value is called the expected runtime. In successful randomized algorithms, the *worst case* is typically so unlikely to occur that it can be ignored.

There are two principal advantages of randomized algorithms. The first is performance. For many problems, randomized algorithms run faster than the best known

deterministic algorithm. Secondly, many randomized algorithms are simpler to describe and implement than deterministic ones of comparable performance.

2.2.1 A taxonomy of randomized algorithms

There are two different types of randomized algorithms *las Vegas* and *Monte Carlo* algorithms.

If the algorithm *always gives the correct solution* and the the only variation from one run to another is its running time, the algorithm is called a las Vegas algorithm. A good example of this is the famous QuickSort algorithm for sorting arrays of k numbers, which chooses a pivotal number randomly. This algorithm always yields a sorted array and its running time can range from $2\log(k)$ steps to k^2 depending on the random choices made.

In contrast, other randomized algorithms can sometimes produce a solution which is incorrect, i.e. with some random choices the output reaches a correct solution and with others not. If it is possible to bound the probability of such an incorrect solution, the algorithm is called a Monte Carlo algorithm. If these algorithms are run repeatedly (on the same input) with independent random choices at each time, the failure probability can be made arbitrarily small, at the expense of running time. We will see an example of this in the following subsection.

For decision problems, for which the answer to an instance is YES or NO, there are two kinds of Monte Carlo algorithm: those with *one-sided error*, and those with *two-sided errors*. It has two-sided error if there is a non-zero probability that it fails when it outpus either YES or NO. On the other hand, it is said to have a one-sided error if the probability that it fails is zero for at least one of the possible outputs.

It is also possible to find algorithms in which both the running time and the correctness of the solution are random variables.

2.2.2 Verifying matrix multiplication

In this subsection we want to illustrate the general functioning and advantages of randomized algorithms with a simple example. We propose a Monte Carlo algorithm

for verifying matrix multiplication.

Suppose we are given three n x n matrices A, B and C whose values can only be 0 or 1. We want to verify whether

$$AB = C.$$

One way to accomplish this is to multiply A and B and compare the result to C. The simple matrix multiplication algorithm takes $\Theta(n^3)$ operations, altough more sophisticated algorithms are known to take $\Theta(n^{2.37})$ operations.

We may use a Monte Carlo algorithm which allows for faster verification, at the expense of returning a wrong answer with a small probability. The algorithm chooses a random vector $\bar{r} = (r_1, r_2, \ldots, r_n) \in \{0,1\}^n$. It then computes $A(B\bar{r})$ and $C\bar{r}$. If $A(B(\bar{r}) \neq C\bar{r}$, then it outputs $AB \neq C$. Otherwise, it returns that $A(B\bar{r}) = C\bar{r}$.

This algorithm requires three matrix-vector multiplications, which can be done in time $\theta(n^2)$. It can be proven that the probability of the algorithm returning $A(B(\bar{r})) = C$ when they are actually not equal is has an upper bound of $\frac{1}{2}$.

To improve on the error probability we can use that fact that the algorithm has a one-sided error and run the algorithm multiple times. If we ever find an $\bar{r}$ such that $AB\bar{r} \neq C\bar{r}$, then the algorithm will correctly return that $AB \neq C$. If we always find $AB\bar{r} = C\bar{r}$, then the algorithm returns that $AB = C$ and there is some probability of a mistake. Choosing r with replacement from $\{0,1\}^n$ for each trial, we obtain that, after k trials, the probability of the error is at most 2^{-k}. Repeated trials increase the running time to $\theta(kn^2)$.

Suppose we attempt this verification 100 times. The running time of the random checking algorithm is still $\theta(n^2)$, which is faster than the known deterministic algorithm for matrix multiplication for sufficiently large n. The probability that an inequality passes the verification test 100 times is 2^{-100}, an astronomically small number. In practice, the computer is much more likely to crash during the execution of the algorithm than to return a wrong answer.

2.2.3 Many successful stories

An excellent example of the success of algorithmic stochastic modeling can be found in the *Primality Testing* problem: finding out whether a given number n is a prime or not. It was formulated in ancient times and has caught the interest of mathematicians again and again for centuries. Only with the advent of cryptographic systems, which use large prime numbers, did it turn out to be of practical importance.

Algorithms were provided which solved the problem very efficiently and satisfactorily for all practical purposes, like the the Miller-Rabin [132], Solovay-Strassen [145] or Adleman-Huang [10] tests, and provably enjoyed a polynomial time bound in the number of digits needed to write down the number n. The only 'drawback' of these algorithms is their stochastic nature, i.e. the computer that carries out the algorithm performs random experiments, and there is a slight chance that the outcome might be wrong. If we force the algorithm to always yield a correct answer, then the running time might not be polynomial.

The scientific community had to wait until 2004 for the first worst-case polynomial, error-free algorithm by Agrawal and Saxena [12]. Although the news of this amazing result spread very fast worldwide, in practical terms, not much has changed. In cryptographic applications, the fast randomized algorithms for primality testing continue to be used, since they are superior in running time and the error can be kept so small that it is irrelevant for practical applications [55]. The new algorithm does not seem to imply that we can factor numbers fast, and no cryprographic system has been broken. This fact reflects the great importance of algorithmic stochastic modelling in real-world applications.

Randomized algorithms are prominent not only in the field of Number Theory, were they were originally used, but also in other areas of computer science such as Artificial Intelligence and Operations Research. In the recent years stochastic local search algorithms such as Simulated Annealing [95], Tabu Search [72], and Evolutionary Algorithms have been found to be very successful for solving NP-hard problems from a broad range of domains. But also a number of systematic search methods, like some modern variants of the Davis-Putnam algorithm for propositional satisfiability

(SAT) problems, or backtracking-style algorithms for Constraint Satisfaction Problems (CSPs) and graph coloring problems make use of non-deterministic decisions (like randomized tie-breaking rules) and can thus be characterized as randomized algorithms [78].

A compendium of fields where randomization has been successful can be found in [120] and [117].

2.3 Evolutionary Computation

Evolutionary Computation (EC) is an umbrella term used to describe computer-based problem solving systems which use computational models of some of the known mechanisms of evolution as key elements in their design and implementation.

Many different kinds of Evolutionary Algorithms (EAs) have been proposed. The major ones are: Genetic Algorithms, Evolutionary Programming ,Evolution Strategies, Classifier Systems , and Genetic Programming. They all share a common conceptual base of simulating the evolution of individual structures via processes of selection, mutation, and reproduction. The processes depend on the perceived performance of the individual structures as defined by an environment.

More precisely, EAs maintain a population of structures, that evolve according to rules of selection and other operators, that are referred to as *search operators* or *genetic operators*, such as recombination and mutation. Each individual in the population receives a measure of its fitness in the environment. Reproduction focuses attention on high fitness individuals, thus *exploiting* the available fitness information. Recombination and mutation perturb those individuals, providing general heuristics for *exploration*. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

We briefly describe the five main paradigms in EC:

- **Evolutionary Programming:** An evolutionary algorithm developed in the mid 1960s by Fogel, Owens and Walsh [64]. Implementations typically use fixed-length character strings to represent their genetic information, together

with a population of individuals which undergo crossover and mutation in order to find interesting regions of the search space.

- **Evolution Strategies:** A type of evolutionary algorithm developed in the early 1960s by Rechenberg and Schwefel in Germany [133]. It employs real-coded parameters, and in its original form, it relied on mutation as the search operator, and a population size of one. Since then it has evolved to share many features with genetic algorithms.

- **Genetic Algorithms:** A type of EA devised by John Holland [83] in the 1970s. It is a stochastic optimization strategy similar to Evolutionary Programming, but instead places emphasis on seeking to emulate specific genetic operators as observed in nature rather than the behavioral linkage between parents and their offspring.

- **Classifier Systems:** First described by John Holland in the 1970s, a (Learning) Classifier System consists of a population of binary rules on which a genetic algorithm alters and selects the best rules. Instead of using a fitness function, rule utility is decided by a reinforcement learning technique.

- **Genetic Programming:** Developed by Koza in the 1990s [99] it can be described as a genetic algorithm applied to the evolution of computer programs. Subsection 2.3.10 gives a brief introducion on Grammatical Evolution, one of the most promising genetic programming approaches at present, which is extensively used throughout this thesis.

Besides this leading crop, there are numerous other different approaches, alongside hybrid local search experiments called Metaheuristics [119].

An excellent introduction and overview of the EC subfields can be found in [58]. If the reader is interested in the history of EC we recommend consulting the EC Fossil Record [63].

2.3.1 EAs general functioning

The Algorithm Template

```
1.  function GenericEvolutionaryAlgorithm()
2.      pop ← InitialPopulation();
3.      Evaluate(pop);
4.      while not Termination()
5.          parents ← SelectParents(pop);
6.          descendants ← Combine(parents);
7.          Mutate(descendants);
8.          Evaluate(pop);
9.          pop ← SelectPopulation(pop,descendants);
```

Figure 2.1: A Generic Evolutionary Algorithm

Figure 2.1 shows the generic Evolutionary Algorithm template. The population 'pop' is initialized in line 2 and evaluated in line 3. Then, a certain number of iterations is repeated until a termination criterion is reached (line 4). During these iterations the individuals are selected (line 5) to be combined (line 6) and their descendants are mutated (line 7) and evaluated (line 8). Afterwards, a new population is generated from the previous one and the descendants (line 9) although sometimes the previous population can be completely forgotten and only new individuals are considered for the next iteration.

In the next sections we are going to detail each one of these steps.

2.3.2 Representation

This is an issue that is prior to the development of the algorithm. Typically, EAs use a string of numbers as a representation, and frequently it is only a binary string. However, we should not forget that choosing the right representation of a problem is key to the algorithm's performance. Thus, it is well worth to devote some time to representation.

The first issue which arises in some EAs is to link the real problem to the problem representation. This mimics biology where a *genotype* encodes the information that

yields a *phenotype* which is the transformation happening in the nature.

Sometimes, this distinction does not appear if the information and the representation are one and the same thing, which happens rarely both in genetics and EC. Nevertheless, as we will latter see, it is important to explicitly make this distinction since the *genotype* is used for individuals interaction, but the *phenotype* is needed to calculate the actual value of the evaluation function, i.e., the fitness of an individual. Every unit of information stored in the *genotype* is typically named *gene*.

The second issue is what kind of structures we need to use to represent our *genotype* and/or *phenotype* (Note that, many times, the phenotype is not actually implemented, and it might be only calculated when the evaluation function needs it). In general, we can distinguish several types of representation:

Binary Representation: this is the simplest representation we can find. It consists on a binary sequence, i.e., a sequence of 1's and 0's. While this technique is very commonly used, it is not always the best suited approach. Its main drawback is based on the genotype-to-phenotype mapping. For example, when the 1's and 0's represent boolean variables, the genotype-to-phenotype mapping is direct: a 1 represents a true variable and a 0 represents a false one. Instead if, for example, we are representing numbers with binary sequences, we can encounter problems derived from the fact that the distances between the numbers and between their representations do not match. Observe that the distance between 3 and 4 is only 1 in the integers, while if we are representing the numbers as a 4-bits sequence, the distance between 0011 and 0100 is not 1 anymore. Here, as in the next types of representation, we have to decide the length of the string.

Integer Representation: to avoid problems like the one previously stated, we can safely represent the individuals as sequences of integers. This is probably a better suited representation for complex problems like the combinatorial ones.

Real or Floating-Point Representation: which consists of a string of real values. This approach is typically better suited for genes that come from a continuous distribution.

There are other more complex representations such as: strings of letters (which is basically equivalent to that of a finite integer representation) and permutation representations (see [58, pages 41–42]).

2.3.3 Evaluation Function

Closely related to the representation is the issue of the evaluation function. This function associates a value to every individual in the population, and corresponds to the quality of that individual. Thus, different representations of the same problem may have different evaluation functions, since this is typically calculated from the values of the genes of each individual and through the genotype-to-phenotype mapping.

The evaluation function is often referred to as *fitness function* in EC.

2.3.4 Initial Population

Once the representation is fixed, the first issue is developing the construction of an initial population. This is typically performed by randomly generating individuals so that the population can cover wider areas of the search space.

Nonetheless, there are other more specialized methods. A very common approach is to generate the individuals in a greedy manner, which means that every individual is constructed in such a way that, at every time the next gene is given the value that optimizes the evaluation function for that individual. Occasionally, the solutions may be somehow seeded in areas where solutions are likely to be found.

2.3.5 Parent Selection

Selection mimics the survival-of-the-fittest phenomenon available in the nature. By this method, certain elements in the population are chosen to pass to the next generation and are usually combined to form the offspring. This selection mechanism tries, in general, to choose parents that are likely to produce a high-quality descendant. Typically, two individuals are chosen to reproduce and yield descendants. Different kinds of selection mechanism are:

Fitness Proportional Selection: consists of giving a certain probability to be chosen for every individual. This probability depends directly on the absolute fitness of the individual. The main drawback of this mechanism is that the best candidates are very likely to take over the whole population very quickly. This method is often called *roulette-wheel selection*.

Ranking Selection: this method is very similar to the previous one. The difference is that, in this case, individuals are ranked according to their fitness, and then probabilities are given based on the ranking rather than on the fitness itself (see [22]).

Tournament Selection: this is maybe the simplest mechanism, and also the least time-consuming. It consists on choosing k individuals completely at random, and then selecting the two individuals with highest fitness function. Obviously, the complexity of this method depends on the value of k.

There are many other methods, mainly variations of the ones described above [58].

2.3.6 Reproduction

This operator is in charge of combining the parents in such a way that a high quality individual (descendant) is obtained. This mechanism is also known as *crossover*. In some EAs, this operator is able to generate more than one descendant (usually two), but we will assume from now on that only one is generated. Thus, different crossover operators are:

One point crossover: this is the most popular method. It consists of choosing a point randomly, and copying the genes of a parent, from the beginning until this point, to the descendant, and the genes of the other parent from that point till the end.

As an example, assume we have two parents of the form:

$$\sigma_1 = \langle\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ \rangle$$
$$\sigma_2 = \langle\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ \rangle$$

and $k = 5$ is the crossover point, the descendant would either be

$$\langle\ 0\ 1\ 0\ 0\ 1\ /\ 0\ 1\ 0\ 0\ \rangle$$

or

$$\langle\ 0\ 0\ 0\ 1\ 1\ /\ 1\ 1\ 1\ 0\ \rangle$$

Note that extending this operator to generate two descendants is trivial.

Multiple point crossover: is based on the previous operator, and its only difference is that instead of 1 point, several k points are chosen randomly. Then, to generate a descendant it would copy the genes of each parent in turns after each crossover point.

Uniform crossover: is slightly different than the previous one. It treats each gene independently and decides from which parent it is going to be inherited (typically with the same probability).

These methods are the most common ones in the literature. Other more complex ones can also be found. It is also very common to implement a smart uniform crossover where instead of proabilities, the decision criterion is based on the fitness of the descendant.

2.3.7 Mutation

This operator is the main source of diversity. It is based in the biological fact that some genes can mutate for different reasons, and thus, the descendant can acquire genes that are from none of its parents. The most common ones are:

- **Random bit modification:** consists on changing the value of some bits with a given probability. The operator changes the value of every bit in the sequence with a certain probability. If the representation is binary, the effect is that of *flipping* a bit, either from 0 to 1 or from 1 to 0.

- **Swap mutation:** simply selects two genes (at random) in the sequence and swaps their values. Imagine the individual:

  ⟨ 0 [1] 0 [0] 1 0 1 0 0 ⟩

  and the swapping genes 1 and 3, the mutated individual would be

  ⟨ 0 [0] 0 [1] 1 0 1 0 0 ⟩

- **Insert mutation:** chooses two genes at random and moves the second one next to the first. Again, if we have the individual

  ⟨ 0 [1] 0 0 [1] 0 1 0 0 ⟩

  and the inserting genes 1 and 4, the mutate individual would be

  ⟨ 0 [1] [1] 0 0 0 1 0 0 ⟩

- **Scramble mutation:** selects a region in the sequence and randomly scrambles its values. For example,

  ⟨ [0 1 0 0] 1 0 1 0 0 ⟩

  and the region from 0 to 3, a possible mutate individual would be

  ⟨ [1 0 0 0] 1 0 1 0 0 ⟩

Note that all these operators can be applied to any kind of representation, even though the illustrations assume a binary representation.

Many other complex and specialized mutation operators can be found in the literature, including the ones where the mutation is not random but biased by the subsequent value of the fitness function of the individual.

2.3.8 Selection of the new generation

As we have previously introduced, this mechanism replaces the last population by a new one. In order to do so, some algorithms completely replace the previous population by the new set of descendants (*offspring*). However, this type of selection, known

as generational replacement, is usually not a very effective technique, and EAs usually implement mechanisms to generate the new population from both the previous population and the offspring. Among these mechanism we can distinguish:

Fitness based: selection focuses on keeping some percentage of the individuals with the highest fitness for the next generation.

Steady state: this is a particular case of fintess based selection, when only a small number of the individuals with the lowest fitness are replaced from generation to generation.

Generations based: selection takes into account the number of generations passed since its creation, and replaces then those individuals which have been in the population for a larger amount of generations.

A technique associated with this operator (independent of the mechanism type) is to *always* maintain the highest quality individual in the population. This technique is usually referred to as *elitism*.

2.3.9 Termination

The termination condition indicates when it is time for the algorithm to stop. At this point, the algorithm will usually return the best individual (according to its fitness function) found in the whole execution. We can distinguish two kinds of termination condition:

- **Objective reached:** when an EA is implemented to reach a certain goal (i.e., a solution of a certain quality), reaching that goal should be the indication for the algorithm to stop.
- **External conditions:** However, the previous case is very rarely achieved, due to the stochastic nature of these algorithms. Therefore, a different criterion must be used. Different conditions include:
  - Fixed number of generations reached.

- Maximum time allowed reached.
- Fitness improvement does not occur for a certain number of generations.
- Manual inspection.
- A combination of the above.

2.3.10 Grammatical evolution

Grammatical evolution [122] (GE) is the latest, most promising Genetic Programming string-based approach. Genotypes are represented by strings of integers (each of which is called a *codon*) and the context-free grammar of the target programming language is used to deterministically map each genotype into a syntactically correct phenotype (a program). In this way, GE avoids one of the main difficulties in Genetic Programming, as the results of applying genetic operators to the individuals in a population are guaranteed to be syntactically correct. The following scheme shows the way in which GE combines traditional genetic algorithms with genotype-to-phenotype mapping.

1. A random initial population of genotypes is generated.
2. Each member of the population is translated into its corresponding phenotype.
3. The genotype population is sorted by their fitness (computed from the phenotypes).
4. If the best individual is a solution, the process ends.
5. The next generation is created: the mating-pool is chosen with a fitness-proportional parent selection strategy; the genetically modified offspring is generated, and the worst individuals in the population are replaced by them.
6. Go to step 2.

This procedure is similar in many respects to biological evolution. There are three different levels. Figure 2.2 shows a graphical scheme of the process in the particular case studied in chapter 5: the automatic generation of fractal curves with a given dimension.

- The *genotype* (nucleic acids), is represented in GE by vectors of integers.
- The *intermediate* level (proteins), is represented in GE by words in a given alphabet, which in our case describe an L system (see below). The translation from the genotype to the intermediate level is performed by means of a fixed grammar (the equivalent of the fixed genetic code).
- The *phenotypic* (organisms), in our case represented by the fractal curves obtained from the intermediate-level words by means of a graphical interpretation.

For a more technical and comprehensive treatment on Evolutionary Computation we recommend the reader to consult [23].

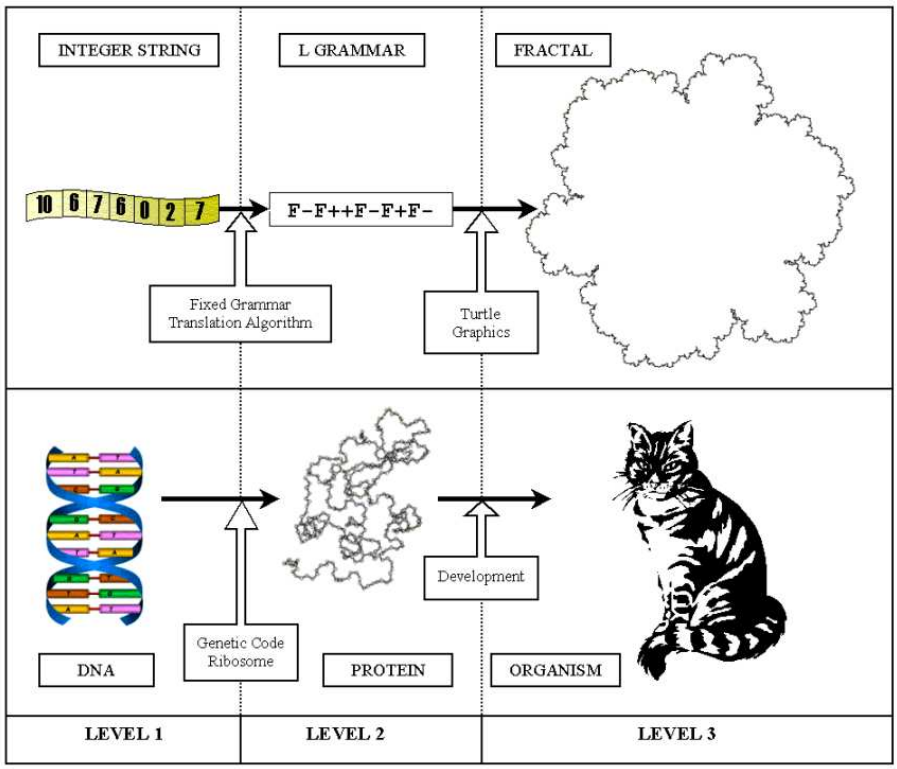

Figure 2.2: Graphical scheme of a GE process

Chapter 3

Advances in Algorithmic Information Theory

In this chapter we present two advances in Algorithmic Information Theory, specially related to the the Normalized Compression Distance (NCD). Both are of theoretical nature.

3.1 The Normalized Compression Distance in the presence of noise

The current massive use of Internet has enormously increased the traffic of files across potentially noisy channels which can change their original contents. NCD is a similarity measure based on the use of compressors, so noise could make NCD get wrong results: a clustering application using NCD as a measure of distance would classify as dissimilar two similar files corrupted by noise.

The experiments described in this subsection have been designed in the following way: all the files contain bytes in a certain range (i.e. genomes can only belong to {A,C,G,T}; texts can contain any ASCII character; music uses MIDI files and images use GIF format, both with their bytes in the range [0,255]).

Noise is applied with certain probability independently to individual bytes, by

integer addition of an uniform random positive (non-zero) value, in such a way that the resulting byte belongs to their appropriate above mentioned ranges. This model of comunication with noise is known in the literature as the *symmetric channel* [39, chapt. 8].

The experimental tests show how the NCD changes when applied to two files, one of which is distorted by an increasing noise ratio (i.e. several percentages of noise are added during the experiments), while the other remains unchanged.

3.1.1 Theoretical Analysis

Let us consider a file of size a which is compressed by a given compressor into another file of size b. If we add noise to the original file and compress it, as the amount of noise increases, the compressor will be able to reduce less and less the file size, until it will be unable to reduce it at all, once the contents of the file become fully random. Therefore the size of the compressed file will start at b, when no noise is added, and will increase steadily to a, which will be reached when the whole initial file has been replaced by random noise.

At a given point in this procedure, if we add Δx noise to the file (i.e. change randomly the values of Δx bytes in the file), the size increase (the compression loss) we may expect will be proportional to the amount of the file which has not yet been replaced by noise. Therefore, the evolution of the compressed size y will be defined by $\Delta y = \gamma(a - y)\Delta x$. When $\Delta x \to 0$, this equation becomes the first-order differential equation $dy/dx = \gamma(a - y)$. The solution of this equation, taking into account the indicated initial conditions, is:

$$y = a - (a - b)e^{-\gamma x} \tag{3.1}$$

where x is the amount of noise added and the value of y (the size of the compressed file) is b for $x = 0$, a for $x \to \infty$.

Consider the definition (2.7, page 16) and assume that we want to study the variation of the distance $\mathrm{NCD}(p, q)$ between a fixed file p, and another file q which is being contaminated by growing amounts of noise. Without loss of generality, we may

assume that $C(p) \leq C(q)$. Therefore, the above distance becomes

$$\mathrm{NCD}(p,q) = \frac{C(pq) - C(p)}{C(q)} \tag{3.2}$$

We have seen that, as the amount of noise x introduced in file q grows, $C(q) = a - (a-b)e^{-\gamma x}$. It is easy to see that $C(pq)$ will evolve in a similar way, although with different constants, because the noise introduced in the second part of the file not only destroys redundancies in that section of the file, but also prevents possible cross-compressions with the first part, which does not receive noise. So, $C(pq) = c - (c-d).e^{-\phi x}$. Finally, $C(p)$ is a constant. Replacing these values, under certain conditions the NCD formula can be approximated by the equation:

$$\mathrm{NCD}(p,q) = \alpha + \beta e^{-\gamma x} - \delta e^{-\phi x}, \tag{3.3}$$

where the values of the constants depend on the actual files p and q compressed; x continues being the amount of noise added. With certain values of the constants, this function reaches values greater than 1 (usually smaller than 1.1). This effect provides a different explanation of the anomaly, signalled in [34], that the value of the NCD may be greater than 1, without any reference to the presence of defects in the compressor implementation.

3.1.2 Experimental results

In the last section we obtained a model (Eq. 3.3) for the NCD in the presence of noise. If we compute the *average distortion* introduced by a noise level l, we come to a similar equation, since $\mathrm{NCD}(p,q)$ is a constant:

$$\begin{aligned} \Delta_l(p,q) &\equiv E[\mathrm{NCD}(p,\ q + n_l \bmod r) - \mathrm{NCD}(p,q)] \\ &\approx \alpha' + \beta' e^{-\gamma' l} - \delta' e^{-\phi' l} \end{aligned} \tag{3.4}$$

where n_l is a random string whose length is equal to the length of q and whose i-th character is non-zero with probability l; r is the size of the file type range (i.e. 4

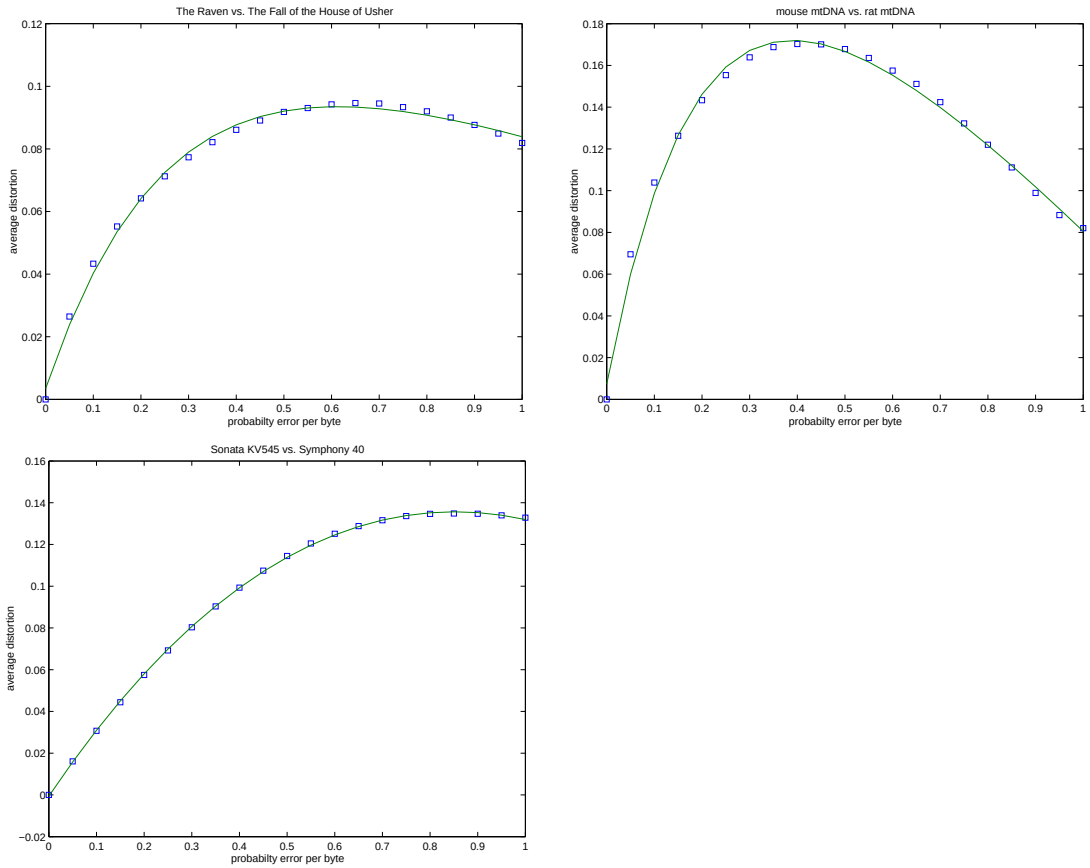

Figure 3.1: Three examples of different types of data which exhibit the typical decay behavior of the average distortion. We compare the texts "The Raven" vs. "The Fall of the House of Usher," (upper left figure) mouse mtDNA vs. rat mtDNA (upper right) and Mozart's "Sonata KV545" vs. "40th Symphony" (lower left).

for DNA, 93 for ASCII and 256 for the rest). The modulo operation is performed to maintain each value in its proper range.

In this subsection we test the goodness of the model (3.4) by adjusting it over experiments with real data: ASCII texts [81], mitochondrial DNA (mtDNA, obtained from [34]), songs in WAV format and face images in GIF format [5]. For each two files p and q we estimate $\Delta_l(x, y)$ averaging over 10 realizations of the random vector n_l, computing distances by means of the CompLearn Toolkit [34], which implements the NCD and NCD-driven clustering; default CompLearn parameters were used in all experiments.

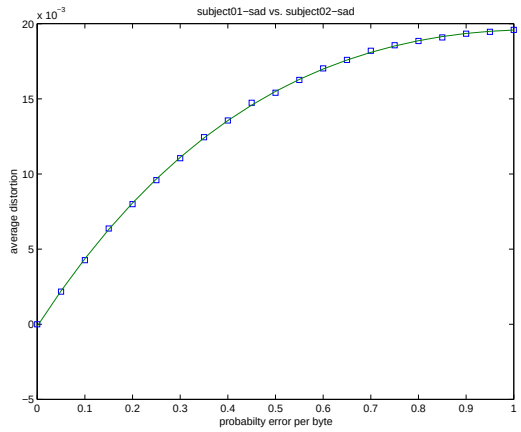

Figure 3.2: An example of the typical non-decay behavior of the average distortion with face images. The two elements in the comparison are subject #1's and #2's face images with a predefined "sad" disposition (subject01-sad vs. subject02-sad).

In all the experiments performed, the model obtained a very accurate fit with a squared 2-norm of the residuals always below 10^{-3}. An interesting result is that, for some data types, the average distortion increases until it reaches a maximum at some $l < 1$ and later decays steadily converging towards a smaller value when the level of noise is increased; texts, mtDNA and songs follow this behavior (Figure 3.1). This phenomenon explains the already mentioned fact that the NCD can sometimes reach values greater than 1 when comparing files that share very little information: the region in which $\Delta_l(x, y)$ has a value greater than $\Delta_1(x, y)$ (distortion with full noise) coincides with the region in with the NCD$(p,\ q + n_l \bmod r)$ is greater than 1.

Other data types like face images (Figure 3.2) increase continuously without any posterior decay. In Table 3.1 we show some results of the experiments performed and their classification according to whether a decay exists or not; 200 experiments were performed for each pair p, q, 10 realizations of n_l for each l, with 20 different values of $l \in \{0.05, 0.10, 0.15, \ldots, 1\}$.

p	q	α'	β'	γ'	δ'	ϕ'	decay
Secret Adversary	The Mysterious Affair at Styles	0.0720	0.9036	2.2057	0.9696	3.0468	yes
Antony and Cleopatra	Hamlet	0.0617	0.9159	2.0486	0.9689	2.6575	yes
An Essay on Criticism	The Fall of the House of Usher	0.0712	0.9116	1.5148	0.9787	2.3714	yes
Hamlet	Secret Adversary	-0.0846	1.0324	1.0488	0.9414	1.4349	yes
The Raven	The Fall of the House of Usher	0.0396	0.9503	1.7972	0.9855	2.1636	yes
mouse	rat	-0.0918	0.8437	1.3972	0.7414	3.0697	yes
finWhale	blueWhale	-0.8439	1.2250	0.1565	0.3683	7.5895	yes
graySeal	Harbor Seal	-0.7520	1.2352	0.1215	0.4693	10.698	yes
human	Blue Whale	-0.2372	0.9206	1.0215	0.6772	2.6497	yes
rat	horse	-0.2220	0.8756	1.0162	0.6464	2.7148	yes
chimpanzee	Gray Seal	-0.2047	0.8422	1.0461	0.6302	2.8282	yes
(Chopin) Prelude 15	(Chopin) Prelude 7	1.4075	-1.1543	0.4476	0.2546	-0.7388	yes
(Chopin) Prelude 15	Begin the Beguine	1.2717	-1.0224	0.3340	0.2508	-0.5981	yes
(Mozart) Sonata KV545	(Mozart) Symphony 40	1.3465	-0.8083	0.6904	0.5388	-0.4068	yes
Begin the Beguine	My heart belongs to daddy	1.3838	-1.0255	0.5251	0.3595	-0.5916	yes
subjec01.centerlight	subject03.centerlight	0.0171	0.6498	6.4166	0.6663	6.2745	no
subject02.happy	subject04.happy	0.0178	0.6501	6.2037	0.6672	6.0454	no
subject01.sad	subject02.sad	0.0193	0.6498	6.2472	0.6684	6.0887	no
subject01.centerlight	subject01.normal	0.6447	0.6705	0.0325	1.3116	0.0305	no
subject02.sleepy	subject02.wink	0.6325	0.6764	0.0334	1.3051	0.0332	no
subject05.surprised	subject05.glasses	0.6441	0.6709	0.0226	1.3111	0.0256	no

Table 3.1: Data-fitted values of the model (3.4) and existence of decay for several experiments performed on texts, mtDNA, songs and face images.

It is worthwile to consider whether other models with the same number of free parameters could fit the experimental data. It is possible, for instance, to get a good 4-th degree polynomial adjustment of these curves in the [0, 1] noise interval, but this would be a consequence of the fact that we are measuring noise as the rate of original information changed. If we had chosen to measure it as the number of changes made in the original streams, the range of our independent variable would be $[0, \infty)$ (see Section 3.1.1). In this case, our model would still be able to fit it without problems, while a polynomial cannot reproduce the asymptotic behavior . Thus, our model is as good as other simpler models with the same number of free parameters, but also shows a correct asymptotic behavior difficult to express with them.

Finally, we show two real clustering experiments performed with the CompLearn Toolkit in the presence of noise. In Figure 3.3 several dendrograms result from clusterings texts by several authors in which several levels of noise have been added to each text. Similar experiments are repeated in Figure 3.4 but this time with mammalian mtDNA.

3.1.3 Discussion

When the NCD is used to compute the distance between two different files, the second file can be considered as a noisy version of the first. Therefore, the effect on the NCD of the progressive introduction of noise in a file can provide information about the measure itself. In this work, we forward a theoretical reasoning of the expected effect of noise introduction, which explains why the NCD can get values greater than 1 in some cases.

A first batch of our experiments confirm the theoretical model. A second batch explores the effects of noise on the precision of clusterings based on the use of the NCD. It can be noticed that the clustering process is qualitatively resistant to noise, for the results do not change much with quite large amounts of it. Different types of files are differently affected, however, which is not surprising: mtDNA files, for instance, which are built on a 4-letter alphabet, are degraded faster than human text, which uses a larger alphabet.

In the future, we intend to tackle a quantitative demonstration of the NCD resistance to noise. We shall also try other metrics and clustering procedures, appropriate to the different file types, to compare their resistance to noise with our NCD results.

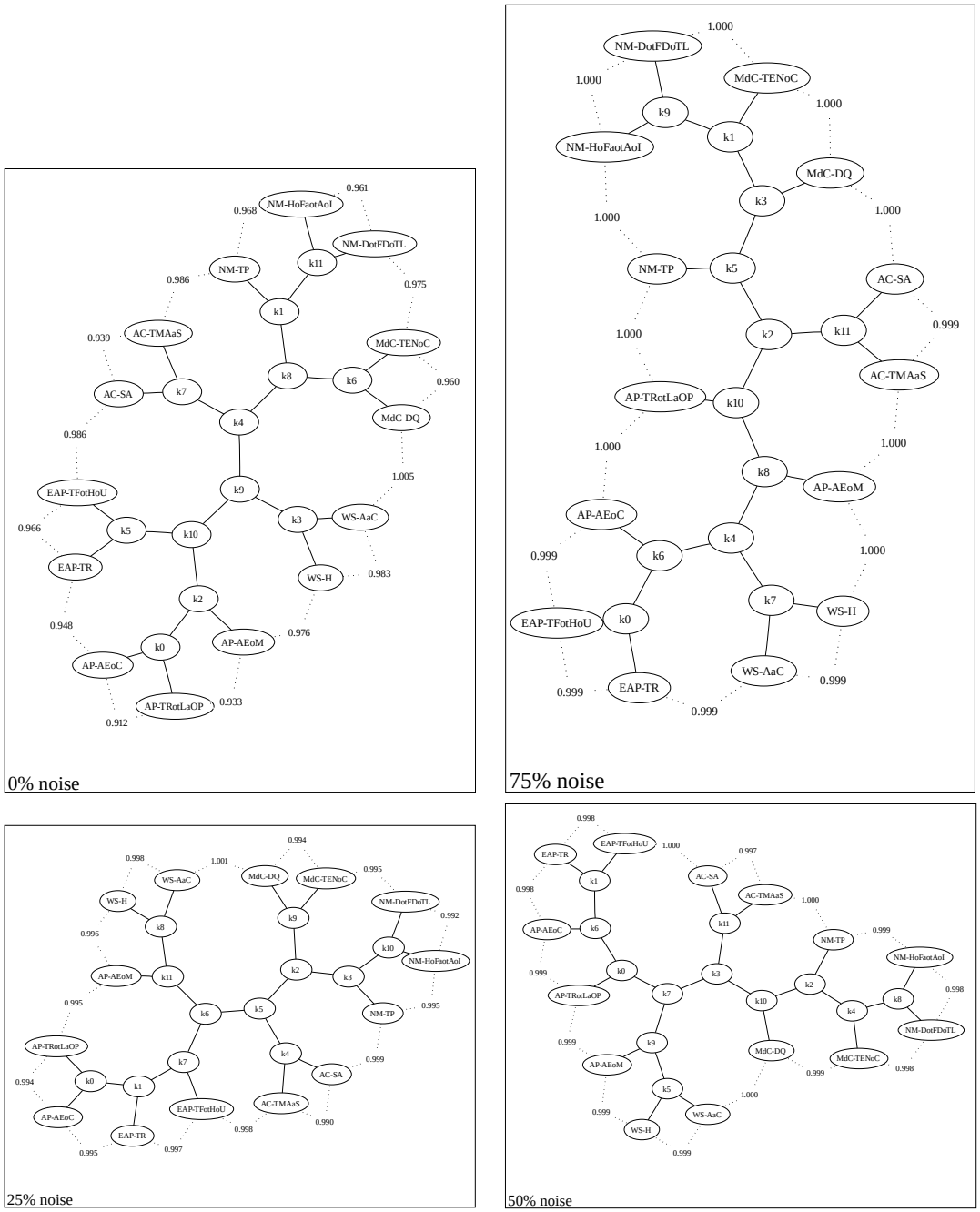

Figure 3.3: NCD-driven clusterings of texts by several authors in which different levels of noise have been added to each sequence. The first characters in the book labels are the initials of their authors: "AC" = Agatha Christie, "AP" = Alexander Pope, "EAP" = Edgar Alan Poe, "WS"= William Shakespeare and "NM" = Nicolò Machiavelli. The quality of the clustering degrades slowly.

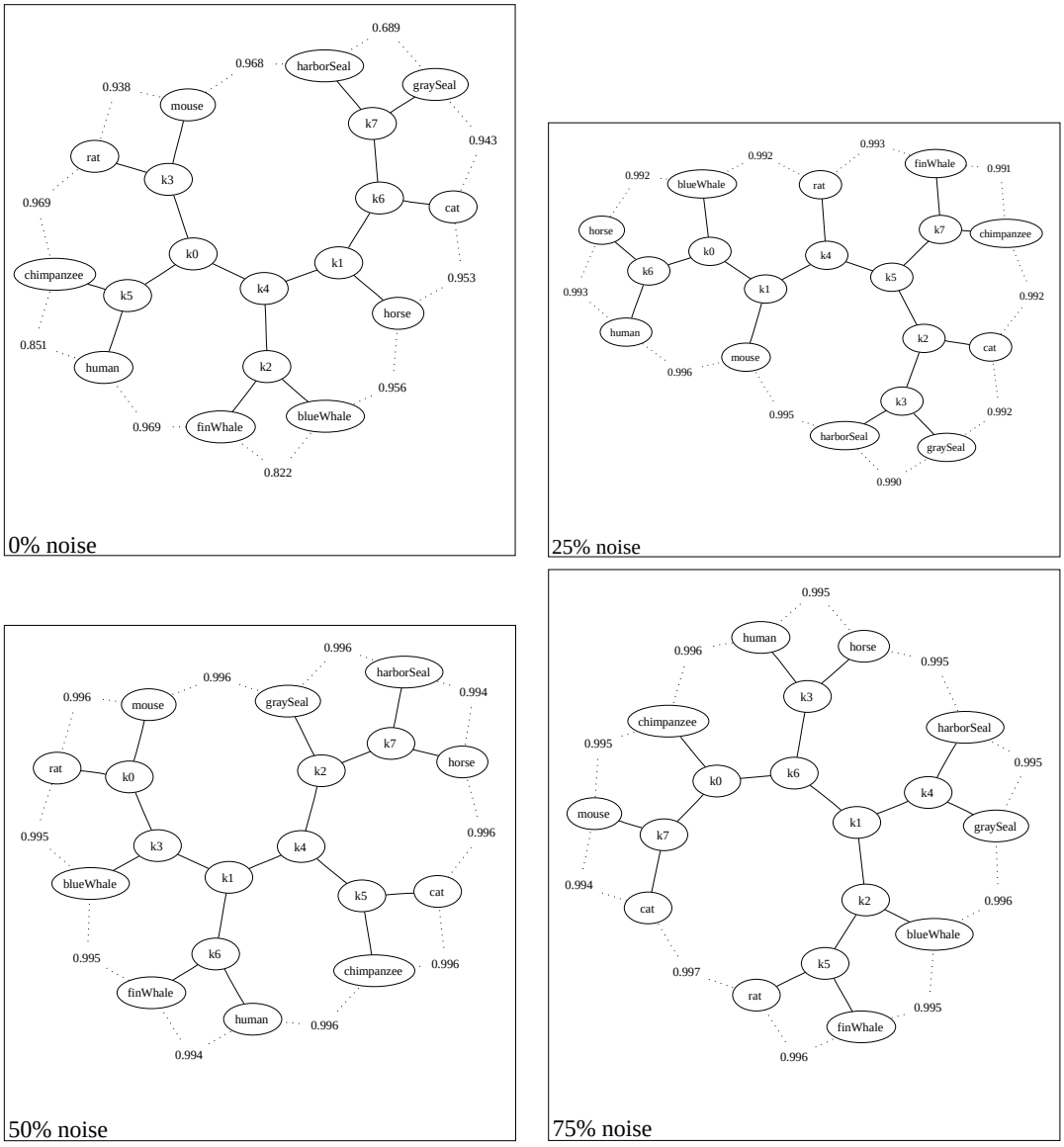

Figure 3.4: NCD-driven clusterings of mammalian mtDNA sequences in which different levels of noise have been added to each element. The quality of the clustering degrades slowly but somewhat faster than in Figure 3.3 due to the faster growth of the average distortion in this type of data.

3.2 Analyzing compressors requirements

In Subsection 2.1.1 (page 14) we introduced the necessary conditions for a distance to be a normalized similarity distance. One implicit need for the NCD to accomplish those conditions is to use compressors which are invariant with respect to the size of the objects. Although this is rather natural, it does not hold for some well-known compressors such as bzip2, gzip, pkzip and many others if the object size exceeds the window size. However, as shown by our results, in the range of usefulness of these compressors, the NCD is very good for its purposes.

Stated formally, we are interested in determining the precision up to which simple conditions like the identity, i.e. $\mathrm{NCD}(x,y) = 0 \iff x = y$ hold for different compressors. It turns out that for compressors using a certain window size, or block size, we obtain $\mathrm{NCD}(x,x)$ close to 1 once we significantly exceed the window size, as the compressors no longer compress. Trivially, in computing the $\mathrm{NCD}(x,y)$ the concatenation xy should comfortably fit the window size or block size. Note that the behavior on (x,x) is possibly different from that on (x,y), with respect to window size. Namely, a window of size $|x|$ sliding over xx has mostly all of x in the window (suffix of first instance, prefix of the next instance). The way in which the identity (of the metric) and the idempotency (of the compressor) are related is the following:

$$x = y \Rightarrow C(xy) - C(x) = O(\log|x|) \Rightarrow \mathrm{NCD}(x,y) = O\left(\frac{\log|x|}{C(x)}\right) \xrightarrow[|x|\to\infty]{} 0$$

These deficiencies observed when measuring identical objects (the easiest scenario) are obviously generalized to any pair of objects. In this way, speaking about identity-idempotency problems is the same as speaking about deficiencies in the whole distance.

In Subsection 3.2.1 we describe the materials we have used to perform our ex/-pe/-ri/-ments. Subsection 3.2.2 presents our results for the bzip2 and gzip compressors, and the anomalous behavior of the distance is analyzed in detail. Finally, in the discussion at the end of this section discussion (Subsection 3.2.3), we provide empirical advice for the correct use of the NCD.

bib	Bibliographic files (refer format)	109
book1	Hardy: Far from the madding crowd	751
book2	Witten: Principles of computer speech	597
geo	Geophysical data	100
news	News batch file	369
obj1	Compiled code for Vax: compilation of progp	21
obj2	Compiled code for Apple Macintosh: Knowledge support system	242
paper1	Witten, Neal and Cleary: Arithmetic coding for data compression	52
paper2	Witten: Computer (in)security	81
paper3	Witten: In search of "autonomy"	46
paper4	Cleary: Programming by example revisited	13
paper5	Cleary: A logical implementation of arithmetic	12
paper6	Cleary: Compact hash tables using bidirectional linear probing	38
pic	Picture number 5 from the CCITT Facsimile test files (text + drawings)	502
progc	C source code: compress version 4.0	39
progl	Lisp source code: system software	70
progp	Pascal source code: prediction by partial matching evaluation program	49
trans	Transcript of a session on a terminal	92

Table 3.2: Name, description and size in Kbytes of the files in the Calgary Corpus.

3.2.1 Materials

This section analyzes the behavior of two real implementations of the distance. The CompLearn toolkit[1] is a package implemented by Rudy Cilibrasi for clustering purposes. The latest version of this package (0.6.2)[2] was used in our experiments. The bzip2 and the gzip compressors can be selected in the toolkit. Our results cover both.

On the other hand, our experimental set is the well known Calgary Corpus, a benchmark for compression algorithms since 1989. Nine different types of text are represented, and to confirm that the performance of schemes is consistent for any given type, many of the types have more than one representative (see table 3.2).

Normal English, both fiction and non-fiction, is represented by two books and six papers (labeled book1, book2, paper1, paper2, paper3, paper4, paper5, paper6). More unusual styles of English writing are found in a bibliography (bib) and a batch of unedited news articles (news). Three computer programs represent artificial languages (progc, progl, progp). A transcript of a terminal session (trans) is included to indicate the increase in speed that could be achieved by applying compression

[1] All the experiments published in [34] were performed using this toolkit.
[2] Available in the Internet at http://www.complearn.org.

to a slow line in a terminal. All of the files mentioned so far use ASCII encoding. Some non-ASCII files are also included: two files of executable code (obj1, obj2), some geophysical data (geo), and a bit-map black and white picture (pic). File geo is particularly difficult to compress, because it contains a wide range of data values, while file pic is highly compressible, because of large amounts of white space in the picture, represented by long runs of zeros. More reasons for choosing this benchmark are explained in reference [146].

3.2.2 Results

In our experiments, all the objects are considered strings of bytes. If x is an object, then x_n is the object composed by the first n bytes of x. Figures 3.5 and 3.13 show the distance $\mathrm{NCD}(x_n, x_n)$ as a function of n in four (bib, book1, book2, pic) of the eighteen files of the Calgary Corpus[3] The files book1 and book2 are selected to be shown because they are the largest ones, while the file bib is selected because of its average size. The reason for showing the file pic is because it is a large but highly compressible file. To analyze the idempotency property, all the objects are compared with themselves.

bzip2

The plots in Figure 3.5, together with many more similar experiments we have performed, show that $\mathrm{NCD}(x, x)$ is between 0.2 and 0.3 in the region where bzip2 can be used properly, while it gives values between 0.25 and 0.9 outside that region.

Two plots in Figure 3.5 (book1 and book2) show two visually different modes of dependency as a function of n. We call *weak dependency* the region starting in 1 Kbytes and ending in 450 Kbytes (the bib and pic files only show this dependency because of the small size of the first one and the high compressibility of the second one). From that point onwards we call it *strong dependency*.

The weak dependency displays a fluctuating slowly-increasing dependence with

[3]The reason for choosing only four of the eighteen files is purely aesthetic. The remaining (not displayed) graphs are available by request to the author.

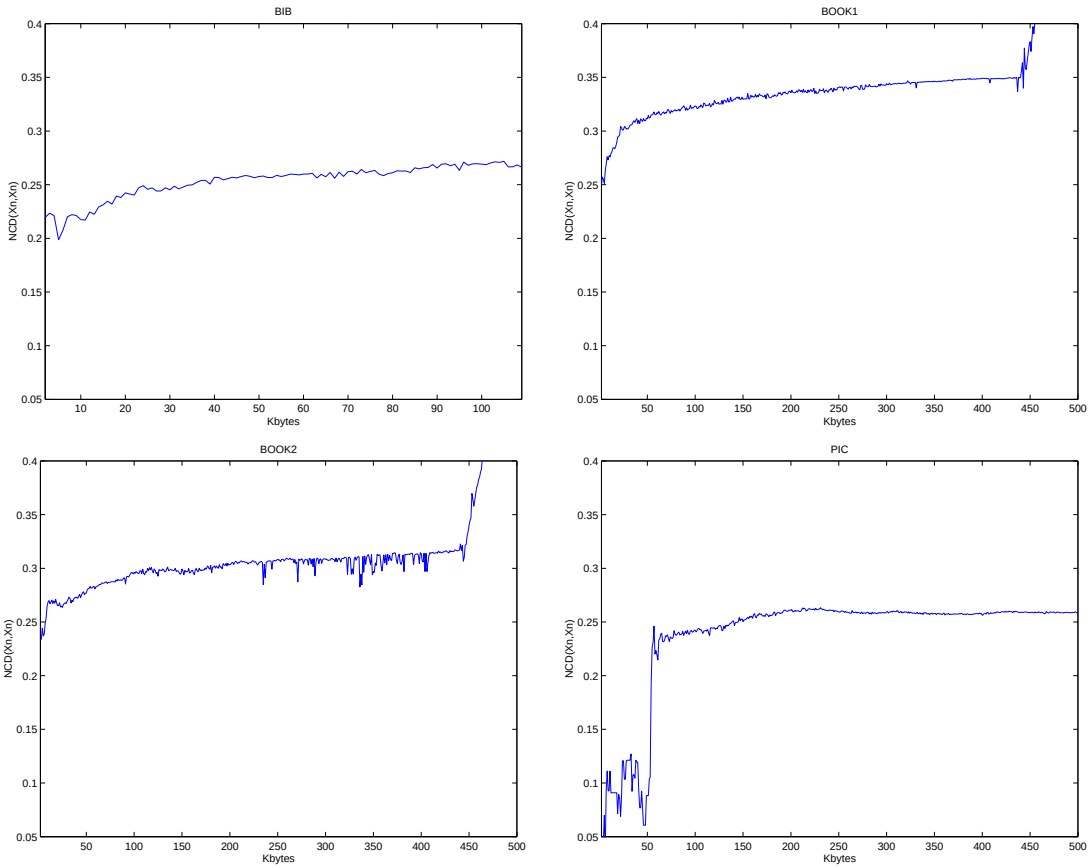

Figure 3.5: Normalized compression distances computed for the first n bytes of four files (bib, book1, book2 and pic, from left to right and top to down) of the Calgary Corpus files using the bzip2 compressor with the --best option.

n. On the other hand, the strong dependence is logarithmic and almost without fluctuations. Both dependencies indicate that the distance is biased by the size of the objects and therefore displays idempotency-identity deviations (see Definitions 1 and 2).

The bzip2 compression algorithm uses three main ideas. In the first stage of the compression, the data suffers a *Burrows-Wheeler transform*; in the second, the *move to front* coding is applied to the output of the transformation; finally a *statistical compressor* (usually Huffman) is used for redundancy extraction. The default block size of the bzip2 compressor is 900 Kbytes which means that, if the size of the object is greater than 900 Kbytes, the object is divided into parts smaller than 900 Kbytes

before being compressed. A more detailed explanation of the algorithms in bzip2 can be found in [28].

Let us start with the weak dependency, which can be observed in the [1 Kbyte, 450 Kbytes] interval, exactly the half size of the block. In this zone, the size of the catenated objects is smaller than 900 Kbytes, thus they do not need to be split.

A simplified example will show how the weak dependency works. Let us assume that the block size is 16 bytes and the object to compress is the string "drdobbs". We need to compute the distance:

$$\mathrm{NCD}(\mathrm{drdobbs}, \mathrm{drdobbs}) = \frac{C(\mathrm{drdobbsdrdobbs}) - C(\mathrm{drdobbs})}{C(\mathrm{drdobbs})} \tag{3.5}$$

The size of the catenated string is 14 bytes, so it fits in a single block. Let us analyze the algorithm step by step:

Burrows-Wheeler transform: A rotation matrix is created from the string "drdobbsdrdobbs" (Figure 3.6). It can be observed that the lower half of the matrix is a repetition of the upper half. Then the matrix is lexicographically sorted and the output for the transformation is the last column of the matrix "oobbrrssddddbb" (Figure 3.7).

move to front coding: the coding is applied and the output is "20103040400030" (see [28]).

Huffman coding: The frequencies of the characters are measured as 0:8, 1:1, 2:1, 3:2, 4:2 and the compressed string is built using 26 bits (see [87]).

Using the same scheme, the string "drdobbs" is compressed using 17 bits, so the distance is NCD $= \frac{26-17}{17} = 0.529$.

Now another symbol "w" is added to the string, so that the new string whose distance with itself we want to measure is "drdobbsw". When the rotation matrix of "drdobbswdrdobbsw" is built and ordered, the first row whose last column has the "w" value will be followed by another row that ends in "w" (in fact both rows will be identical).

d	r	d	o	b	b	s	d	r	d	o	b	b	s
r	d	o	b	b	s	d	r	d	o	b	b	s	d
d	o	b	b	s	d	r	d	o	b	b	s	d	r
o	b	b	s	d	r	d	o	b	b	s	d	r	d
b	b	s	d	r	d	o	b	b	s	d	r	d	o
b	s	d	r	d	o	b	b	s	d	r	d	o	b
s	d	r	d	o	b	b	s	d	r	d	o	b	b
d	r	d	o	b	b	s	d	r	d	o	b	b	s
r	d	o	b	b	s	d	r	d	o	b	b	s	d
d	o	b	b	s	d	r	d	o	b	b	s	d	r
o	b	b	s	d	r	d	o	b	b	s	d	r	d
b	b	s	d	r	d	o	b	b	s	d	r	d	o
b	s	d	r	d	o	b	b	s	d	r	d	o	b
s	d	r	d	o	b	b	s	d	r	d	o	b	b

Figure 3.6: Rotation matrix for "drdobbsdrdobbs".

Just by looking at the string (even without constructing the rotation matrix) we can see that, when the string is coded using move to front, the second "w" of "drdobbswdrdobbs*w*" will get the value 0. This means that the cost of adding "w" to the string will be the cost of coding the first "w" plus the cost of coding the second (only one bit), due to the symmetry of the rotation matrix.

In this way, when we add a symbol π to a string x giving y, the expected difference $C(yy) - C(y)$ will be larger that the expected difference $C(xx) - C(x)$ by just one bit. The codings should not differ too much, because the information of the symbol is the same in both strings, $log(\frac{1}{p(\pi)})$ in Shannon terms. This explains that the weak dependency increases very slowly with n and fluctuates.

In the example, C(drdobbswdrdobbsw) = 33, i.e. after adding two "w" to the original string (16 bits) the size of the compressed version only increases by 7 bits. The new distance is NCD = $\frac{33-22}{22}$ = 0.5, almost identical to that in the original string. The compressor has noticed that the second half of the string is identical to the first, not in a direct way, but by detecting the redundancy of the second half of the string, and therefore coding that half with very few bits.

Let us now explain the strong dependency. The objects in this zone have a size greater than 450 Kbytes, therefore the catenated object has a size greater than the

b	b	s	d	r	d	o	b	b	s	d	r	d	o
b	b	s	d	r	d	o	b	b	s	d	r	d	o
b	s	d	r	d	o	b	b	s	d	r	d	o	b
b	s	d	r	d	o	b	b	s	d	r	d	o	b
d	o	b	b	s	d	r	d	o	b	b	s	d	r
d	o	b	b	s	d	r	d	o	b	b	s	d	r
d	r	d	o	b	b	s	d	r	d	o	b	b	s
d	r	d	o	b	b	s	d	r	d	o	b	b	s
o	b	b	s	d	r	d	o	b	b	s	d	r	d
o	b	b	s	d	r	d	o	b	b	s	d	r	d
r	d	o	b	b	s	d	r	d	o	b	b	s	d
r	d	o	b	b	s	d	r	d	o	b	b	s	d
s	d	r	d	o	b	b	s	d	r	d	o	b	b
s	d	r	d	o	b	b	s	d	r	d	o	b	b

Figure 3.7: Lexicographically ordered rotation matrix for “drdobbsdrdobbs”.

block size of bzip2 (900 Kbytes). In our explanation, we will assume that the block size is 8 bytes, and will use a string that, catenated to itself, is bigger than that size. We use the string “drdobbs” again, so when compressing “drdobbsdrdobbs” it must be split in the two strings “drdobbsd” and “rdobbs”. Let us apply the compression algorithm to both:

Burrows-Wheeler transform: The rotation matrix for the two strings (Figures 3.8 and 3.9) is built and ordered (Figures 3.10 and 3.11). There is a big difference between having the whole string in one block or in two: the upper-lower half symmetry is lost due to the splitting, and much redundancy achieved in the weak dependence zone is not achieved here. The outputs are “obsrdddb” and “obrdsb”, whose equal characters are much less grouped[4] than in the previous example “oobbrrssddddbb”.

move to front coding: The coding of both strings is “21444003” and “213343”.

Huffman coding: The character frequencies are 0:2, 1:1, 2:1, 3:1, 4:3 for the first string and 1:1, 2:1, 3:3, 4:1 for the second one. The resulting output built using

[4]Grouping identical characters is the main purpose of the Burrows-Wheeler transform.

d	r	d	o	b	b	s	d
r	d	o	b	b	s	d	d
d	o	b	b	s	d	d	r
o	b	b	s	d	d	r	d
b	b	s	d	d	r	d	o
b	s	d	d	r	d	o	b
s	d	d	r	d	o	b	b
d	d	r	d	o	b	b	s

Figure 3.8: Rotation matrix for "drdobbsdrd".

b	b	s	d	d	r	d	o
b	s	d	d	r	d	o	b
d	d	r	d	o	b	b	s
d	o	b	b	s	d	d	r
d	r	d	o	b	b	s	d
o	b	b	s	d	d	r	d
r	d	o	b	b	s	d	d
s	d	d	r	d	o	b	b

Figure 3.10: Lexicographically ordered rotation matrix for "drdobbsdrd".

r	d	o	b	b	s
d	o	b	b	s	r
o	b	b	s	r	d
b	b	s	r	d	o
b	s	r	d	o	b
s	r	d	o	b	b

Figure 3.9: Rotation matrix for "rdobbs".

b	b	s	r	d	o
b	s	r	d	o	b
d	o	b	b	s	r
o	b	b	s	r	d
r	d	o	b	b	s
s	r	d	o	b	b

Figure 3.11: Lexicographically ordered rotation matrix for "rdobbs".

Huffman coding has a size of 30 bits (4 more than when using a single block). The main question is not the final size of the string, but the fact that the second half of the string has been coded using 16 bits, while it was coded using only 8 bits in the single block example (exactly double). Splitting the string has caused a worse character-grouping when the Burrows-Wheeler transform is performed. The long distance redundancies of the string have been lost, and this introduces a bias in the distance: the same objects are now farther apart: NCD $= \frac{30-17}{17} = 0.765$.

If the string is divided in two blocks, the expected compression cost of adding a symbol π to x will be $2\log(\frac{1}{p(\pi)})$, therefore an expected increase of $\log(\frac{1}{p(\pi)})$ in $C(xx) - C(x)$. This explains the logarithmic growth of the strong dependency zone.

In order to compare both dependencies, it should be remembered that the expected

compression cost of adding one symbol is $\log(\frac{1}{p(\pi)}) + 1$, therefore the expected value of $C(xx) - C(x)$ is 1 bit when the concatenated string fits into a single block.

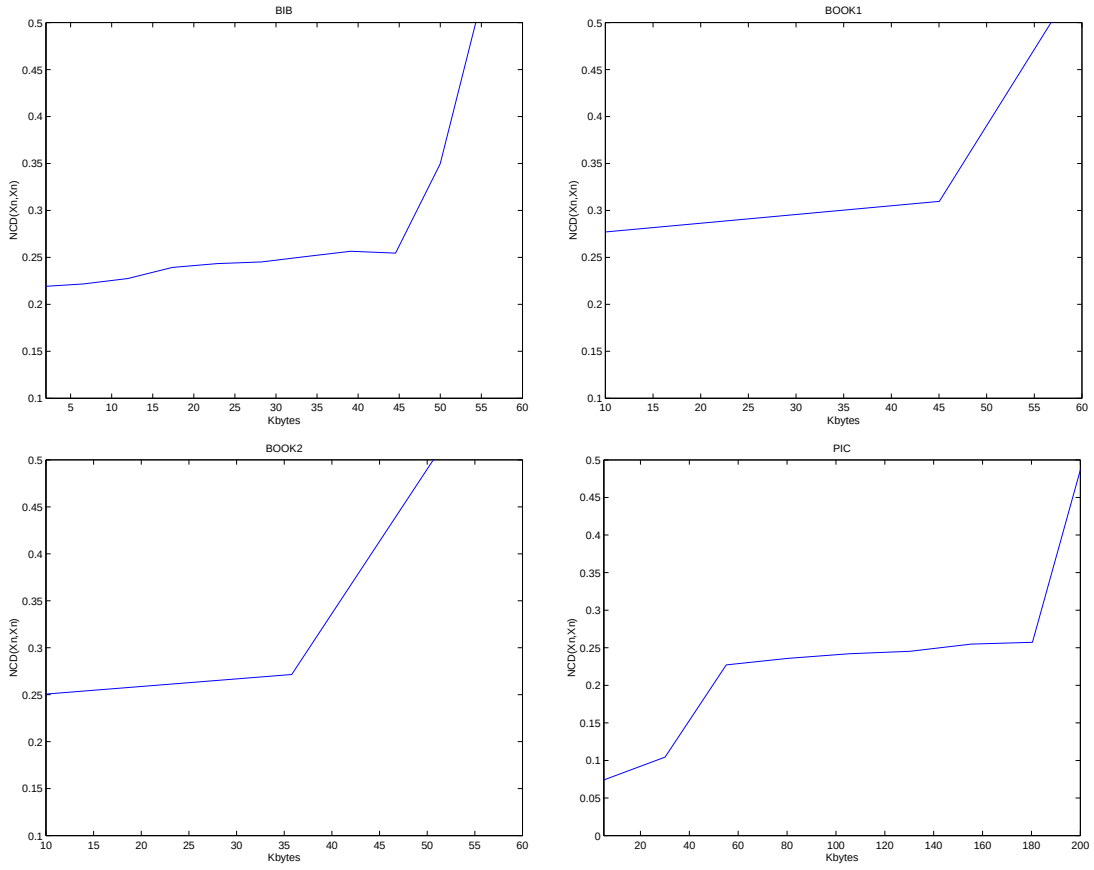

Figure 3.12: Normalized compression distances computed for the first n bytes of four files (bib, book1, book2 and pic, from left to right and top to down) of the Calgary Corpus files using the bzip2 compressor with the --fast option.

We have repeated our experiments with bzip2 in a different situation, by selecting the --fast option rather than the --best option (see Figure 3.12). In this case, the block size used by the compressor can be seen to be much smaller (about 100 Kbytes, vs. 900 in the --best case) which means that files over 50 Kbytes are not properly managed. Even the small files in our examples suffer now from this effect and show a strong dependency region.

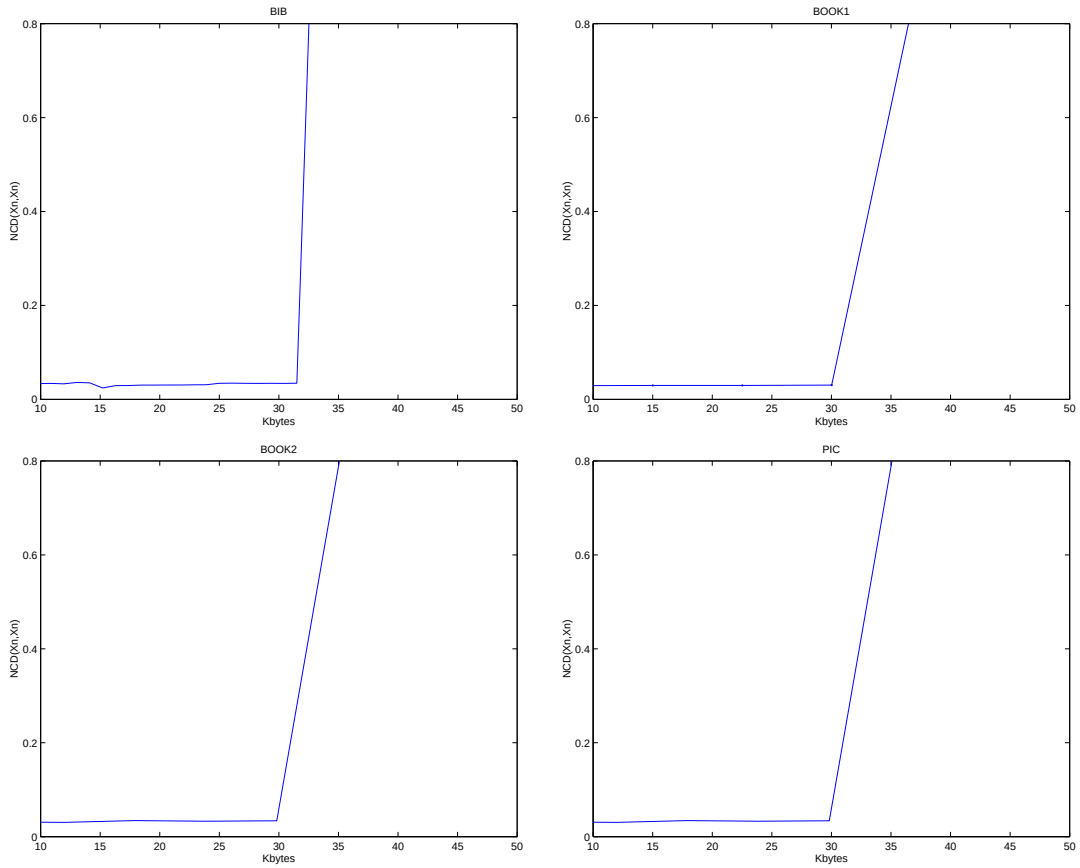

Figure 3.13: Normalized compression distances computed for the first n bytes of four files (bib, book1, book2 and pic, from left to right and top to down) of the Calgary Corpus files using the gzip compressor with the `--best` option.

gzip

The plots in Figure 3.13, together with many more similar experiments we have performed, show that NCD(x, x) is between 0.0 and 0.1 in the region where gzip can be used properly, while it gives values which grow to 1 outside that region.

The experimental results obtained using the gzip compressor in the NCD are displayed in Figure 3.13. We can observe an initial slow-fluctuating growth with n, followed by a strong discontinuity, with a jump to 0.9 at 32 Kbytes, and finally a new slow (but slightly faster) growth, until the distance saturates in 1. We will call again the two zones weak dependency and strong dependency, for analogy with the

previous subsubsection.

The kernel of gzip [3] uses a variant of the LZ77 algorithm [155] for preprocessing and a statistical compressor (usually Huffman) as post-processing. The bias caused by the object size is fully explained by the compression scheme of the LZ77 algorithm[5].

As in the previous subsubsection, the two modes will be explained by means of three simple examples. The string to be compared with itself is again "drdobbs". This time there are two parameters that play the same role as the block size in the bzip2 compressor: the sliding window and the lookahead window. The sliding window W_S is a buffer that contains the previous $|W_S|$ characters to the character that is being compressed. On the other side, the lookahead window W_L is the buffer that contains the next $|W_L|$ characters that follow the character being coded. The LZ77 algorithm searches the longest string that begins in the current coding character and is contained in both windows.

current coding character	W_S	W_L	coded string
$\overset{\circ}{d}$rdobbsdrdobbs	empty	drbobbs	0_1
d$\overset{\circ}{r}$dobbsdrdobbs	d	rbobbsd	$0_1 0_1$
dr$\overset{\circ}{d}$obbsdrdobbs	dr	bobbsdr	$0_1 0_1 2_1$
drd$\overset{\circ}{o}$bbsdrdobbs	drd	obbsdrd	$0_1 0_1 2_1 0_1$
drdo$\overset{\circ}{b}$bsdrdobbs	drdo	bbsdrdo	$0_1 0_1 2_1 0_1 0_1$
drdob$\overset{\circ}{b}$sdrdobbs	drdob	bsdrdob	$0_1 0_1 2_1 0_1 0_1 1_1$
drdobb$\overset{\circ}{s}$drdobbs	drdobb	sdrdobb	$0_1 0_1 2_1 0_1 0_1 1_1 0_1$
drdobbs$\overset{\circ}{d}$rdobbs	drdobbs	drdobbs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 7_7$

Figure 3.14: W_S= 7 bytes, W_L = 7 bytes.

In our first example, let's assume that $|W_S| = |W_L| = 7$ bytes. Remember that we want to compute $\frac{C(xx)-C(x)}{C(x)}$. The LZ77 algorithm is applied to the string "drdobbsdrdobbs". The sliding window and the lookahead window were large enough, and the compressor realized that the second half of the string is an exact repetition of the first. Let us assume that each compression chunk $\text{offset}_{\text{length}}$ has a size of 2 bytes

[5]The Huffman coding does not have a relevant influence in the bias, so it is left out of this explanation.

current coding character	W_S	W_L	coded string
$\overset{\circ}{d}$rdobbsdrdobbs	empty	drd	0_1
d$\overset{\circ}{r}$dobbsdrdobbs	d	rdo	$0_1 0_1$
dr$\overset{\circ}{d}$obbsdrdobbs	dr	dob	$0_1 0_1 2_1$
drd$\overset{\circ}{o}$bbsdrdobbs	drd	obb	$0_1 0_1 2_1 0_1$
drdo$\overset{\circ}{b}$bsdrdobbs	drdo	bbs	$0_1 0_1 2_1 0_1 0_1$
drdob$\overset{\circ}{b}$sdrdobbs	drdob	bsd	$0_1 0_1 2_1 0_1 0_1 1_1$
drdobb$\overset{\circ}{s}$drdobbs	drdobb	sdr	$0_1 0_1 2_1 0_1 0_1 1_1 0_1$
drdobbs$\overset{\circ}{d}$rdobbs	drdobbs	drd	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 7_3$
drdobbsdrd$\overset{\circ}{o}$bbs	obbsdrd	obb	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 7_3 7_3$
drdobbsdrdobb$\overset{\circ}{s}$	sdrdobb	s	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 7_3 7_3 7_1$

Figure 3.15: W_S= 7 bytes, W_L = 3 bytes.

(Figure 3.14). In this way $\frac{C(xx)-C(x)}{C(x)} = \frac{2}{14} = 0.143$. We can generalize: whatever the size of x, if the windows are large enough, $C(xx) - C(x) = 2$.

A new scenario is proposed: now $|W_S| = 7$ and $|W_L| = 3$ (see Figure 3.15). In this example, the compressor was unable to extract all the redundancy from the string, due to the insufficient size of the lookahead window. Rather than detecting that the second half of the string is identical to the first, the compressor only detects three substrings identical to three other substrings in the sliding window. This is what underlies the weak dependency. For the Calgary Corpus, the window size (32 Kbytes) is larger than the size of the object, but the lookahead window is smaller. This is why the NCD increases slightly with n in this zone: $C(xx) - C(x)$ is proportional to $\frac{|x|}{|W_L|}$. In our example, the distance has significantly increased: NCD = $\frac{20-14}{14} = 0.428$. This is a deviation in the distance, which depends little on the size of the objects.

It remains to explain the most important feature, the discontinuity point at 32 Kbytes. In this point, the size of the catenated object overflows the size of the sliding window.

In our last example, we will assume that $|W_S| = 6$ and $|W_L| = 7$. The results are shown in Figure 3.16. The insufficient size of the sliding window causes the first byte of the string to be unreachable by the compressor, which loses all the redundancy

current coding character	W_S	W_L	coded string
$\overset{\circ}{d}$rdobbsdrdobbs	empty	drdobbs	0_1
d$\overset{\circ}{r}$dobbsdrdobbs	d	rdobbsd	$0_1 0_1$
dr$\overset{\circ}{d}$obbsdrdobbs	dr	dobbsdr	$0_1 0_1 2_1$
drd$\overset{\circ}{o}$bbsdrdobbs	drd	obbsdrd	$0_1 0_1 2_1 0_1$
drdo$\overset{\circ}{b}$bsdrdobbs	drdo	bbsdrdo	$0_1 0_1 2_1 0_1 0_1$
drdob$\overset{\circ}{b}$sdrdobbs	drdob	bsdrdob	$0_1 0_1 2_1 0_1 0_1 1_1$
drdobb$\overset{\circ}{s}$drdobbs	drdobb	sdrdobb	$0_1 0_1 2_1 0_1 0_1 1_1 0_1$
drdobbs$\overset{\circ}{d}$rdobbs	rdobbs	drdobbs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1$
drdobbsd$\overset{\circ}{r}$dobbs	dobbsd	rdobbs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1 0_1$
drdobbsdr$\overset{\circ}{d}$obbs	obbsdr	dobbs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1 0_1 2_1$
drdobbsdrd$\overset{\circ}{o}$bbs	bbsdrd	obbs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1 0_1 2_1 0_1$
drdobbsdrdo$\overset{\circ}{b}$bs	bsdrdo	bbs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1 0_1 2_1 0_1 6_1$
drdobbsdrdob$\overset{\circ}{b}$s	sdrdob	bs	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1 0_1 2_1 0_1 6_1 1_1$
drdobbsdrdobb$\overset{\circ}{s}$	drdobb	s	$0_1 0_1 2_1 0_1 0_1 1_1 0_1 5_1 0_1 2_1 0_1 6_1 1_1 0_1$

Figure 3.16: W_S= 6 bytes, W_L= 7 bytes.

detection. A very small change (only 1 byte) in the sliding window size can cause the compressor to be absolutely blind (the NCD calculation in this example gives $\frac{28-14}{14} = 1$. This is what causes the discontinuity at 32 Kbytes. When the size of the object is one byte more than the sliding window, the first byte of the first object is lost, and the compressor becomes unable to detect the full redundancy of the catenation, giving rise to an NCD value near to absolute dissimilarity (0.9 for almost all files in the Calgary Corpus).

The purpose of using the LZ77 algorithm in the NCD is based on the fact that it can use the sequences that appear in the first object to make the coding of the second object less expensive. If the size of the sliding window is significantly smaller than the size of any of the objects, the blind effect will outperform the redundancy detection task.

From all performed experiments (those described in this subsubsection and many others not shown) we can extract the following conclusion: if $|W_S| \ll |x|$ or $|W_S| \ll |y|$

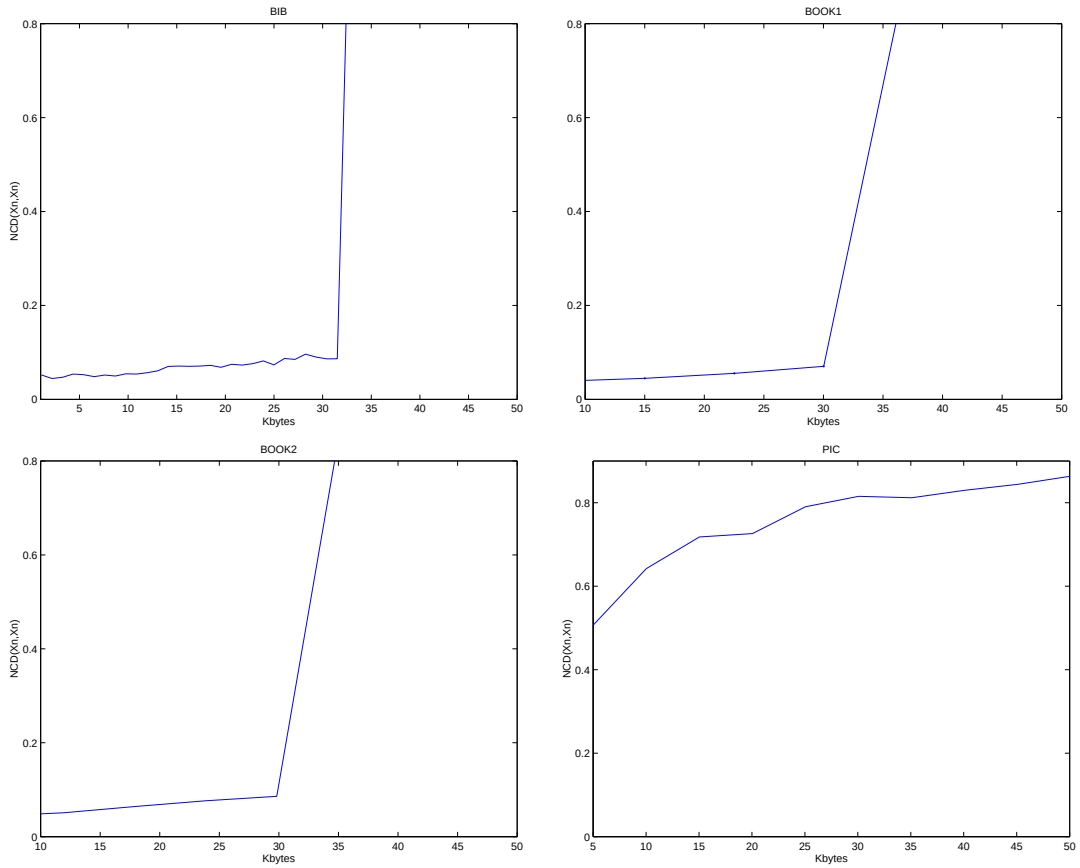

Figure 3.17: Normalized compression distances computed for the first n bytes of four files (bib, book1, book2 and pic, from left to right and top to down) of the Calgary Corpus files using the gzip compressor with the --fast option.

then $\mathrm{NCD}(x, y) \approx 1$ for any possible value of $\max\{K(x|y), K(y|x)\} / \max\{K(x), K(y)\}$, i.e. for any similarity degree between x and y.

We have also repeated our experiments with gzip, by selecting the --fast option rather than the --best option (see Figure 3.17). In this case, the size of the sliding window used by the compressor does not change, so the results obtained are very similar to those with the --best option. Only the compression ratio obtained is affected (see table 3.3).

compressor	options	comp. ratio	acceptable region (Kbytes)
ppmz	none	25%	$(0, \infty)$
bzip2	--best	27%	[1, 900]
bzip2	--fast	29%	[1, 100]
gzip	--best	32%	[1, 64]
gzip	--fast	38%	[1, 64]

Table 3.3: Comparison table over the Calgary Corpus for all compressors and options used. For a proper use of the NCD, the addition of the sizes $|x| + |y|$ of the objects x, y involved in the computation of NCD(x, y) should be in the acceptable region.

3.2.3 Discussion

In this section we have analyzed the impact on the NCD quality of some features of two compressors: the block size in bzip2, and the sizes of the two windows (sliding and lookahead) used by gzip. The well-known Calgary Corpus has been used as a benchmark. Any similarity distance should measure a 0 distance (or, at least, a very small value) between two identical objects. The empirical results obtained with both compressors for the Calgary Corpus reveal that the NCD is biased by the size of the objects, independently of their type. For object sizes smaller than certain values (related to the block and window sizes in the compressors), the distance between two identical objects is usually quite small, which proves that the NCD is a good tool for this purpose. However, for larger sizes, when the inner limitations of the compressors are violated, obviously the distance between two identical objects grows to very high values, making the NCD practically unusable. Other widely used compressors (such as winzip and pkzip) also show the same limitations.

The use of block and window sizes in the compressors aims to increasing the computation speed at the expense of the compression ratio. This section proves that this balance between quality and speed should be treated carefully for clustering, where quality is tantamount. When considering clustering problems, all considerations about speed should be left apart if they imply exceeding the system parameters. The proper use of this powerful distance depends on selecting compression algorithms without limiting factors related to the size of the objects, such as the high compression Markov predictive coder PPMZ [91], which does not set any window or block

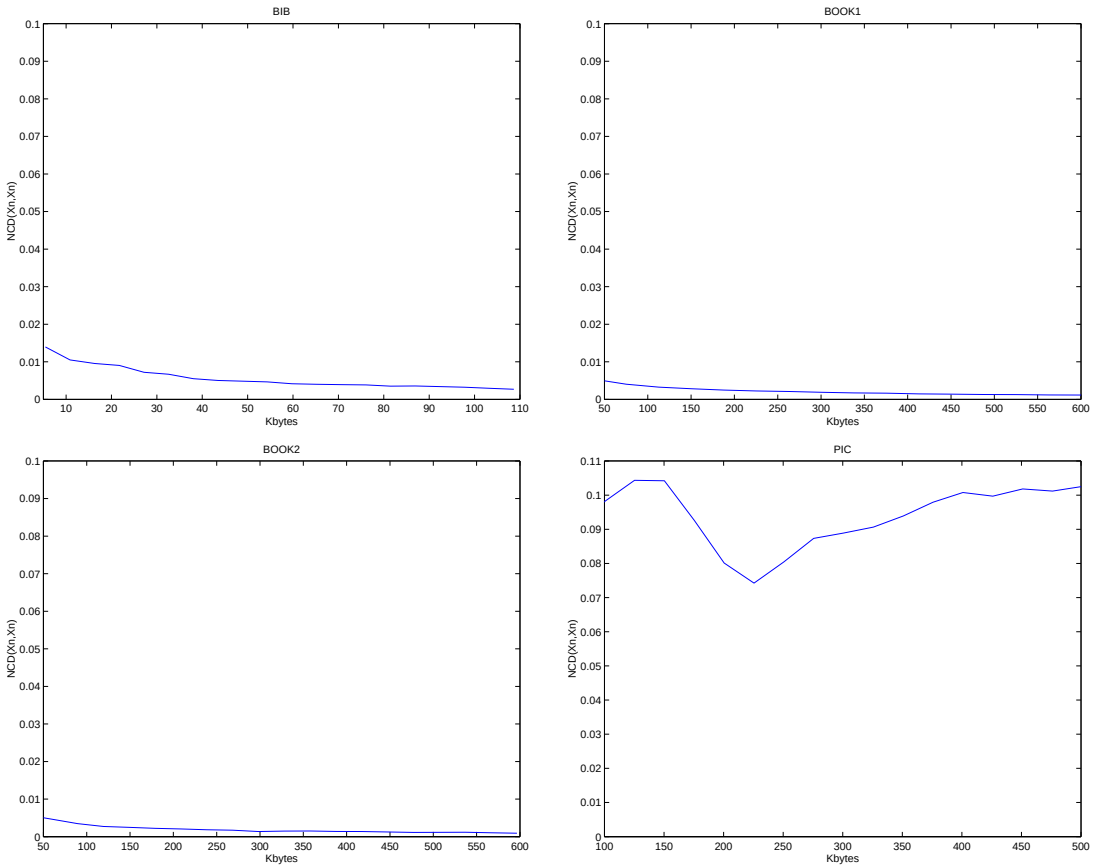

Figure 3.18: Normalized compression distances computed for the first n bytes of four files (bib, book1, book2 and pic, from left to right and top to down) of the Calgary Corpus files using the PPMZ compressor.

limit, but is much slower than those mentioned above. The results of using PPMZ in our experiments are shown in Figure 3.18 and are coherent with our conclusions: the distance computed with PPMZ does not depend on the size of the objects and is always between zero and a very small value (0.1043). On the other hand, this also confirms that the NCD is a very good distance measurement, when used in the proper way.

In the case of bzip2 and gzip, the block, the sliding window and the lookahead window should be at least as large as the sum of the sizes of the objects to be compared. The table in Figure 3.3 summarizes the results obtained for all three compressors under different circumstances, both as regards the compression ratio

obtained and the size limits where the use of the NCD is acceptable for each. For a proper use of the NCD, the addition of the sizes $|x| + |y|$ of the objects x, y involved in the computation of $\mathrm{NCD}(x, y)$ should be in the acceptable region.

Chapter 4

New applications of Algorithmic Information Theory

In this chapter we present three new applications of Algorithmic Information Theory, which are specially related to the document similarity framework it provides: Common Source Data Detection, Source Code Plagiarism Detection and Music Generation.

4.1 Common Source Data Detection

Because the common text compression algorithms are based on the presence of repetitions in the text, it is clear that the only situation detectable by the compression algorithms is when literal repetitions of the texts are present.

This essentially leads to a paradoxical situation. Namely, one and the same text, written using two different, non-intersecting alphabets, are classified as totally different, because the compression of the catenation of the texts t_1 and t_2 is $C(t_1 \circ t_2) = C(t_1) + C(t_2)$ and thus $\mathrm{NCD}(t_1, t_2) = 1$. This can be very easily tested, for example, using the Cyrillic and the Latin version of the Universal Declaration of the Human Rights in Serbian[1]. If we care to eliminate the structure, imposed by the

[1]Serbian language has the peculiarity, that it can use both alphabets – Latin and Cyrillic usually not mixing them in one and the same text.

formatting rules of the document (white space and enumeration), we actually find these texts very dissimilar. But the two texts, on the other side, are exactly the same.

Therefore, the interesting question arises: Can we find an algorithm that can detect translations of one and the same text?

In this article we present a Lempel-Ziv (LZ) [155], inspired algorithm, that can detect whether some text is a translation of another text.

To make the introduction clearer, we explain LZ in a few sentences: LZ parses the string in one direction. If the string is coded up to some position p, the next portion of the string is coded by finding the position q in the already coded portion such that (1) the substrings starting at the positions p and q coincide; (2) the coinciding substring has the maximal length of all such strings and (3) if there exist more than one positions q with these properties, the maximal q is chosen. Thus, the portion of the text that is coded is represented as a triplet $(p-q, l, s)$, consisting of the displacement from the current position $p-q$, the length l of the substring that coincides and the symbol s, that follows the coinciding part of the text.

There are two very different types of textual string data. The first type is text data, usually produced by humans, in which some concept is represented by using unidimensional character strings. This type of textual data, essentially includes cross references and therefore repetitions imposed by the nature of the concept, as well as structure imposed by the rules of description of the concept, as for example language syntax and morphology. Examples of these kinds of string data are the textual data produced directly by humans and computer programs. Usually, this type of data is compressed well by LZ. The compression is due to the high degree of predictability of the future of the text, looking at its past.

Actually, there are two sources of the ability to achieve compression in these texts. On the one side, the rules that ought to be imposed to transmit the information, as for example, formatting rules and the grammar (how many "the" in this text!) and on the other side, the structure of the concept itself that ought to be transmitted (how many "compress..." in this text!). Using LZ, the grammar can achieve coding, with low displacements $p-q$, e.g. short-range coding. The context can achieve long-range

coding.

On the other hand, there are text-like data that are of very different nature. This type of data does not include any grammar[2] and the coding of the text is usually the product of some stochastic process. The common characteristic is that the next symbol is very unpredictable from the previous string. Usually, algorithms as LZ do not function on this type of data. A common characteristic of these type of data is that, coding the data using entropy coding symbol by symbol, for example, by using arithmetic coding [7], we achieve better compression than using structure based algorithms as LZ, BWT [28] or PPM [91]. Such kinds of data are, for example DNA sequences and financial series of fluid markets. Usually, for these types of data there exist some common "reason" for the string. For example, in the DNA sequence, this "reason" can be the common predecessor and in the case of the financial data one of the reasons can be the political environment. In the examples given, it seems that the representation is preserved (the genetic code is the same), but the structure of the underlying "concept", if existing, is poorly expressed. For example, in the case of genetic material, the cross-similarities between the whole genomes of two different species are higher than the intra-similarities between two pieces of the genome of the the same organism, because this material is a result of a random evolution process with restrictions (the survival of the organism).

Nevertheless, the measure $\mathrm{NCD}(t_1, t_2)$ is useful in these cases, because although the separate strings do not compress in any way, the concatenation of the two strings compresses well, due to the fact that the texts are clearly similar and the cross-references between t_1 and t_2 prevail in the compression of the second string in $t_1 \circ t_2$. Therefore, the compression of this type of strings is a consequence of its cross-similarity due to the common process that generates the data.

Summarizing, the reasons for the compression of some set of similar strings is the structure, imposed due to (1) *the coding rules*, the structure, imposed due to (2) *the intrinsic structure of the transmitted concept* and the structure imposed due to (3) *the common initial source of the data.* In all the cases, we can regard that one of the strings is in a way a translation of the other. In this translation, the coding rules

[2]Except trivial ones like $[agtc]*$ for the genetic code.

can change, but the common source and the intrinsic structure of the concept are preserved.

It is clear, that detecting and separating the effect of these sources is, at least, an interesting task.

In this section we are trying to find these effects, using LZ code representation of the string and ignoring the characters from the alphabet (the member s in the triplet). We will process the LZ code of a concatenation of a set of strings $L(t_1 \circ t_2 \circ t_3 ... \circ t_n)$, where we suppose that t_i is a translation of t_j. We extract all the information from the LZ coding and try to separate the types of compressibility from this information. In this work we attempt to diminish the influence of the coding rules of the string (e.g. the grammar and the morphology).

This section is organized as follows: Subsection 4.1.1 briefly surveys existing approaches. Subsection 4.1.2 explains the concepts and the algorithms used. Subsection 4.1.3 describes the data and the computer experiments that were carried out and gives a brief comment of each of them. Subsection 4.1.4 gives a rather simplified phenomenological model of the text translation similarity observed and Subsection 4.1.5 offers a summary of the results.

4.1.1 Related work

The problem of automatic translation detection, known in the literature as *parallel text*, *parallel corpora* or *bitext* detection, dates from 1998 with Resnik's seminal paper [134]. Since that moment, efforts towards solving this problem can be clearly separated in two classes: *structure based* and *content based*.

All structure based proposals are strongly oriented to text documents with machine oriented markups, specially Web pages, and are generally divided in three stages:

1. Locating pages: Web pages which contain links to its information translated into other languages are searched trough Web search engines like AltaVista and Google. This process is automatic and is achieved trough predefined queries to the search engines.

2. Generating candidate pairs: the Web pages obtained from the previous step are

investigated by means of several techniques: automatic language identification [89], intelligent URL-matching [134, 135] or document length filtering [142].

3. Structural filtering: the HTML markups are used to create a syntax skeleton of the document on which several matching algorithms can be applied. Most of the non-markup contents of the document are ignored. Several measures of matching quality are obtained, and according to some hand-tuned or machine-learned thresholds, documents are classified as translations from another.

Some examples of this architecture are STRAND [134, 135], PTMiner [32] and BITS [110].

The content based approach, was developed latter and with the aim to fill the gap of translations where markups are absent. The most representative work is the derivation of the *tsim* similarity score [143]; in that article, Smith proposes two levels of matching. First, a translation lexicon is used to find the best matching of words in text pairs, posed as a bipartite matching problem. The second level matches the documents according to their similarity score.

Both approaches were synergistically joint in [136] by the leading authors of the two approaches, Resnik and Smith. In that article the authors use machine learning to design decision trees that classify the text as translated or not using the features extracted from the markup matching algorithm (structure based) and the *tsim* score (content based).

Up to our knowledge of the literature, these approaches are the most similar to ours in tackling this problem. In this section we are trying to design a simpler algorithm that relies only on the statistical properties of text, which is not necessarily human written and does not use lexicon.

4.1.2 The algorithm

We are going to measure the similarity between two texts t_1 and t_2. To begin with, we apply the LZ algorithm to each text obtaining the typical LZ-triple set $\{(p - q, l, s)\}$ consisting of the displacement of the current position $p - q$, the length l of the substrings that coincide at positions p and q and the symbol s, that follows the

coinciding part of the text. We do not care about the alphabet but only about its compression structure, so we can leave apart the symbol s, obtaining an equivalent set $G_{\mathrm{LZ}} = \{(q, p, l)\}$, which can be interpreted as a graph with the positions of the text p and q as vertices, and edges between them, with weight $l > 0$ which is the length of the identical string in the positions p and q. This graph is extremely sparse.

If the substrings with length l in the positions p and q are identical, then the substrings at position $p + 1$ and $q + 1$ are also identical with length $l - 1$, the same happens with $p + 2, q + 2, l - 2$ and so on. Therefore, we can increase the density of the graph, by defining:

$$G^0 = \{(q + i, p + i, l - i), (p + i, q + i, l - i) \mid (p, q, l) \in G_{\mathrm{LZ}},\ 0 \leq i < l\}$$

Members of G^0 with small l are generally imposed by the grammar and also by random matches between short strings due to the limited alphabet. Although the grammar is an interesting aspect, we prefer to ignore it in the present work and we prune the graph by deleting the edges with small weights. We can simplify the consideration by regarding all edges of weight more than some limit L as equivalent and thus, instead of regarding the weighted graph G^0, we consider a normal graph G^1_L defined as:

$$G^1_L = \{(q, p) \mid (q, p, m) \in G^0,\ m \geq L\}.$$

Some edges remain to be added to complete our graph: if we have edges (p, q) and (q, r) in G^1_L, it is clear that we have to add edges (q, p), (p, r) and (r, p) because of two reasons: (i) p can be compressed with position r, (ii) p and r belong to substrings with lengths greater than L which LZ identified as useful for achieving compression. This process can continue iteratively, until finally each node will be connected to all its reachable nodes, which is the transitive closure of the graph:

$$G_L = \{(q, p) \mid \text{there exists a path from } p \text{ to } q \text{ in } G^1_L\}.$$

In order to compare the structure of two texts t_1, t_2, strictly speaking, we must compare the structure of the graphs $G_L(t_1)$ and $G_L(t_2)$. However, as we will see, this

level of detail is not necessary when dealing with human written texts. In these cases we will compare instead of $G_L(t_1)$ and $G_L(t_2)$ just the degrees of the nodes of the graphs. As the two texts t_1 and t_2 may have different length (number of positions) their number of nodes may be different and therefore their degree functions may have different ranges, which can be a drawback for a direct comparison. For example, the Russian version of the universal declaration of human rights can be coded with KOI8-r, which means that each character will occupy 1 byte. The same text coded using UTF-8 will use two bytes per Cyrillic character and therefore the length of the text will be almost doubled.

To overcome situations like this, we choose an arbitrary but fixed number of bins $B \ll N$ and unite several text positions of the text (vertices of G_L) in one vertex.

More exactly, we define the function $\deg_{L,B}(t;k)$ as the number of edges in $G_L(t)$ of the vertices p with $k = \lfloor Bp/(N+1) \rfloor$. We will omit the parameter k if we refer it as a vector index. Of course, we have to choose the same B for the objects being compared. In this way, the two binned degree functions of both Russian versions will be very similar.

The scale parameter B can be chosen in a way to achieve enough statistics for the estimation of the density of each bin k which in practical terms means to have some 10 edges per bin [126].

Once we have $\deg_{L,B}(t_1)$ and $\deg_{L,B}(t_2)$ in the same, unidimensional range, $[0, B-1]$, we can compute the distance between these two values in many different ways. However, for this study even a simple correlation ρ of the smooth averages of these these functions $\rho(\deg_{L,B}(t_1), \deg_{L,B}(t_2))$ serves in order to demonstrate the proof of concept. Thus we have a similarity measure:

$$M_1(t_1, t_2; L, B) \equiv \rho(\deg_{L,B}(t_1), \deg_{L,B}(t_2)), \tag{4.1}$$

where with $\rho(x, y)$ is by the definition the correlation between x and y:

$$\rho(x, y) \equiv \frac{\langle xy \rangle_k - \langle x \rangle_k \langle y \rangle_k}{\sqrt{(\langle x^2 \rangle_k - \langle x \rangle_k^2)(\langle y^2 \rangle_k - \langle y \rangle_k^2)}}.$$

4.1.3 Experimental results

We have explored the following sets of data:

1) The Universal Human Rights Declaration (UD) [4].

2) The first two chapters of four translations of "Don Quixote", by Miguel de Cervantes (DQ) [48, 49, 47, 46].

3) The Chapters 11 to 20 of the Bible book Exodus [2].

4) Mitochondrial DNA data (mtDNA).

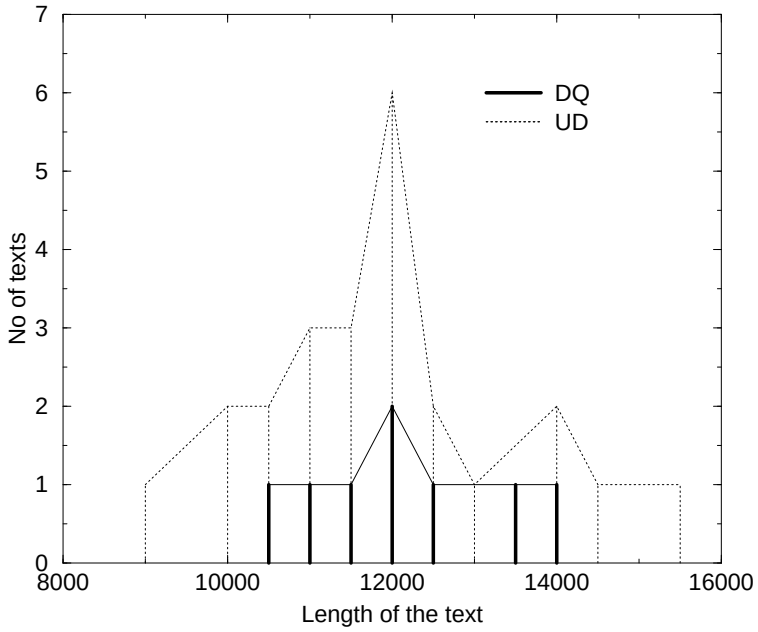

Figure 4.1: The histogram of the lengths of the texts used. The three texts that are double-byte coded UD are excluded from this graph. They have length from 20 to 21 kB. However, this does not affect the matching. It is clear that DQ and UD have similar lengths.

All these datasets have a reasonable size of about 10 Kbytes per text (See Fig.4.1). With the first and the second set we are trying to see whether we can detect the similarity in the structure and differentiate it from the influence of the grammar and the formatting of the document.

In all cases of human-written text (UD and DQ) the parameter L was set to 5

letters, which seams to eliminate many speech particles that carry essentially grammatical and morphological information, such as particles that determine the gender, definitive and indefinite particles, etc.

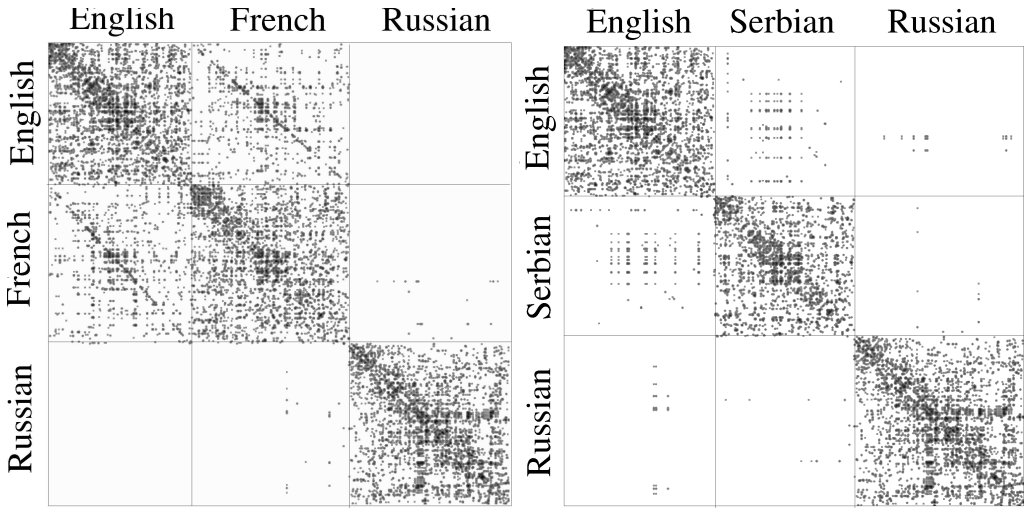

Figure 4.2: The connectivity of the graph G_L for two groups of three languages each, using UD as text. The left panel represents the concatenation of the English, French and Russian (KOI8-r coding) version of the UD, that gives significant cross compression between French and English. The right panel represents the connectivity matrix of English, Serbian (Latin) and Russian version. The cross-correlation between the English and the Serbian version is larger, but actually the Serb language is much closer to the Russian than to English.

Fig. 4.2 (left) represents a typical result of compression of the concatenation of the texts of the UD in three languages – English, French and Russian. The grey levels represent the presence of a link between the nodes that are placed in the axes of the graph. The graph is blurred, converting each point in a Gaussian bump with radius of about 40 text positions. The total length of the concatenated text is about 30K, so each third is represented by a square-like area. The upper left quadrant represents the result of compression of UD English, compressed with UD English, the quadrant at the center represents the UD of French, compressed with itself and the lower right quadrant represents the Russian version, compressed with itself. Two of the concatenated texts use one and the same alphabet and belong to similar language

groups. Therefore, we can observe significant cross-compression pattern in the middle quadrant of the first raw (and the second quadrant of the first column, which is the same). However, we can not see any cross-compression between the Russian version of the UD and the two other versions, due to the non-intersecting codes of the Cyrillic and Latin alphabets, with the exception of the formatting information (white spaces, capture enumeration and similar).

Using the NCD, we observe a small distance between the English and French versions, but large distances between Russian and both of them that seems a reasonable result.

However, on the right panel of the same figure, Fig. 4.2, we see the UD text compression of the concatenation of English, Serbian (Latin alphabet) and Russian. We see that the dissimilarity between the English and the Serbian version, using the NCD is smaller (0.9466) than the dissimilarity between the Serbian and the Russian version (0.9944), which can be observed also by the density of the middle quadrant of the first row. But in reality, the Russian and the Serbian belong to one and the same language group and ought to group closer.

The compression in either cases (English - French - Russian) and (English - Serbian - Russian) is dominated by the compression within the same text, e.g. the structure and the vocabulary of each language are predominant factors in the compression. The connectivity matrices of G_L have a typical block-diagonal structure. Therefore, we can try to compare the similarity of the texts, using the unidimensional measure M_1, Eq.(4.1). Fig. 4.3 (the two bottom panels) represents the smoothed degrees of each graph e.g. $\deg_{L,B}(\mathrm{UD}_{\mathrm{French}})$ and $\deg_{L,B}(\mathrm{UD}_{\mathrm{Russian}})$. The correlation coefficient is rather large, 52.3%. In all cases of the human rights declaration, written in different languages, we can observe similar values (with the exception of very similar languages).

The results for several translations of the UD are represented in Table 4.1 (right-down quadrant). The correlations vary from 51.6% to 66.8%.

To check that what is compared is not just the dictionary, we shuffle by random permutation the words of 4 texts of UD and measure the correlation. Fig. 4.4 shows the result. The shuffled versions are grouped near a zero correlation (actually it is

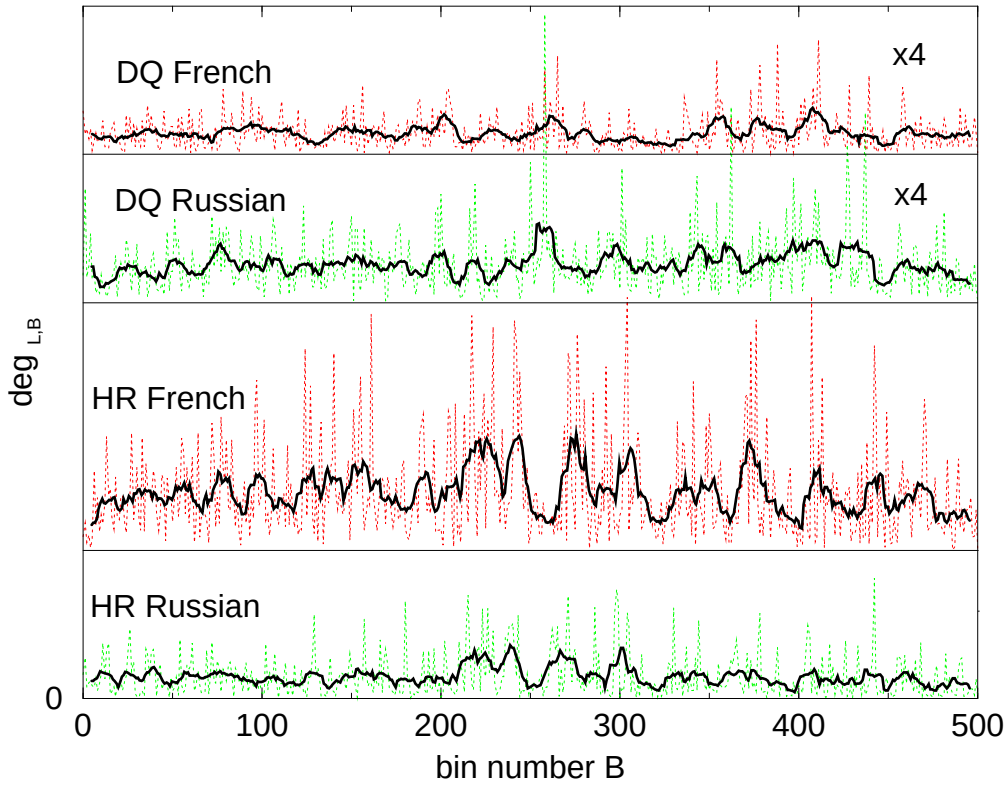

Figure 4.3: The degrees of the binned nodes of the graphs G_L of UD.

not exactly zero, because the distributions of the words is highly non-uniform), but the difference between the distributions is clear.

In order to see that what is captured is the structure of the text, and not the particular language coding, we can contrast the results of UD with another text. We choose as a different text the translations of the first chapter of DQ and we calculate the same similarity measure M_1 for English, French, Russian and Spanish versions of DQ and UD. The results are represented in Table 4.1 and as an histogram — in Fig. 4.5.

All pairings between the same text and different languages has $M_1 > 0.17$. In contrast, all pairings between different texts, including different texts in the same language, give $M_1 \in [-0.2, 0.06]$, which corresponds to random match. The smoothed node-degrees of the French and the Russian version of DQ are represented in the two

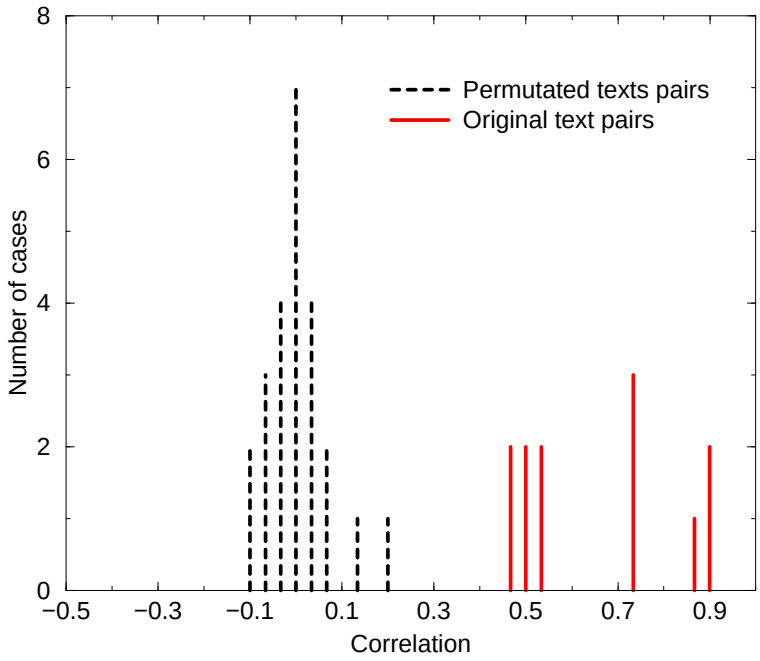

Figure 4.4: The correlations between randomly permuted texts (both between the permuted texts and the permuted texts and the original ones) and the distribution of the distances between the original texts.

upper panels of Fig.4.3. There exists a clear distinction between DQ and UD texts and a similarity between the Russian and the French version of one and the same text (either DQ or UD).

Because the UD and DQ texts are very different in style, one can argue that what is actually captured is the specific dictionary of each text and not the structure of the text itself. In order to discard such a possibility, we also compare two chapters of DQ with similar lengths. The results are represented in Table 4.2, where it can be clearly seen that the distinction is between different texts and not between different writing styles, dictionaries or authors.

Furthermore, in order to get more statistics we use several chapters of the Bible [2]. However, caution must be taken with old versions of the Bible because these are usually literal translations from other literal translations and sometimes the result is quite different from the original. For example, the ancient Reina-Valera Spanish

		DQ				UD					
		eng	fra	rus	spa	eng	fra	rus	spa	sre	sry
DQ	eng	100	31	18	34	5	2	-10	4	-6	-3
	fra	31	100	24	51	-1	-6	-10	-17	-9	-5
	rus	18	25	100	17	-16	4	-6	5	6	-2
	spa	34	51	17	100	6	-4	-5	-13	-10	-9
UD	eng	5	-1	-16	6	100	58	52	34	56	56
	fra	2	-6	4	-4	58	100	52	56	64	50
	rus	-10	-10	-6	-5	52	52	100	27	58	42
	spa	4	-17	5	-13	34	56	27	100	50	34
	sre	-6	-9	6	-10	56	64	58	50	100	67
	sry	-3	-5	-2	-9	56	50	42	34	67	100

Table 4.1: The similarity measure M_1 in percents, between the texts of the universal declaration of human rights and "Don Quixote" in different languages (eng – English, fra – French, rus – Russian, sre – Serbian/Latin, sry – Serbian/Cyrillic/UNICODE). The difference between the Serbian versions is due to the two or one byte coding of the Cyrillic alphabet in "sry".

version and the New International version differ significantly due to the very distinct translational history that have suffered, Hebrew-Latin-Greek-Spanish the first, and direct translation the last. Therefore, we tried, up to our knowledge, to use only direct translations from Hebrew.

To evaluate the performance on this benchmark we choose 31 translations of the Exodus, concatenating the chapters to form 4 corpora of reasonable magnitude, namely (A) Chap. 11 and 12; (B) Chap. 13,14 and 20; (C) Chap. 15, 16 and 17; and finally (D) Chap. 18 and 19. All these sets are more or less 10000 characters in length. We measure, on one side, the similarity between different chapter sets and on the other side, similarities between one and the same chapter set, excluding the trivial comparisons of identical text (same languages and same chapters set). As in the previous examples, it is expected that languages do not play an important role. The results are presented in Fig. 4.6, left. The figure shows that, in this corpus, we can judge whether the texts are translations with probability of about 95% and 5% error. The probabilities of errors of type I and II are presented in Fig.4.6, right.[3]

[3]We have noticed that relatively small correlation between one and the same text is observed

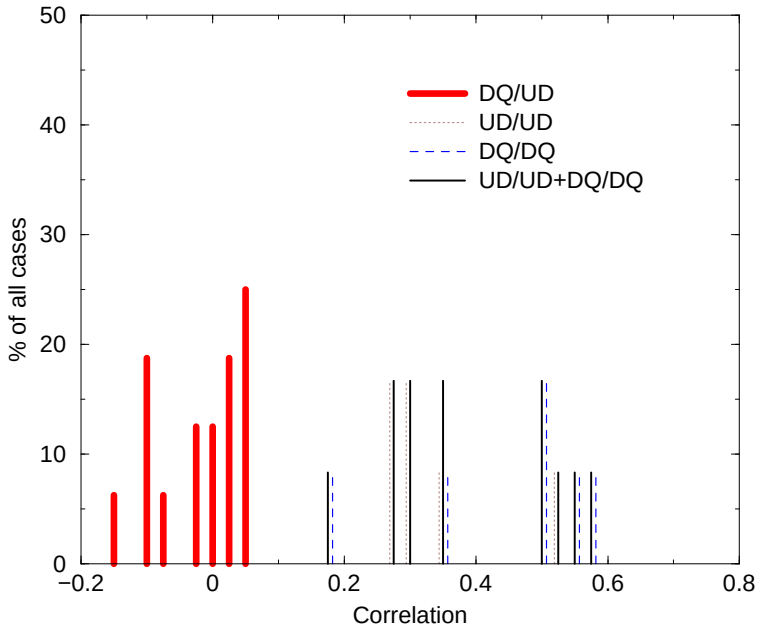

Figure 4.5: The histogram representation of the correlation coefficients of the similarity distance M_1 of UD and DQ.

As a conclusion we can see that the structure of the text is captured by the similarity measure M_1, Eq. (4.1) and the measure can effectively detect human-like text translations.

On the other side, if we apply the same procedure, used for UD, to mtDNA set, we can see that the similarities in a one-dimensional comparison are purely random. Compressing the concatenation of mtDNA of "Balaenoptera musculus" (BM) [21], "Balaenoptera physalus" (BF) [20] and "Homo sapiens" (HS) [37] and representing in the same manner the connectivity graph G_L, as for the UD (Fig. 4.2), we can see that the situation depicted now (Fig. 4.7) differs radically from that of the human generated text. Namely, the compression of the concatenation is achieved exclusively by the cross-similarities between the genetic material of different species. Internal similarities whiting a single mtDNA string are negligible in our experiments. This

if different translational histories of the Bible were selected, even if we compare one and the same language. We suspect that the overlaps below 20% are probably due to that reason, but because this reflects the particular history of that text we are not trying to extend the discussion in this article.

		C1				C2			
		eng	fra	rus	spa	eng	fra	rus	spa
C1	eng	100	14	10	13	-5	-5	-1	-3
	fra	14	100	16	34	-4	4	2	-4
	rus	10	16	100	10	-13	0	-2	3
	spa	13	34	10	100	-4	-1	-2	-2
C2	eng	-5	-4	-13	-4	100	24	15	24
	fra	-5	4	0	-1	24	100	25	32
	rus	-1	2	-2	-2	15	25	100	20
	spa	-3	-4	3	-2	24	32	20	100

Table 4.2: The similarity measure M_1 in percents, between two chapters of DQ. The notation of the languages is the same as in the previous table.

can be observed by the fact that the density of points in the block-diagonal is very small, while the non-diagonal blocks display the typical diagonal structure.

As it is natural, the species BM and BF are very similar between them and very dissimilar with respect to HS.

On the smoothed version one can observe self-similarities, concentrated in the diagonal, in the genetic material of BM. In order to see in more detail the behaviour of the connectivity of the graph in these points, we represent zoomed version of the diagonals of the connectivity (Fig.4.8), once within BM (left) and between the two whales – BF and BM (right). We can see that the self-similarities within BM are very short-range, which are absent in BF and HS species. The zoomed cross diagonal, on the other hand, shows very solid structure corresponding to the randomness of the process of evolution.

In Fig.4.8 we can see that the cross-diagonals are predominant in the compression of the concatenation of the genetic material, due to the nature of the evolution process and to the fact that mtDNA-s practically have no structure but only common predecessor.

Up to this moment we have ignored the non-diagonal blocks because no literal information was shared, which compelled us to focus on the structure of each text. Using genetic material, we can not extract information from the block-diagonal elements, as explained above, but we do have information in the non-diagonal blocks,

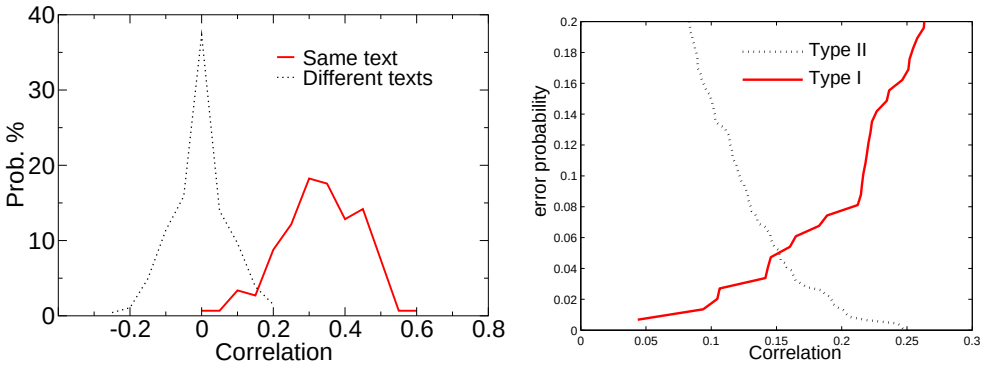

Figure 4.6: Left – the correlation between identical set of Bible chapters and different ones. Right – the probabilty of errors of type I and II in the estimation of the detection, based on Bible chapters detection. Note that the translations are actually very heterogeneous.

reflecting literal similarities between the elements. Thus, we can try to reconstruct a phylogenetic tree using as similarity measure some measure based on that blocks. In this thesis we use just the mass of these non-diagonal blocks, i.e., the distance between mtDNA t_1 and mtDNA t_2 is postulated to be $1/|G_L(t_1 \circ t_2)|$.

Once the distance matrix is obtained, we feed it into a novel quartet-based heuristic for clustering trees proposed in [34] and which implementation is available at http://www.complearn.org. The result is shown in Fig.4.9 for ten species and the obtained tree is equivalent to the one obtained with the NCD, being considered correct in the taxonomy literature.

Although we can not state that the results using just the density of the non-diagonal elements introduce some novel method, the consideration of the mtDNA using the same construct, G_L, as for human texts, serves as a proof of concept of the versatility of the approach: on one hand the information contained in the diagonal blocks can be used to compare objects in the scenario of non-literal information shared, while on the other hand, the information contained in the non-diagonal blocks can be used to measure the literal information shared between the objects.

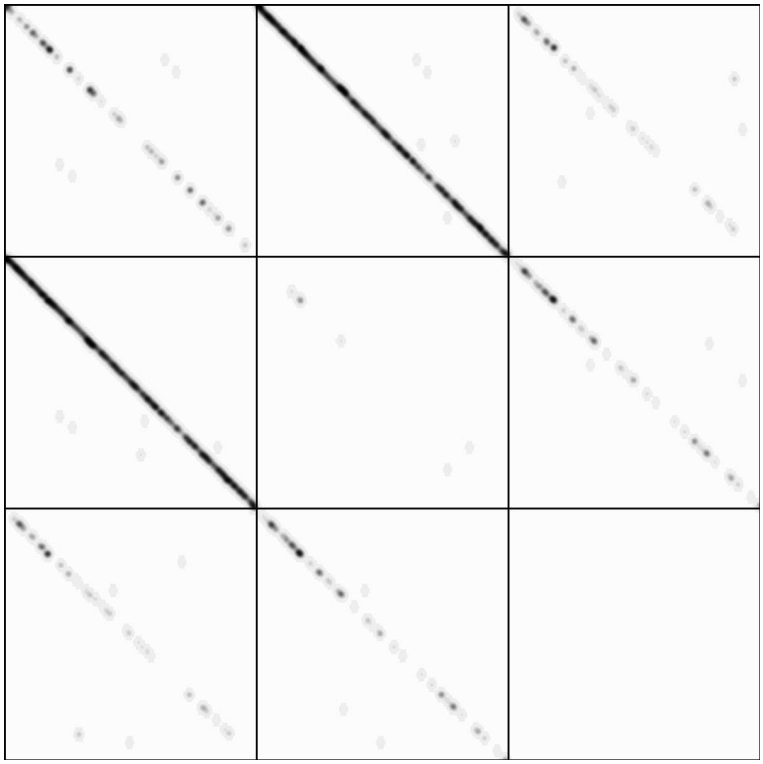

Figure 4.7: The mtDNA connectivity matrix of three species (two whales and a human) with $L = 12$. The dots are smoothed with a Gaussian smoothing of radius 40. The first row/column corresponds to "Balaenoptera musculus", the second to "Balaenoptera physalus" and the last to "Homo sapiens". We can see that cross-compression (information shared between two different strings) prevails over self-compression (rendundancy in a single string).

4.1.4 Phenomenological model of human written text similarities

In this section we give a phenomenological model of one dimensional comparison of the structure of texts $deg_{L,B}$.

Let us assume that one and the same concept, expressed is some text, is translated to two different languages. We can regard one of the texts t_1 as original and the other t_2, as a translated version.

If the grammar is exactly the same, then we simply must change the words of the

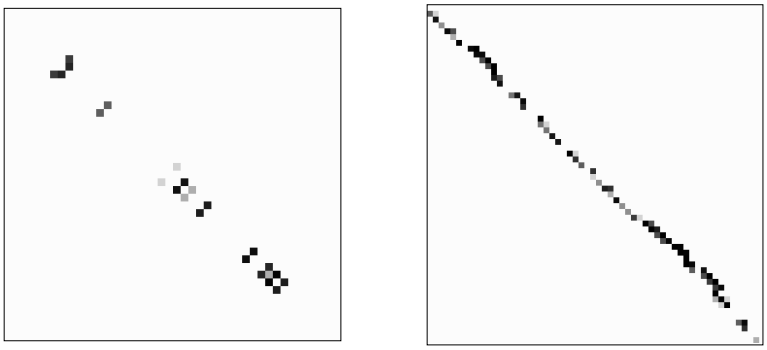

Figure 4.8: Detail of the diagonal of Fig. 4.7 (not smoothed) in the first quadrant left and in the quadrant to its right.

text in order to "translate" it. This morphological level of translation, in practice, never occurs[4], but it is a good starting point to analyze the problem. In this level of translation we can observe a weak change in the positions of the string that are compressed, due to the different word lengths in the vocabularies of t_1 and t_2. We can also observe accidental edges of G_L deleted or added due to causal coincidences in the modified or the original text. However, the word order of the text will be preserved. Therefore, choosing for each text a different L, proportional to the length of the text, $L_i \propto |t_i|$ and looking at the structure of the graph $G_{L_i}(t_i)$, we can expect to observe only fluctuations of the degrees of each group of nodes in $\deg_{L,B}(t)$.

As a first approximation, we can introduce an elementary operation on $\deg_{L,B}(t)$ called $H_M(\deg_{L,B}(t); \sigma_M)$, equivalent to adding Gaussian noise with zero mean and variance σ_M^2 to the degree of each node.

If the grammar is changed, then one must make the translation usually sentence by sentence, analyzing the grammatical structure of the text, building a parser tree, adding additional arcs in order to recover the semantic structure and finally, building another tree out of this graph.

Let us imagine that we can change the grammar, without changing the morphology. Then the only operations that would be performed on the text are: exchange of the order of the morphemes, repeat of a morpheme and delete of a morpheme.

[4]With one notable exception — the change of alphabet coding

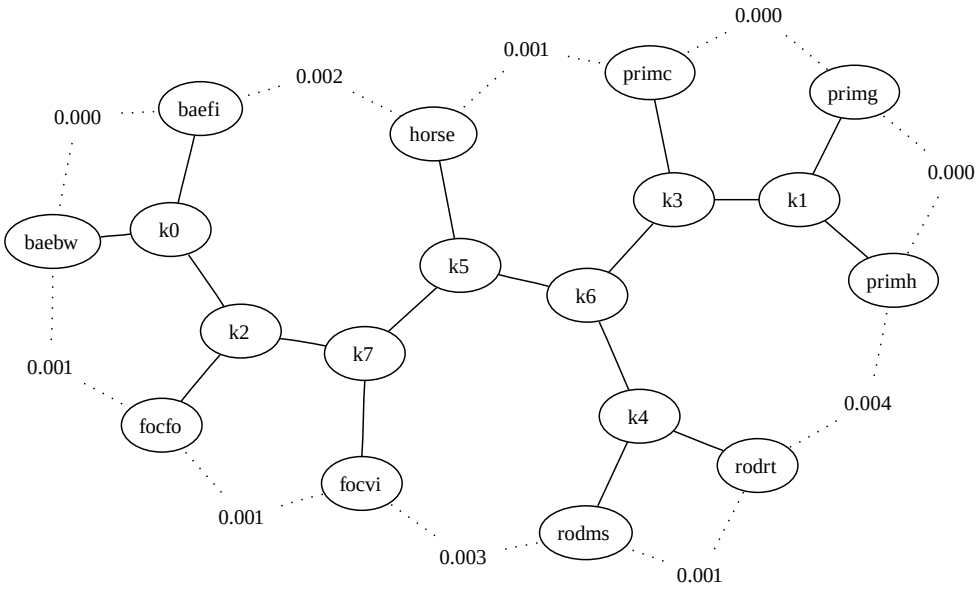

Figure 4.9: Phylogenetic tree for ten species: Balaenoptera musculus (baebw) [21], Balaenoptera physalus (baefi) [20], Phocoena phocoena (focfo) [19] Phoca vitulina (focvi) [121], Equus caballus (horse) [121] , Pan troglodytes (primc) [84] , Gorilla gorilla (primg) [65], Homo sapiens (primh) [37], Mus musculus (rodms) [9], Rattus norvegicus (rodrt) [68].

Because the number of the words is more or less the same in almost any language, we can rise the hypothesis that changing the order of the words by far prevails on the deletion or the duplication of words, with the exception of the particles that carry the grammatical information, which are usually ignored, choosing L properly.

Therefore, in a rather simplified phenomenological model we can assume that the operation of changing the grammar is the operation of word permutation. We can emulate this by introducing an exchange operation on $\deg_{L,B}(t)$ consisting of incrementing the degree of one node and decrementing the degree of another node. We can suppose that the exchanging nodes are near-by nodes, because usually the translation can be done sentence by sentence. The exchange is equal to the difference in the

frequencies of the words exchanged in each node. Let us introduce an elementary exchange operation that supposes that the two nodes are exchanging words of difference of referential frequency of the words δ with a probability that in first approximation can be supposed to have Gaussian distribution $G(0, \sigma_G)$ and introduce this operation as $H_G(\deg_{L,B}(t); \sigma_G, \delta)$.

Combining the morphological and the parser-like translator, and once again assuming as an approximation that the grammatical and the morphological operations are independent, we can state the problem:

Calculate the correlation coefficient between the original text t_1 and the translated text t_2, on which the operations $H_G(.)$ and $H_M(.)$ are performed.

Because the operations do not need the graph itself, but only the degrees of each node, we can safely perform the calculations using only the degrees of the nodes of the graph. In other words, we ought to calculate the quantity:

$$\rho(H_M(H_G(\deg(t_1); \sigma_G, \delta); \sigma_M), \deg(t_1)).$$

In order to calculate it we assume that the distribution d_k of $\deg(t_1; k)$ has variance σ_1, where we abuse slightly of the notation by omitting the indices L_i, B. There is no requirement to have some particular distribution; only the existence of the second momentum is required, which is guaranteed for any stochastic source (the length of the compressed text becomes proportional to the length of the original).

After some simple algebra, consisting of calculation of the quantities:

$$\langle R(d_a)R(d_b)\rangle_k, \quad R \in \{H_G, H_M, E\},\ a, b \in \{1, 2\},$$

where E means identity and d_x is an abbreviation of $\deg(t_x)$, the result is:

$$\rho(H_M(H_G(\deg(t_1); \sigma_G, \delta); \sigma_M), \deg(t_1)) = \left[1 + (2\delta^2\sigma_G^2 + \sigma_M^2)/{\sigma_1}^2\right]^{-1/2}. \qquad (4.2)$$

Details of the calculations are given in appendix A. We can observe that, actually, all the factors mentioned above enter as a single factor, $(2\delta^2\sigma_G^2 + \sigma_M^2)$, which measures the divergence of the texts.

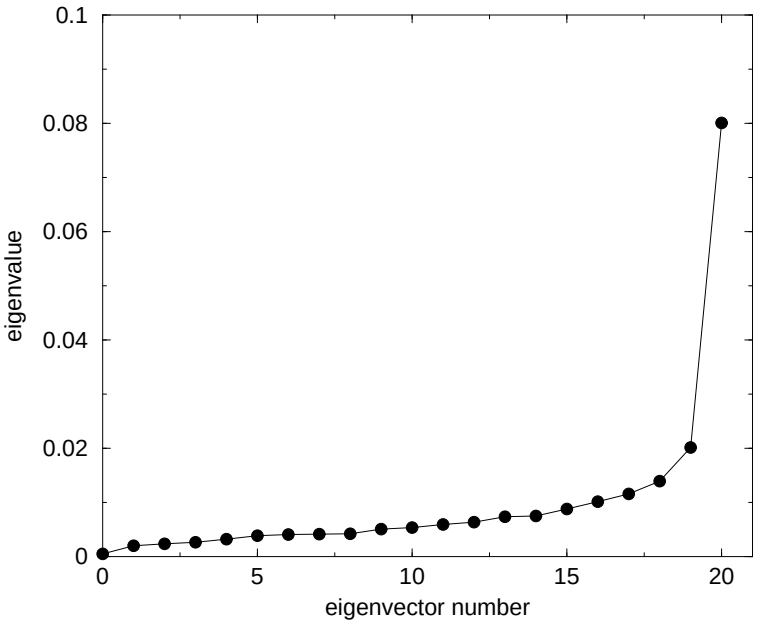

Figure 4.10: The eigenvalues of the correlation matrix. It is clear that there is one predominant factor (with eigenvalue 0.08) and the next factor is 4 times weaker. The diagonal elements of the correlation matrix were zeroed in the intermediate calculus in order to increase the precision.

It is easy to check on the experimental data of the UD, using linear factor analysis, that only one factor predominates in the correlation. Checking the eigenvectors of the resulting covariance matrix, Fig.4.10, one can see that the first eigenvector (the rightmost one), clearly predominates, which shows that even such a simple phenomenological model captures the most important characteristics of the translation.

4.1.5 Discussion

As a conclusion, the experimental data confirm the hypothesis that by using LZ inspired structures, we can detect similarities in the texts even if the alphabet is different and we can also detect the type of the similarities, namely if the similarities are from common concepts with well-defined structure or from common data predecessor with poorly expressed or missing structure.

The measure M_1 works reliably in texts larger than 10 KB. However, when using

short texts, of length inferior to 5 KB we can not achieve good results. It would be interesting to find a representation of short texts that can give a reliable similarity measure. As a first attempt the measure is good enough. Further consideration shows that measures based on dynamical programming are promising.

The comparison of correlations in order to judge the similarity between texts as the only similarity measure is of course, prone to errors. First of all we must choose the parameters L and B in a way that allows us to have sufficient, but not too many arcs of the graphs within one bin. In practice, we are looking for some 5-15 arcs in 500 bins. But this implies that having some reasonable L for texts (about the length of one word, e.g. 4-8 symbols), we need significant length of the text. Actually this is observed also empirically. The method works well with texts of length 5-15 KB.

To avoid this limited text length range, one can use much more sophisticated methods, based in general on statistical physics conformation analysis, that is beyond the scope of this thesis and subject of an ongoing work.

The challenging question is whether the grammar can also be captured using similar methods. The grammar is an element may be in use when a text is compressed. As an example, let us consider some arbitrary Spanish phrase, for example “las chicas altas son buenas bailarinas” (the tall girls are good dancers). The gender/plural information is carried by the ending of the word “-as” and the string “as” should be coded according to this.

4.2 Source Code Plagiarism Detection

The development of the World Wide Web and the increasing standardization of electronic documents has lead to a greater incidence of plagiarism in many aspects of life. According to a recent article in Nature [70], incidence of plagiarism has also reached the scientific community. However, it is much more widespread in the case of undergraduate students, where educators are generally ill-equipped to face the technological challenges posed by plagiarism detection. Realizing this, different national educational authorities (for instance, those of the United Kingdom [45, 27]) have begun funding projects dedicated to study the impact and growth of plagiarism, and to

propose adequate measures.

Exact figures are unknown, since successful plagiarism is by definition not detected, but are believed to be high and growing [36, 90]. Alex Aitken, one of the leading experts in operating plagiarism detection software, asserted in a personal communication prior to 2001 that for any (USA) student corpus, 10% of submissions are plagiarized [45, p. 4].

Two major types of documents are being targeted by undergraduate plagiarism: essays and computer assignments, although plagiarism cases in art degrees have also been reported [153, p. 4]. This thesis section focuses on source code plagiarism.

Plagiarism detection in programming courses is tedious and extremely time consuming for graders. Additionally, it is emotionally and legally risky for student and educator alike [80]. University experience also shows that even minor plagiarism levels can cause a mistrust for the work of students which can lead to baroque examinations to prove the authenticity of each student's work, or to the relative weight of possibly-plagiarized practical assignments in the final grade being far lower than the actual share of effort they truly required from the student. Ignoring the problem posed by plagiarism results in unfair grading, and can have an avalanche effect in plagiarism incidence levels.

There are two facets to the the prevention of plagiarism in computer programs. The first is of ethical and normative nature, and involves fighting the deeper causes of plagiarism, selecting appropriate academic and legal deterrents, and related issues. This facet is examined, for example, in [26] and [115]. The second facet is directly related with software engineering, and addresses the technical measures required to detect plagiarism within a set of assignments.

The present work is focused on this second facet, and seeks to assist educators in the task of plagiarism detection by following the approach of [93]: monitoring excesses of collaboration which can signal anomalous behaviors in students and/or lecturers.

Among the most usual causes for plagiarism, we may find the following:

- Low ethical and/or technical preparation of the students.
- Ambiguity in assignments: poorly understood exercises are more likely to suffer

plagiarism.

- Low clarity in the university's guidelines on student collaboration.
- Bad course planning causing an excessive work load.

In this light, a plagiarism detection tool can be considered as a sanity check, to be used in the diagnosis of the relative health of the teaching environment. If this check is to be objective, we feel that plagiarism detection tools should be updated and augmented with modern technologies.

The main contribution of this research is the design and development of AC, a novel plagiarism detection tool available at

http://tangow.ii.uam.es/ac

Notable features of AC include the following:

- Intuitive display of results, providing visualizations such as graphs and different types of histograms, allows visual exploration of analysis results . From the visualizations, plagiarism suspects can be selected in order to perform detailed manual analysis (see as an example figs. 4.11, 4.12, 4.13 and 4.15).
- Extraction/filtering utility designed to ease the initial task of preparing assignment submissions for analysis. Submission preprocessing would otherwise be a repetitive and error-prone task; this utility seeks to automate the process without being specific to any particular institution (fig. 4.14).
- Altough AC's design was initially planed to improve the NCD plagiarism detection power, it currently implements a novel distance integration architecture, allowing other similarity distance algorithms to be combined, compared and refined. Many distance algorithms found in the scientific literature on plagiarism detection are included, together with several others developed by our research group.
- It incorporates a novel threshold recommendation system based on statistical outlier detection. Distances lower than this threshold are suggested for manual

analysis, establishing an heuristic starting point to help grader's unveiling of plagiarism patterns (figs. 4.16 and 4.17).

- Open Source code, featuring a modular design that allows easy customization of both algorithms and visualizations. AC can therefore be analyzed, extended and adapted for any particular requirements.

- Stand-alone platform-independent program which can be executed in any computer with a JAVA runtime environment. There is no need to send the submissions to a server in a different institution, avoiding privacy concerns. Furthermore, a desktop application can deliver interactive visualizations which would be difficult to perform online.

- Has been tested by using special plagiarism detection benchmarks. These benchmarks are computer generated, and make it possible to deploy aritificial submission corpora in which the assignment nature, the plagiarism pattern, and the corpus size is set by the tester (see appendix B).

- Intended to be a long-term supported tool by means of facilities such as version source-control system, bug-tracking record, forums and an updated website. Transition to Sourceforge or other Open Source collaborative software management portals is currently under study.

4.2.1 Discussion

The problem of plagiarism detection is a difficult one. The frontier between, on one side, random similarity or simple inspiration from anothers' work and, on the other side, blind cut+paste plagiarism is not clear-cut, and certain cases will always require a human grader to distinguish between what is acceptable and what is not. However, different algorithms and heuristics can be used to identify suspects of blatant plagiarism and flag the more complex cases, greatly simplifying the grader's task.

This section has presented AC, a plagiarism detection tool which also doubles as a framework for research into source code plagiarism detection. Even though AC was

initially designed as an enviroment to increase the NCD applicability to plagiarism detection, it has suffered a great development, currently offering many improvements over other tools described in current literature: the use of rich visualization greatly simplifies the task of analyzing the result of similarity tests; its stand-alone, cross-platform implementation does not raise privacy concerns found in web-based systems; and preparation of assignment submissions for automated plagiarism detection, overlooked by many systems, can be automated with a graphical user interface.

AC is released as open source software, and full implementation details are available to any interested parties. Lack of comparable information for other plagiarism detection tools makes their results difficult to replicate or improve. Furthermore, the availability of AC under an open source license guarantees that other researchers can follow the project and participate in its development. Due to the modular design of the program, it is easy to integrate new similarity distance algorithms, allowing graders or researchers to compare or complement their performance. Many of these algorithms have been already incorporated into AC, including two novel approaches. Finally, AC includes the first application of outlier statistical methods to plagiarism detection, proving fast initial identification of suspicious assignments. The analysis of related experiments has revealed interesting aspects of the patterns found in typical assignment corpora.

For an in-depth technical presentation of all AC's implemented features, please consult reference [66].

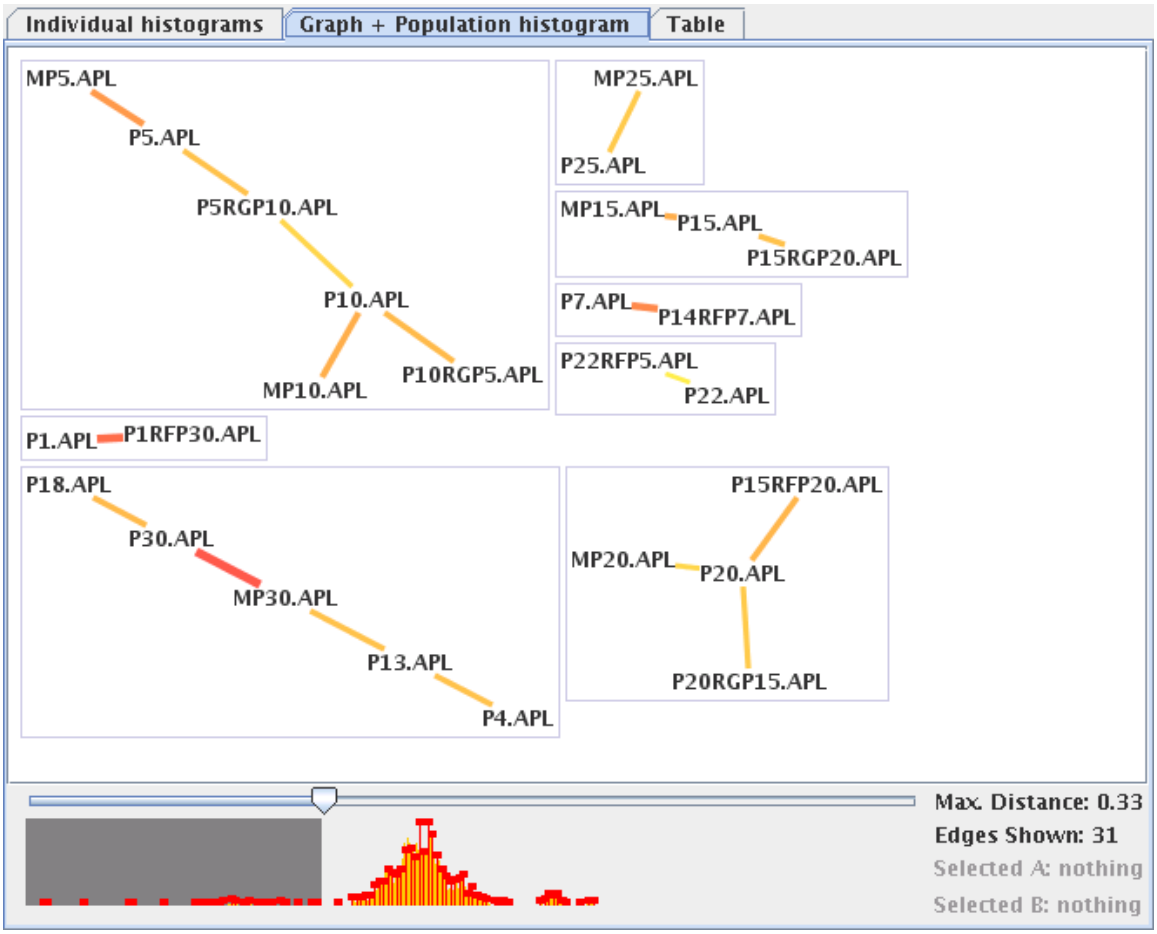

Figure 4.11: Test results visualized as a graph, with a histogram reflecting the frequency of each distance (ranging from 0, most similar, to 1) and a horizontal slider, used to select the maximum distance that is used for inclusion in the graph: only pairs of assignments with a distance lower than this threshold are included.

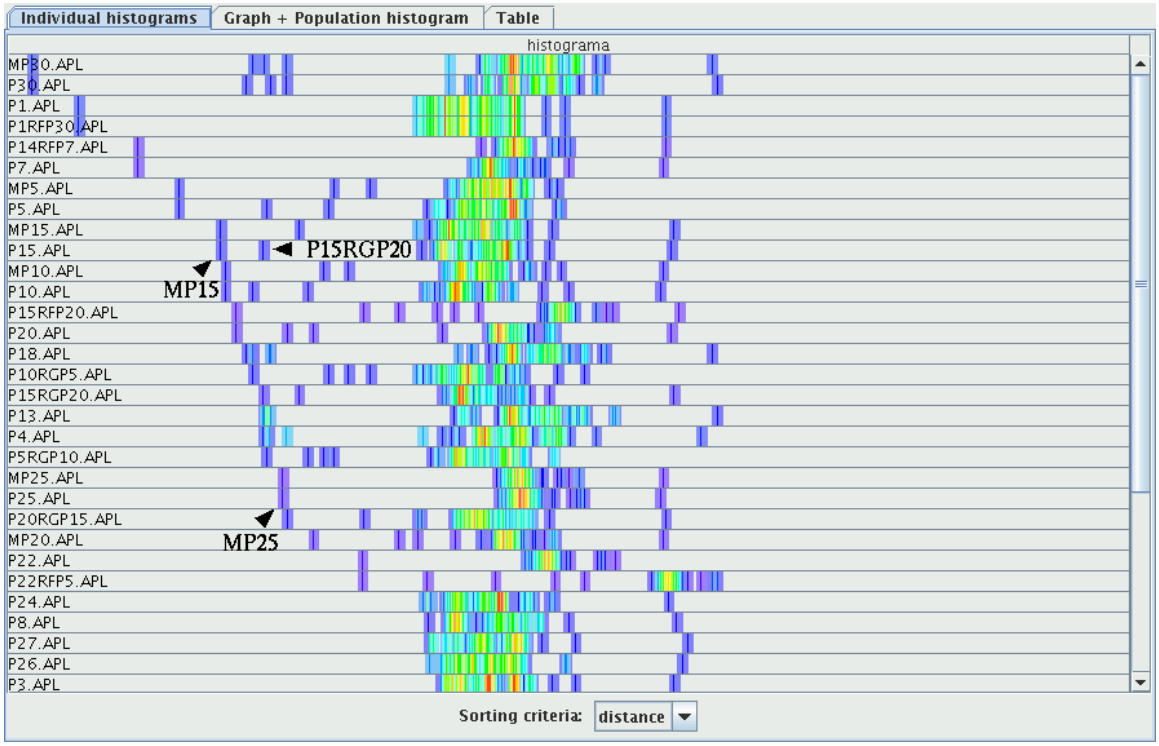

Figure 4.12: Test results visualized as individual histograms. Each row represents a color-coded histogram (blue is low, red is high) of the frequency with which other assignments have presented a given similarity to this one. Unexpected gaps in the leftmost side of the histogram suggest existence of plagiarism

Individual histograms | Graph + Population histogram | Table

Distance	One	The other
0.021857923	P30.APL	MP30.APL
0.062992126	P1RFP30.APL	P1.APL
0.11627907	P7.APL	P14RFP7.APL
0.15272728	P5.APL	MP5.APL
0.18978103	P15.APL	MP15.APL
0.1941392	P10.APL	MP10.APL
0.2037037	P20.APL	P15RFP20.APL
0.21311475	P30.APL	P18.APL
0.21851853	P10RGP5.APL	P10.APL
0.21857923	P18.APL	MP30.APL
0.227758	P15RGP20.APL	P15.APL
0.22797927	P4.APL	P13.APL
0.2287234	P13.APL	MP30.APL
0.23076923	P5RGP10.APL	P5.APL
0.23316061	P4.APL	P18.APL
0.23404256	P18.APL	P13.APL
0.23404256	P30.APL	P13.APL
0.24522293	P25.APL	MP25.APL
0.24870466	P4.APL	MP30.APL
0.24870466	P4.APL	P30.APL
0.24915825	P20RGP15.APL	P20.APL
0.2597865	P15RGP20.APL	MP15.APL

Figure 4.13: Test results visualized as a distance table.

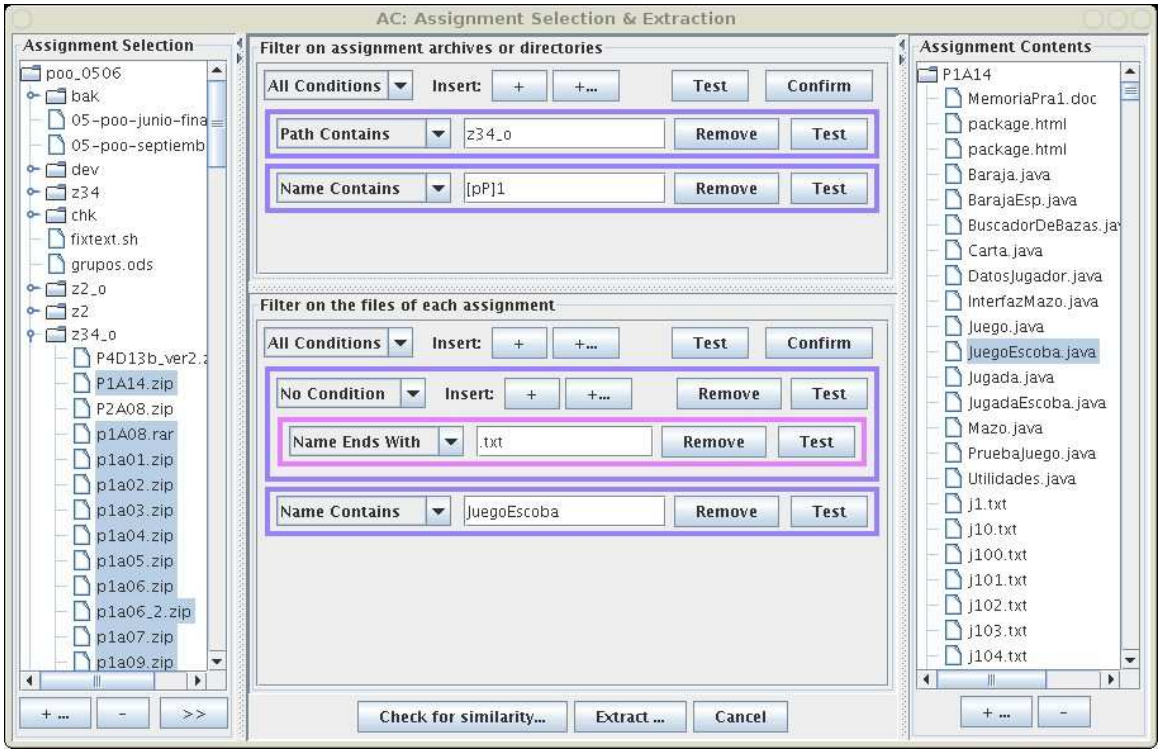

Figure 4.14: Screenshot of the filtering interface

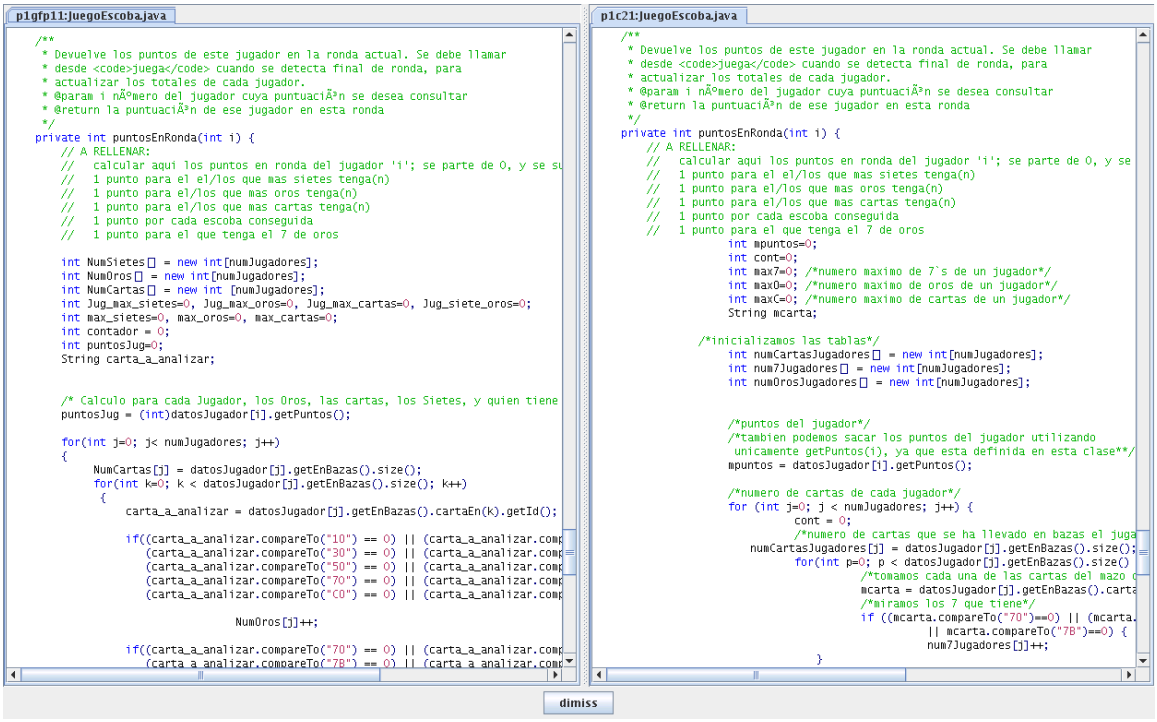

Figure 4.15: Visual comparison of two assignments

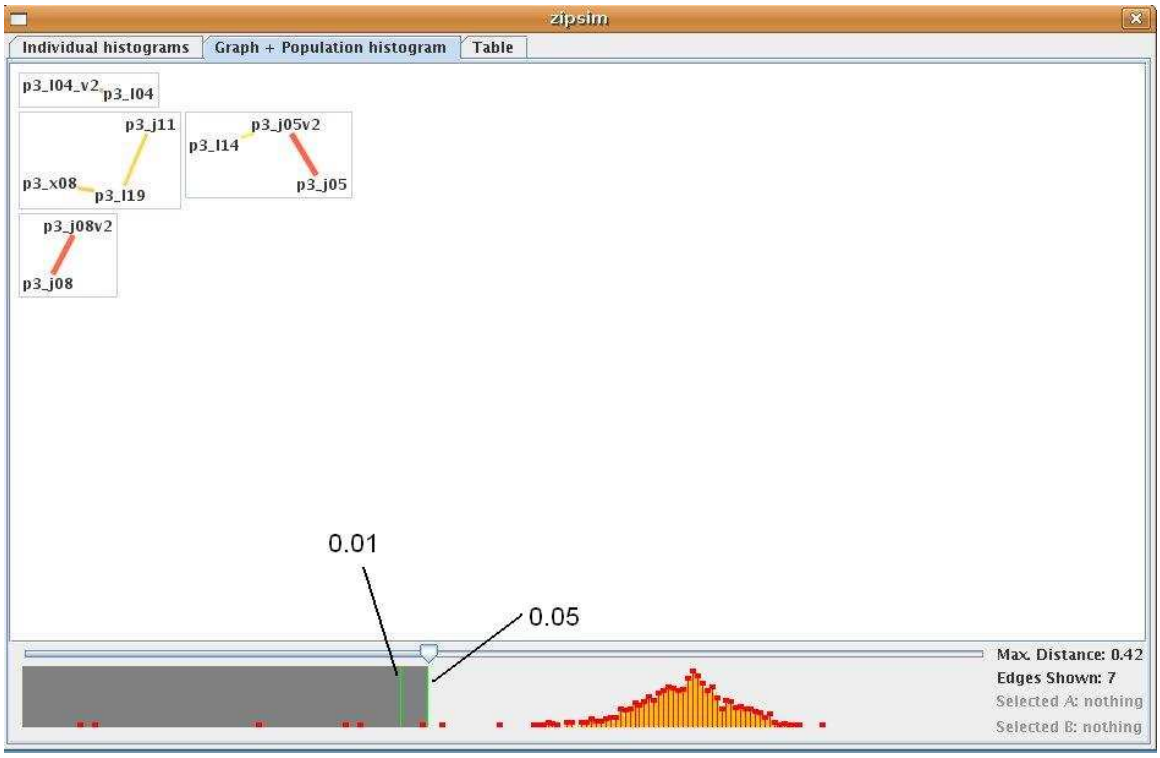

Figure 4.16: AC gives two threshold recommendations for plagiarism (outlier) detection in the Graph+Histogram visualization. The probability of a non-plagiarized pairwise distance falling below the threshold is annotated.

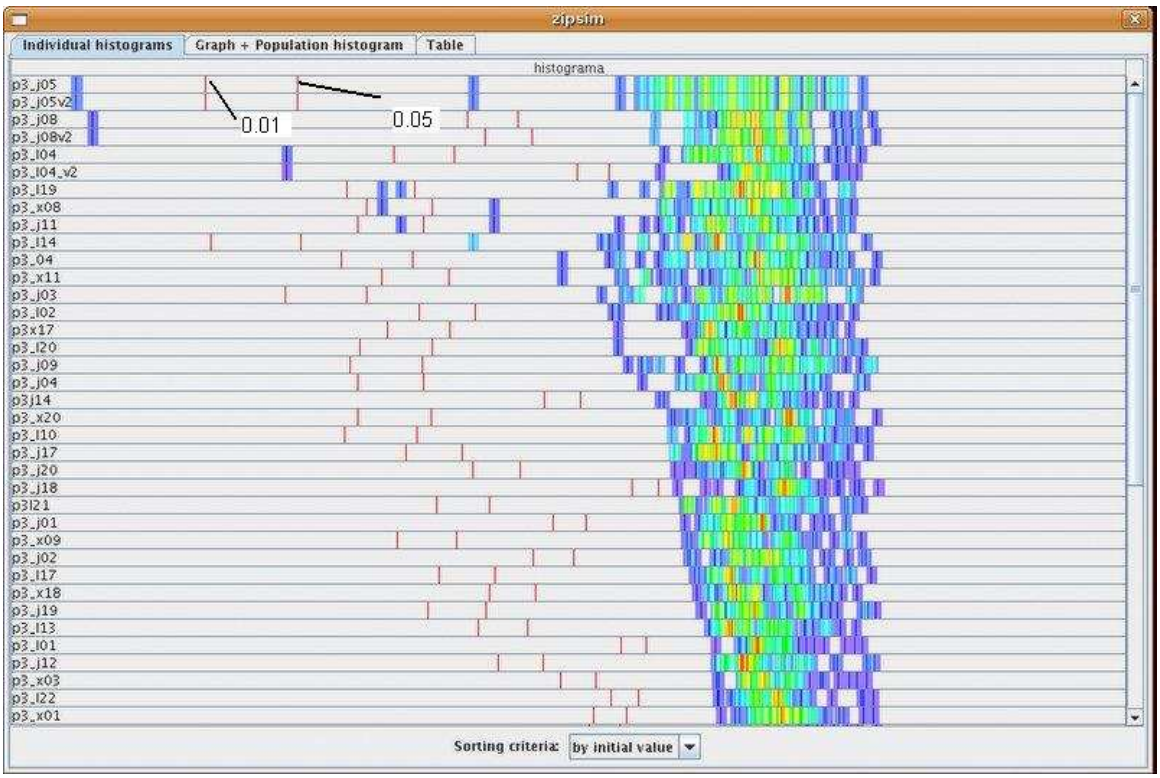

Figure 4.17: AC gives two threshold recommendations for plagiarism (outlier) detection for each Individual Histogram visualization.

4.3 Music Generation

The automatic generation of musical compositions is a long standing, multi disciplinary area of interest and research in computer science, with over thirty years history at its back.

Some of the current approaches try to simulate how the musicians play [116] or improvise [24], while others do not deal with the time spent in the process. Many of them apply models and procedures of theoretical computer science (cellular automata [25], parallel derivation grammars [116], or evolutionary programming [106, 100, 85, 92]) to the generation of complex compositions. The models are then assigned a musical meaning. In some cases, the music may be automatically found (composed) by means of genetic programming.

In a previous paper [15] we proposed the use of the well-known Normalized Compression Distance, an algorithmic information measure [34], as a fitness function which may be used by genetic algorithms to automatically generate music in a given predefined style. The superiority of the relative pitch envelope over other musical parameters, such as the lengths of the notes, has been confirmed in [104], bringing us to develop a simplified algorithm that nevertheless obtains interesting results.

In this thesis we start on the results of the previous work and refine them, trying to increase the efficiency of the procedures described in the above mentioned paper. This is done by testing several variations of the recombination operator to fine tune the genetic algorithm for this application, as it has been observed that this operator plays an important role in this procedure.

This section is organized thus: the first subsection provides a short introduction to musical concepts needed to better understand the remainder, with a description of the restrictions applied in our experiments and an enumeration of different ways of representing music. The second subsection introduces the idea of using the Normalized Compression Distance for computing the distance of the results of the genetic algorithm from the target musical pieces. The third subsection describes the genetic algorithm we have used for music generation. In the fourth and fifth subsections we describe our experiments, where we have compared the use of one or two target

guides, and six different recombination procedures for the genetic algorithm. Finally, the last subsection presents a discussion of the results and several possibilities for future work.

4.3.1 Musical representation: restrictions

Melody, rhythm and harmony are considered the three fundamental elements in music. In the experiments performed in this thesis, we shall restrict ourselves to melody, leaving the management of rhythm and harmony as future objectives. In this way, we can forget about different instruments (parts and voices) and focus on monophonic music: a single performer executing, at most, a single note on a piano at a given point in time. Melody consists of a series of musical sounds (notes) or silences (rests) with different lengths and stresses, arranged in succession in a particular rhythmic pattern, to form a recognizable unit.

In Western music, the names of the notes belong to the set {A, B, C, D, E, F, G}. These letters represent musical pitches and correspond to the white keys on the piano. The black keys on the piano are considered as modifications of the white key notes, and are called sharp or flat notes. From left to right, the key that follows a white key is its sharp key, while the previous key is its flat key. To indicate a modification, a symbol is added to the white key name (as in A# or A+ to represent A sharp, or in B$\flat$ or B-, which represent B flat). The distance from a note to its flat or sharp notes is called a *half step* and is the smallest unit of pitch used in the piano, where every pair of two adjacent keys are separated by a half step, no matter their color. Two consecutive half steps are called a whole step. Instruments different from the piano may generate additional notes; in fact, flat and sharp notes may not coincide; also, in different musical traditions (such as Arab or Hindu music) additional notes exist. However, in these experiments, we shall restrict to the Western piano lay-up, thus simplifying the problem to just 88 different notes separated by half steps. An interval may be defined as the number of half steps between two notes.

Notes and rests have a length (a duration in time). There are seven different standard lengths (from 1, corresponding to a whole or round note, to 1/64), each

of which has duration double than the next. Their names are: whole, half, quarter, quaver, semi-quaver, quarter-quaver and half quarter-quaver. Intermediate durations can be obtained by means of dots or periods. The complete specification of notes and silences includes their lengths.

A piece of music can be represented in several different, but equivalent ways:

1. With the traditional Western bi-dimensional graphic notation on a pentagram.

2. By a set of character strings: notes are represented by letters (A-G), silence by a P, sharp and flat alterations by + and - signs, and the lengths of notes by a number (0 would represent a whole note, 1 a half note, and so on). Adding a period provides intermediate lengths. Additional codes define the tempo, the octave and the performance style (normal, legato or staccato). Polyphonic music is represented with sets of parallel strings.

3. By numbering (1 to 88) the pitches of the notes in the piano keyboard. Another number can represent the length of the note as a multiple of the minimum unit of time. A voice in a piece of music would be a series of integer pairs representing notes and lengths. Note 0 would represent a silence. Polyphonic music may be represented by means of parallel sets of integer pairs.

4. Other coding systems are used to keep and reproduce music in a computer or a recording system, with or without compression, such as wave sampling, MIDI, MP3, etc.

In our experiments, we represent melodies by the second and third notation systems.

4.3.2 The NCD as a fitness function

Li and Sleep have reported that this distance, together with a nearest neighbor or a cladistic classifier, outperforms some of the finest (more complex) algorithms for clustering music by genre [104]. Earlier research have also reported a great success in clustering tasks with the same distance [35]. These results suggest that the NCD

although not achieving the universality of its incomputable predecessor (the NID), works well to extract features shared between two musical pieces.

On the other hand, genetic algorithms need to define a fitness function to compare different individuals, subject to simulated evolution, and classify them according to their degree of adaptation to the environment. In many cases, fitness functions compute the distance from each individual to a desired goal.

Suppose we are to generate a composition that resembles a Mozart symphony; in this situation, we can elaborate a natural fitness measure: an individual (representing a composition) has a high fitness if it shares many features with as many as possible of the Mozart's symphonies. We propose to use a genetic algorithm (with musical compositions as individuals of the population) which uses the NCD as the fitness measure to compute these shared features between the individuals and the target musical guides which, in this example, would be the set of Mozart's symphonies.

It remains to choose the compressor used to estimate the NCD. Li and Sleep compute it by counting the number of blocks generated by executing the LZ78 compression algorithm ([155]) on an input . In our initial experiments, we used both the LZ78 and LZ77 algorithms, and found that LZ77 performs better, which agrees with theoretical results from Kosaraju and Manzini [98]; therefore LZ77 has been used as our reference compressor in all the experimental results presented in this thesis.

4.3.3 The genetic algorithm used to generate music

Our genetic algorithm generates music coded as pairs of integers, the third format described in subection 4.3.1, which is specially fitted for our purpose. This notation can then be transformed to a note string (the second representation) for reproduction.

We also decided to start with monophonic music, leaving harmony for a latter phase ad working with melodies.

Finally, we made the decision, in this first stage of experiments, to apply the genetic algorithm only to the relative pitches of the notes in the melody (i.e. we only consider the relative pitch envelope), ignoring the absolute pitches and the note lengths, because our own studies and others [104] suggest that a given piece of music

remains recognizable when the lengths of its notes are replaced by random lengths, while the opposite does not happen (the piece becomes completely unrecognizable if its notes are replaced by a random set, while maintaining their lengths).

The proposed genetic algorithm scheme is now described. It includes a previous pre-process step, made of the following parts:

- One or more (M) musical pieces of the same style/author are selected as targets or guides for music generation. We define each of the guides as g_i and the guide set as $G = \{g_i\}_{i=1}^{M}$.
- All the guides must be coded in the same way, as pairs of integers, as described above.
- The individuals in the population are coded in the same way as the guides.
- The fitness function for an individual x and a guide set G is defined as

$$f(x) = \left(\sum_{g_i \in G} \mathrm{NCD}(x, g_i) \right)^{-1} \tag{4.3}$$

  By maximizing $f(x)$ (minimizing the sum of the distances), we expect to maximize the number of features shared by the evolving individuals with the guide set.

The remaining steps of the genetic algorithm are:

1. The program generates an initial random population of 64 vectors of N pairs of integers, where N is the length of the first piece of music in the guide set. The first integer in each pair is in the [24,48] interval and represent the note interval. The second is in the [1,16] interval and represents its length as multiple of the minimum unit of time. Each vector represents a genotype.
2. The fitness of the genotypes is computed as in Eq. 4.3.
3. The genotypes are ordered by their increasing distance to the guide set, i.e., decreasing fitness..

4. If some perdetermined fitness has been reached, the genetic algorithm stops. The notes in the target genotype are paired with a function of the lengths of the guide piece(s).

5. The 16 genotypes with lowest fitness are removed. The 16 genotypes with highest fitness are paired randomly. Each pair generates a pair of children, a copy of the parents modified by four genetic operators. The children are added to the population to make again 64, and their fitness is computed as in step 2.

6. Go to step 3.

We need to say some words about our coding choice. The use of only two octaves for the notes (i.e. [24,48]) does not represent an important restriction (actually many real melodies comply with it), while it reduces significantly the size of the search space. The fact that absolute notes are later converted to intervals has the consequence that a piece of music becomes invariant under transposition, which is proper, because human ear recognizes transposed pieces as the same.

The second number, belonging to the [1,16] interval in each pair, represents the length of the note and is currently ignored (remember that these lengths are replaced by a function of the lengths of the guide pieces). In this way, however, the system is ready for the future objective of automatically generating the lengths.

The four genetic operators mentioned in the algorithm are:

- Recombination (applied to all generated genotypes). The genotypes of both parents are combined using different procedures to generate the genotypes of the progeny. Different recombination procedures have been tested in this set of experiments to find the best combination (see Sect. VI).

- Mutation (one mutation was applied to every generated genotype, although this rate may be modified in different experiments). It consists of replacing a random element of the vector by a random integer in the same interval.

- Fusion (applied to a certain percentage of the generated genotypes, which in our experiments was varied between 5 and 10). The genotype is replaced by

a catenation of itself with a piece randomly broken from either itself or its brothers genotype.

- Elision (applied to a certain percentage of the generated genotypes, in our experiments between 2 and 5). One integer in the vector (in a random position) is eliminated.

The last two operations, together with some recombination procedures, allow longer or shorter genotypes to be obtained from the original N element vectors.

4.3.4 Testing different number of guide pieces

In our first experiments, we selected the simplest recombination procedure (strategy 1 of Sect. VI) and tested the effect of varying the number of guide pieces and the functions which generate the lengths of the notes in the best output pieces.

First, we used as the guide a single piece of music, Yankee doodle, represented (using representation 2 of Sect. II) by the following string:

```
M2T2O3L2C+4C+4D+4F4C+4F4D+4O2G+4O3C+4C+4D+4F4C+3C4P4C+4C+4D+4F4F+4F4D+
4C+4C4O2G+4A+4O3C4C+3C+4P4O2A+4.O3C5.O2A+4G+4A+4O3C4C+4P4O2G+4.A+5.G+4
F+4F3G+4P4A+4.O3C5.O2A+4G+4A+4O3C4C+4O2A+4G+4O3C+4C4D+4C+3C+3
```

The WAV formated file of Yankee doodle ready for listening, Yankee.wav, as well as the rest of the music files referenced in this section can be found at

http://www.eps.uam.es/~mcebrian/music

In this case, the fitness function was straightforward: the objective was to minimize the NCD of the vectors of note intervals of the evolving individuals to the corresponding vector in the guide piece.

After applying the genetic algorithm, the succession of notes obtained was completed by adding length information in the following way: each note was assigned the length of the note in the same position in the guide piece (the guide piece was shortened or circularly extended, if needed, to make it the same length as the generated piece, which could be shorter or longer).

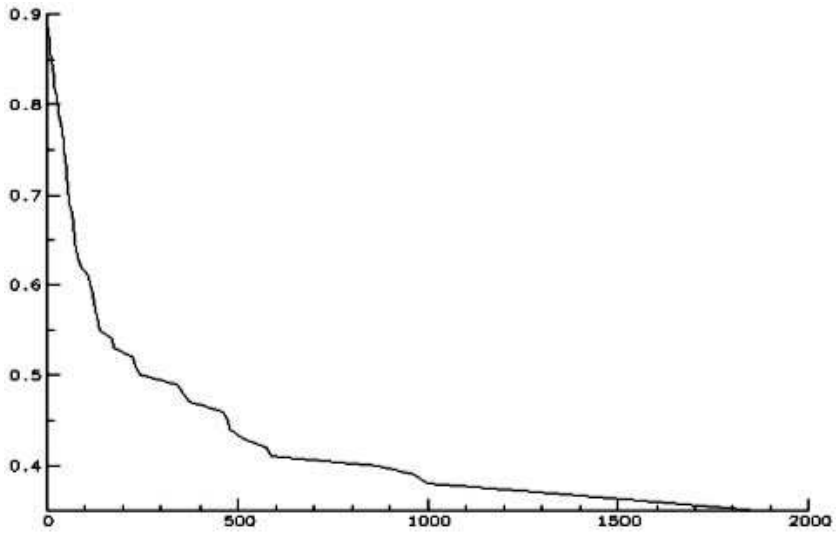

Figure 4.18: Number of generations needed to reach a given distance to the target.

In successive executions of the algorithm, we obtained different melodies at different distances from the guide. It was observed that a lower distance made the generated music more recognizable to the ear, as related to the guide piece For instance, the distance to the guide of the following generated piece, named Yankee_NEW.wav is 0.43:

```
T5O3D+2O2G+2O3C+2C+2D+2F2F+2F2E2C2D2E2O2F1D2E2D2C2D2E2F+2G2G2A2B2O3C2O
2B2O3D2E1O2F2D+2F2.G3.G+2F2D+2G+2F+2E2F+2.F3.C+2C2D+1C+2C+2A+2.O3C3.C+
2O2G+2A+2G+2F+2O3D+2B2O3D+2C+2
```

The number of generations needed to reach a given distance to the guide depends on the guide length and the random seed used in each experiment, and follows an approximate Poisson curve, as shown in 4.18, which represents the result of one experiment.

In our second experiment, we used two guide pieces simultaneously: Begin the beguine (Begin.wav), and My heart belongs to daddy (Heart.wav), both by Cole Porter. In this case, the fitness function to be minimized was the sum of the NCDs of the note intervals of the evolving individuals to the note intervals of the two guide pieces.

The following represents one of the results we obtained, which we named as Porter_NEW.wav, which happens to be at a distance of 0.67 from the first guide piece, and 0.72 from the second, while the NCD between both guide pieces is 0.81, i.e. the generated piece was nearer to both guides than they are among themselves:

T5O3C+3.D3.O2A3.O3F+1.F+3.D+3.O2G+3.O3C+1O2G+3.C+3.D+3.D3.F1D+3.C+3.C3.C3.C+3.D+1O3C3.D3.F3.D1F3.E3.O2G+3.O3D+2C2O2A+1O3C3.O2A+3.A+3.A+1A+1G+3.E1.F+3.O4C3.O3F3.G1.F+3.C+3.D+3.E1G+3.E3.E3.O2C3.D+1C+3.D+3.O3C3.C3.G+3.C+1D+2E2F+3.E1.O2B3.O3G+3.O2C3.C+3.C+1C+3.F3.G3.G1F1D+3.O3C1C3.O2A3.D3.A+3.O3C1D3.O2F+3.F+3.D3.G3.F1O3D3.E2D+2E1.D+3.G+3.O2D+3.D+1G3.A3.G+3.O3C3.O2A1A3.E3.F+3.G3.B3.G1D+3.D+3.F3.A+3.B1O3D+3.C+3.F+3.F+1.E3.D+3.D+3.C2O2B1O3D+3.C3.O2B2O3E2O2A1A2G+3.G+3.E2.F+3.F3.D+2F1D3.D+3.O3F2D+3.F+3.D1O2A2G+3.O3C+3.G+2F2C+2O2A2F1O3F+3.B2.O2F+3.E3.G3.F+1E3.E3.D+3.C+3.O3C+1.O+2G+1C+3.C+1D+3.F3.A+2G+2G+3.F+1D+2E3D0D+2F3.D+3O3G3.D2B2O2D+2O3C3C3C3.C2O2B0A3.A3A3.B2.O3C3.F+1G3.A+3.G+3F+3A3.F1.C2O2G+3.G+3.O3G+0A+3.B3.B0B3.O2E3A+3B3.O3D3.D3.O2A+1O3D+3F+3F0F+3.D+3F3G3.G3.G3.G+2A+3O4C3O3A3A+2G+0O2F+3G+3F+3.F3.F+3.O3G+2F+3A3A+3G+3.F+1D+3.C+3.C3.O2A+3.A+0A+3.O3C3.O2A+3.O3C3.C+0D3.C3.O2C3.

To obtain the preceding piece, we completed the succession of notes generated by the genetic algorithm with the required length information, in the following way: each note was assigned the average lengths of the two notes in the same position in the two guide pieces (the guide pieces were shortened or circularly extended to make them the same length as the generated piece).This approach happens to provide a more esthetically appealing result than the one obtained when the length of only one of the guide pieces is used.

In our third experiment, Chopin preludes numbers 4 (Chopin4.wav) and 7 (Chopin7.wav) were used as simultaneous guides. The result (Chopin47_NEW.wav) came to be at distances of 0.61 and 0.74 from the two guide pieces, which are separated from one another by a distance of 0.96. The length of the notes was generated in the same way as in the preceding experiment. Compared with this, the piece obtained using as guides two works by Cole Porter has a distinctly lighter sound.

T5O3G+2.O2A+2O3G1.O2A+1O3GOO3F+1.O3COO2B2.O3D+1.O3F+1O2F+OO2F+1.O2GOO2F+2.O2F1.O2E2.O2E2O2EOO2B1.O3C2O3D+3.O3D+2.O3D+2.O3D1O3C+2.O2A+1.O2A2O2G+OO2G+2O2A1O3C2.O3E2O3G3.O2B2.O3D2.O3C1O4C2.O4C2O2C3O2DOO2F1O2D+OO2A1O3F+1.O3G2O3E2.O2F+2O2B1.O2B2O2B3.O2D+4O2G+2O2F1O2G+1O2F2O2F+2O2A+3.O2A+2.O2A+2.O2C+1O2A+2.O2A+2O2A3.O2A+2.O3C+2.O3F1O2B2O2B2O3C+2.O2B2.O3BOO3B1O2B2.O3F+1.O3G2O3B2O2BOO3C+1.O2BOO3C+2.

We performed another two experiments which the reader can also find available online.

The third (in the order presented in this section) generates a piece (Chopin7_NEW.dat) at distance 0.39 from its guide, the Chopins's seventh prelude. The lengths were generated as in the first experiment.

T5O4C1O3E2.C+3A+1A+1A+OO2A+1O3C+2.O2B3O3B1B1BOD+1D+2.E3G+1O4C1O3EOC+1D2.F3F1F1O2BOC+1A+2.B3G+1G+1G+OO3G+1G+2.G+3O2B1O3E1EOE1O2B2.O3F+3D+1E1AOA1A2.

Finally, in the last experiment, two pieces by Mozart were used as simultaneous guides: a few bars of the first movement in symphony 40 (Mozart40.wav), and a part of the second movement in sonata KV545 (KV545.wav). The result (Mozart_NEW.wav), which sounds like a mixture of both sources to the ear in some parts, happens to be at distances of 0.65 and 0.58 from the two guide pieces, which on the other hand differ from one another by a distance of 0.90.

T5O4G+OF+3B3.A+3A3G+2.G+3B1F1G+1.F3.D+3D+3D3.C3O3A+3O4F2.G+3F1C+1F+2.D+3.F+2E2D2C+2O3F2.F+3.G+1O4C1O3A3.F3.O4F3.G3F3F3.G3E3G3.B3O3E3G3.F2.G+3G4G+2A2O4G2G+2G1G+3.F3.G+3.O5C3O4A+3G+3.A+3.A+2.G+3.G3.G3.E3D+3O3F+3F3F+3.G3.G+3.A3.A3.G3F+3F+3.D+3.D3.O4D3.C+2C+3E2O3A+2O4C+3C2O3B2G1B1O4D3.C3.O3G3.O4D+3G3D3.D+3D3D+3.D3E3F3.G3.F3.E3F3G1O3D+3E3D+3.D+3B3B3.A+3.O4F3.G2.G+3A+3.G3

The length of the notes was generated in the same way as in the second experiment.

4.3.5 Testing different recombination procedures

As it happens in many genetic programming applications, we have evidence that that the recombination operator plays a key role in our approach both in the quality of the generated musical pieces and in the time the algorithm takes for it.

In order to to fine tune the genetic algorithm for this application, we devote a subsection to present and discuss several variations tested experimentally. The following strategies were used:

Strategy 1: given a pair of genotypes, $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_m)$, a random integer is generated in the interval $[0, \min(n, m)]$, let it be i. The resulting recombined genotypes are: $(x_1, x_2, \ldots, x_{i-1}, y_i, y_{i+1}, \ldots, y_m)$ and $(y_1, y_2, \ldots, y_{i-1}, x_i, x_{i+1}, \ldots, x_n)$. This is the base case (the simplest recombination strategy) which was used in all the experiments described in the preceding subsection.

Strategy 2: given a pair of genotypes, $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_m)$, two random integers are generated in the interval $[0, n]$ (let us call them $i, j, i < j$) and another two in the interval $[0, m]$ (let us call them $p, q, p < q$). The resulting recombined genotypes are: $(x_1, x_2, \ldots, x_{i-1}, y_p, y_{p+1}, \ldots, y_{q-1}, x_j, x_{j+1}, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_{p-1}, x_i, x_{i+1}, \ldots, x_{j-1}, y_q, y_{q+1}, \ldots, y_m)$.

Strategy 3: given a pair of genotypes, $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_m)$, four random ordered integers are generated in the interval $[0, n]$, $[0, m]$ for each parent genotype. Each genotype is then cut into the five corresponding pieces, which are shuffled together (one of them is reversed). Finally, the genotypes of the progeny are obtained by concatenating five of the pieces in the shuffled set

Strategy 4: similar to the preceding one, but only three random ordered integers are used to divide the parent genotypes into four pieces, which are then joined, shuffled, and used (four at a time) to generate the genotypes of the progeny.

numb. gener.	strategy 1	strategy 2	strategy 3	strategy 4	mixed strategy 1
1	0.930	0.930	0.930	0.930	0.930
100	0.782 (-0.148)	0.766 (-0.164)	0.807 (-0.123)	0.791 (-0.139)	0.766 (-0.164)
200	0.734 (-0.048)	0.710 (-0.056)	0.756 (-0.051)	0.744 (-0.047)	0.697 (-0.069)
300	0.714 (-0.020)	0.692 (-0.018)	0.740 (-0.016)	0.712 (-0.032)	0.676 (-0.021)
400	0.702 (-0.012)	0.692 (-0.000)	0.722 (-0.018)	0.704 (-0.008)	0.659 (-0.017)
500	0.690 (-0.012)	0.689 (-0.003)	0.722 (-0.000)	0.704 (-0.000)	0.648 (-0.011)
600	0.681 (-0.009)	0.683 (-0.006)	0.716 (-0.006)	0.704 (-0.000)	0.643 (-0.005)
1000	0.663 (-0.018)	0.682 (-0.001)	NP	NP	NP
1500	0.658 (-0.005)	0.666 (-0.016)	NP	NP	NP
2000	0.656 (-0.002)	0.658 (-0.008)	NP	NP	NP
2500	0.644 (-0.012)	0.652 (-0.006)	NP	NP	NP

Table 4.3: A comparison of the performance of different recombination strategies for a typical music generation experiment. NP stands for 'Not Performed.'

The one-point crossing-over strategy 1 has the property that the lengths of the parent genomes are invariant under recombination in the progeny. Since mutation also keeps the length of the genome, only fusion and elision change it. In fact, we did notice that, in our preceding experiments, fusion almost never leads to a fitter genome, while elision sometimes does, which means that the version of our genetic algorithm described in the previous subsection, which starts with a genome length copied from one of the target pieces of music, leads to genome lengths usually reduced by a little from their initial value.

Strategies 2, 3 and 4, however, all lead to progeny genomes with lengths usually quite different from those of their parents (even when both parent genomes had the same length), which provides the population with a larger genome length diversity than strategy 1.

After performing several experiments we noticed that, at the beginning of the evolution, the second recombination strategy converges more quickly towards the target, but after a certain number of generations (usually between 150 and 200), the first and fourth strategies becomes better, while beyond about 500 generations after the beginning of the process the first strategy is clearly the best. Above 1000 generations, the first two strategies tend to converge, i.e. to obtain similar distances

to the goal after the same number of generations.

This brought us to our add two new strategies to the testbed which are simple combinations of the four described above:

Mixed strategy 1: In the first 150 to 200 generations, the algorithm uses the second strategy (the two point recombination procedure with four different crossing-over points between both parents). During all the remaining generations, the first strategy is used instead (i.e., the one point recombination procedure with a single crossing-over point for both parents).

Mixed strategy 2: In the first 200 generations, the program uses the second strategy; between generations 200 and 500 it switches to the fourth strategy, and above 500 generations it uses the first strategy.

The data in Table 4.3 corresponds to a typical experiment in which two Mozart's pieces were used as the guide set: Symphony 40 and KV545. The results of the combined strategies are much better than those of any of the four strategies applied separately. It can be observed that the mixed strategy 1 reaches, in just 600 generations, target distances similar to those attained by the first two strategies in over 2500 generations.

Tabulated values for the mixed strategy 2 are not shown in Table 4.3 due to its great similarity in performance to the mixed strategy 1. In case of strategy 3 and 4, no data is shown for more than 600 generations; now the reason is that they are clearly outperformed by strategy 1 and 2 in all executions before that point and no further improvement was experimentally observed.

Figure 4.19 shows a graphical representation of the results. Figure 4.20 shows the results of a different experiment with the same three strategies on the same guide set. Summarizing, the improvement of the mixed strategies is quite impressive. On the other hand, the two mixed strategies attain comparable results.

In our analysis of the reasons for this behavior, we come to the conclusion that,

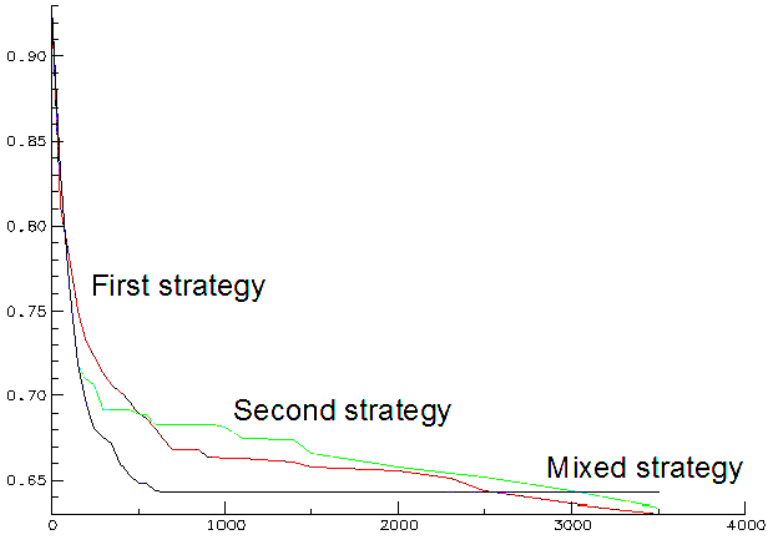

Figure 4.19: A comparison between three different recombination strategies. 'Mixed strategy' refers to the mixed strategy 1.

with the first strategy, the population reaches a small genetic variability where favorable mutations have a great probability of appearing. On the other hand, the second strategy generates large genetic variability, both with respect to genome lengths and contents, where favorable mutations are much hard to come by. This means that, on the long range, the first strategy should work better than the second, which on the other hand gets faster results during the first part of the process by evolving simultaneously in many directions and testing widely different genomes at the same time. Thus, the mixed strategies makes the best use of both recombination procedures, which is the reason for its outstanding performance success.

4.3.6 Discussion

We have found that that the Normalized Compression Distance is a promising fitness function for genetic algorithms used in automatic music generation. Some of the pieces of music thus generated recall the style of well-known authors, in spite of the fact that the fitness function only takes into account the relative pitch envelope. Our

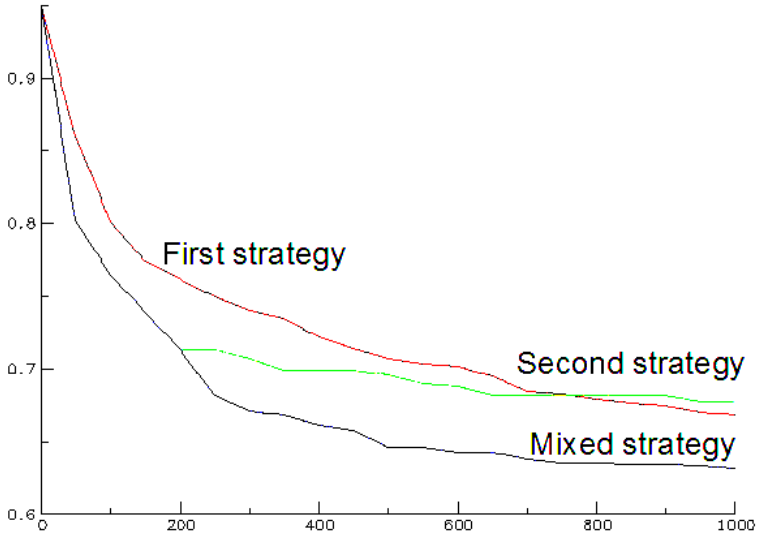

Figure 4.20: Performance comparisons of another experiment with the same recombination strategies as in Figure 4.19.

results have been qualitatively superior to those obtained previously with a different fitness function [124].

Several recombination operators have been tested to fine tune the genetic algorithm for this application, finding that mixed strategies which promote diversity in the first generations and then change to a more exploitative strategy give the best results. This scheme of initial exploration and posterior exploitation is analogue to the idea behind Simulated Annealing [95].

In the future we intend to combine our results with those of other authors [104, 35] to use as the target for the genetic algorithm, not one or two pieces of music by a given author, but a cluster of pieces by the same author, thus trying to capture the style in a more general way.

Although we have introduced the information about note duration in the genetic process, we have ignored it so far. As the current design of our algorithm facilitates this (the NCD can easily deal with integers representing note lengths) we intend to perform a new set of experiments to evolve the note length information along with the melody.

We shall also work with a standard and richer system of music representation, such as MIDI.

The results presented in this section serve as a proof-of-concept. As future research, we plan to provide a comparison with state-of-the-art music composition techniques from machine learning to reveal both the strengths and weaknesses of our proposal.

Chapter 5

New applications of Algorithmic Stochastic Modeling

In this chapter we present three new applications of Algorithmic Stochastic Modelling, a methodology introduced in Section 2.2 (page 17).

5.1 Accelerated Generation of Fractals of a Given Dimension

In the last decades, genetic algorithms, which emulate biological evolution in computer software, have been applied to ever wider fields of research and development and have given rise to a few astounding successes, together with a certain amount of disappointment, frequently related to the apparently inherent slowness of the procedure. This is not a surprise, as biological evolution, which serves as the source for most of the ideas used by the research in genetic algorithms, is an extremely slow and difficult to experiment field, where actual processes require millions of years in many cases. This slowness is in part a consequence of the fact that randomness is a basic underlying of the search performed by genetic algorithms. For this reason, the discovery and proposal of general procedures to accelerate their execution time is one of the most interesting open questions in this field.

The procedure we propose in this thesis has made it possible to increase by an order of magnitude the performance of at least one application of genetic algorithms: the use of grammatical evolution to generate fractal curves of a given dimension. It is probable that the application of the same procedure may be useful to accelerate many other applications of similar techniques, although there are cases where it cannot provide any improvement [15]. This work offers ways to predict the situations where this procedure may be useful, and recognize those where it will not provide any improvement, by analyzing the statistical distributions of the execution time of the algorithms. In fact, the family of heavy-tail distributions embodies those applications where the best improvement can be attained by the application of re-start techniques, while another family (leptokurtic distributions) also offer a significant acceleration.

The remainder of this introduction contains a simple exposition of the two main fields affecting the experiment we have used as the template for the experimentation of the acceleration techniques: fractal curves and their dimension; and L systems, which provide an easy way to represent the former and making their computation straightforward.

Subsection 5.1.1 summarizes an algorithm we have developed and described in a previous publication, which makes it possible to compute the dimension of a fractal curve from its equivalent L system. Subsection 5.1.1 describes the concrete case we have used as the benchmark for our acceleration techniques: a genetic algorithm which generates a fractal curve with a given dimension. This algorithm has also been previously published in the scientific literature.

Subsection 5.1.2 describes the families of heavy tail and leptokurtic distributions, where the acceleration techniques proposed in this work are most useful. Subsection 5.1.3 proves that the experiment described in subsection 5.1.1 gives rise to execution time distributions belonging to those families. Subsection 5.1.4 describes the restart strategy whose use significantly accelerates the execution time of our algorithm and all others with a distribution in the same families. Finally, subsection 5.1.5 offers the conclusions of this thesis chapter and proposes several lines of future work.

Fractals and fractal dimension

The concept of dimension is very old and seems easy and evident: sometimes it can be clearly and elegantly defined as the number of directions in which movement is allowed: with this interpretation, dimensions are consecutive integers: 0 (a point), 1 (a line), 2 (a surface), 3 (a volume), with no doubtful cases. This is called a *topological dimension*. However, as Mandelbrot et al. describe in his seminal article [113], some doubtful cases exist: depending on the size of the observer, a ball of thread can be considered as a point (dimension 0, for a large observer), a sphere (dimension 3, for an observer comparable to the ball), a twisted line (dimension 1, for a smaller observer), a twisted cylinder (dimension 2, for an even smaller observer), and so forth.

There is a class of apparently one-dimensional curves for which the concept of dimension is tricky: in 1890, Giuseppe Peano defined a curve which goes through every point in a square, and therefore can be considered as two-dimensional. In 1904, Helge von Koch devised another, whose shape reminds a snowflake and whose longitude is infinite, although the surface it encloses is limited. Von Koch's snowflake does not fill a surface, therefore its dimension should be greater than 1 but less than 2. In 1919, Hausdorff proposed a new definition of dimension, applicable to such doubtful cases: curves such as those just described may have a fractional dimension, between 1 and 2. Peano's curve has a Hausdorff dimension of 2; Von Koch's snowflake has a Hausdorff dimension of 1.2618595071429... Other alternative definitions of dimension were proposed during the twentieth century, such as the Hausdorff-Besicovitch dimension, the Minkowsky dimension, or the boxcounting dimension (see [60, 154]). They differ only in details and are known as *fractal dimensions*.

The name *fractal* was introduced in 1975 by Mandelbrot and applies to objects with some special properties, such as a fractal dimension different from their integer topological dimension, self-similarity (containing copies of themselves), and/or non-differentiability at every point.

Fractal curves have been generated or represented by different means, such as fractional Brownian movements, recursive mathematical families of equations (such as those that generate the Mandelbrot set), and recursive transformations (generators) applied to an initial shape (the initiator). They have found applications in antenna

design, the generation of natural-looking landscapes for artistic purposes, and many other fields. The generation of fractals with a given dimension can be useful for some of these applications.

This work discusses only the initiator-generator family of fractals.

L systems

L systems, devised by [107], also called parallel-derivation grammars, differ from Chomsky grammars because derivation is not performed sequentially (a single rule is applied at every step) but in parallel (every symbol is replaced by a string at every step). L systems are appropriate to represent fractal curves obtained by means of recursive transformations [44]. The initiator maps to the axiom of the L system; the generator becomes the set of production rules; recursive applications of the generator to the initiator correspond to subsequent derivations of the axiom. The fractal curve is the limit of the word derived from the axiom when the number of derivations tends to infinity.

Something else is needed: a graphic interpretation which makes it possible to convert the words generated by the L system into visible graphic objects. Two different families of graphic interpretations of L systems have been used: turtle graphics and vector graphics. In [16], we have proved an equivalence theorem between two families of L systems, one associated with a turtle graphics interpretation and the other with vector graphics. Our theorem makes it possible to focus only on turtle graphics without a significant loss of generality.

The turtle graphics interpretation was first proposed by [127] as the trail left by an invisible *turtle*, whose state at every instant is defined by its position and the direction in which it is looking. The state of the turtle changes as it moves a step forward or as it rotates by a given angle in the same position. Turtle graphics interpretations may exhibit different levels of complexity. We use here the following version:

- The angle step of the turtle is $\alpha = (2k\pi/n)$, where k and n are two integers.
- The alphabet of the L system is expressed as the union of the four disjoint subsets: N (non-graphic symbols), D (visible graphic symbols, which move the

turtle one step forward, in the direction of its current angle, leaving a visible trail), M (invisible graphic symbols, which move the turtle one step forward, in the direction of its current angle, leaving no visible trail) and extra symbols such as $\{+, -\}$, which increase/decrease the turtle angle by α, or a parenthesis pair, which are used in conjunction with a stack to add branches to the images. These symbols usually are associated with L system trivial rules such as $+ ::= +$. In the following, the trivial rules will be omitted but assumed to be present.

A string is said to be *angle-invariant* with a turtle graphics interpretation if the directions of the turtle at the beginning and the end of the string are the same. In this work we only consider *angle-invariant D0L systems* (where D0L describes a deterministic context-free L system), i.e. the set of D0L systems such that the right-hand side of all of their rules is an angle-invariant string.

Summarizing: a fractal curve can be represented by means of two components: an L system and a turtle graphics interpretation, with a given angle step. The length of the moving step (the scale) is reduced at every derivation in the appropriate way, so that the curve always occupies the same space.

5.1.1 An algorithm to determine the dimension of a fractal curve from its equivalent L system

Several classic techniques make it possible to estimate the dimension of a fractal curve. All attempt to measure the ratio between how much the curve grows in length, and how much it advances. The ruler dimension estimation computes the dimension of a fractal curve as a function of two measurements taken while *walking* the curve in a number of discrete steps. The first measurement is the *pitch length* (p_l), the length of the step used, which is constant during the walk. The second is the number of steps needed to reach the end of the walk by walking around the fractal curve, $N(p_l)$. The fractal dimension, D_{p_l}, is defined as

$$D_{p_l} = \lim_{p_l \to 0^+} \frac{-\log N(p_l)}{\log p_l} \tag{5.1}$$

In a previous work [17] we presented an algorithm that reaches the same result by computing directly from the L system that represents the fractal curve, without performing any graphical representation. The L system is assumed to be an angle-invariant D0L system with a single draw symbol. The production set consists of a single rule, apart from trivial rules for symbols +, −, (, and). Informally, the algorithm takes advantage of the fact that the right side of the only applicable rule provides a symbolic description of the fractal generator, which can be completely described by a single string. The algorithm computes two numbers: the length N of the visible walk followed by the fractal generator (equal in principle to the number of draw symbols in the generator string), and the distance d in a straight line from the start to the endpoint of the walk, measured in turtle step units (this number can also be deduced from the string). The fractal dimension is:

$$D = \frac{\log N}{\log D} \tag{5.2}$$

The scale is reduced at every derivation in such a way that the distance between the origin and the end of the graphical representation of the strings is always the same. For instance, the D0L scheme associated with the rule

$F ::= F + F - -F + F$

with axiom $F - -F - -F$ and a turtle graphic interpretation, where F is a visible graphic symbol and the step angle is 60, represents the fractal whose fifth derivation appears in figure 5.1.

The string $F + F - -F + F$ describes the fractal generator. The number of steps along the walk (N) is the number of draw symbols in the string, 4 in this case. The distance d between the extreme points of the generator, computable from the string by applying the turtle interpretation, is 3. Therefore, the dimension is

$$D = \frac{\log 4}{\log 3} = 1.2618595074129\ldots \tag{5.3}$$

This is the same dimension obtained with other methods, as specified in [113, p. 42].

This algorithm can be easily extended to fractals whose L systems contain more

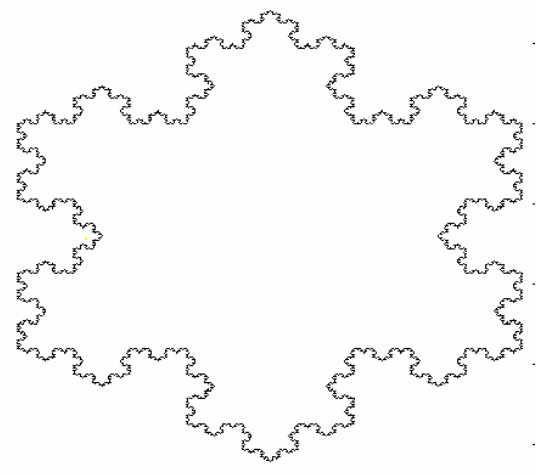

Figure 5.1: Von Koch snowflake curve.

than one draw symbol and more than one rule, if all the rules preserve the ratio between N and d in the previous expression. Most of the initiator-iterator fractals found in the literature can be represented by angle-invariant D0L systems whose draw symbols-contribute to the dimension in this way. The algorithm was also refined to successfully include fractal curves which overlap, either in the generator itself, or after subsequent derivations. In those cases, the definition of the fractal dimension is replaced by

$$D = \lim \frac{\log N}{\log d} \tag{5.4}$$

where the limit is taken when the number of derivations goes to infinity. Our algorithm computes this case by computing the dimension of a certain number of derivations until the quotient converges.

Grammatical evolution to design fractal curves with a given dimension

Designing fractal curves with a given dimension is relatively easy for certain values of the desired dimension (for instance, 1.261858... or $\log 4/\log 3$), but very difficult for others (the reader can try to hand design a fractal curve with a dimension of 1.255). To do it, one has to find two integer numbers, a and b, such that $1.255 =$

$\log a / \log b$. Then one has to design a geometrical iterator such that it would take a steps to advance a distance equal to b.

This problem can be solved automatically by means of grammatical evolution. Our genetic algorithm acts on genotypes consisting of vectors of integers and makes use of a fixed grammar to translate the genotypes into an intermediate level, which can be interpreted as a single rule for an L system which, together with a turtle graphic interpretation, generates the final phenotype: a fractal curve with a dimension as approximate as desired to the desired value. The algorithm can be described as follows:

1. Generate a random population of 64 vectors of eight integers in the $[0, 10]$ or the $[0, 255]$ interval (the latter case introduces genetic code degeneracy). All the genotypes in the initial population have the same length. Subsequent populations may contain individuals with genotypes of different lengths.

2. Translate every individual genotype into a word in the alphabet $F, +, -$ as indicated below.

3. Using the algorithm described in section 5.1.1, compute the dimension of the fractal curve represented by the D0L system which uses the preceding word as a generator.

4. Compute the fitness of every genotype as $(\text{target} - \text{dimension})^{-1}$.

5. Order the 64 genotypes from higher to lower fitness.

6. If the highest-fitness genotype has a fitness higher than the target fitness, stop and return its phenotype.

7. From the ordered list of 64 genotypes created in step 5, remove the 16 genotypes with least fitness (leaving 48) and take the 16 genotypes with most fitness. Pair these 16 genotypes randomly to make eight pairs. Each pair generates another pair, a copy of their parents, modified according to four genetic operations (see below). The new 16 genotypes are added to the remaining population of 48 to make again 64, and their fitness is computed as in steps 2 to 4.

8. Go to step 5.

The algorithm has three input parameters: the target dimension (used in step 4), the target minimum fitness (used in step 6) and the angle step for the turtle graphics interpretation (used in step 3).

In step 2, the following grammar is used to translate the genotype of one individual into its equivalent intermediate form (the generator for an L system representing a fractal curve):

$0 : F ::= F$
$1 : F ::= FF$
$2 : F ::= F+$
$3 : F ::= F-$
$4 : F ::= +F$
$5 : F ::= -F$
$6 : F ::= F + F$
$7 : F ::= F - F$
$8 : F ::= +$
$9 : F ::= -$
$10 : F ::= \lambda$

The translation is performed according to the following developmental algorithm:

1. The axiom (the start word) of the grammar is assumed to be F.

2. As many elements from the remainder of the genotype are taken (and removed) from the left of the genotype as the number of times the letter F appears in the current word. If there remain too few elements in the genotype, the required number is completed circularly.

3. Each F in the current word is replaced by the right-hand side of the rule with the same number as the integers obtained by the preceding step. With genetic code degeneracy, the remainder of each integer modulo 11 is used instead. In any derivation, the trivial rules $+ ::= +$ and $- ::= -$ are also applied.

4. If the genotype is empty, the algorithm stops and returns the last derived word.

5. If the derived word does not contain a letter F, the whole word is replaced by the axiom.

6. Go to step 2.

The four genetic operations mentioned in step 7 of the genetic algorithm are the following:

- *Recombination* (applied to all the generated genotypes). Given a pair of genotypes, $(x_1, x_2, ..., x_n)$ and $(y_1, y_2, ..., y_m)$, a random integer is generated in the interval $[0, \min(n, m)]$. Let it be i. The resulting recombined genotypes are $(x_1, x_2, ..., x_{i-1}, y_i, y_{i+1}, ..., y_m)$ and
  $(y_1, y_2, ..., y_{i-1}, x_i, x_{i+1}, ..., x_n)$.

- *Mutation*, applied to n_1 per cent of the generated genotypes if both parents are equal, to n_2 per cent if they are different. It consists of replacing a random element of the genotype vector by a random integer in the same interval.

- *Fusion*, applied to n_3 per cent of the generated genotypes. The genotype is replaced by a catenation of itself with a piece randomly broken from either itself or its brother's genotype. (In some tests, the whole genotype was used, rather than a piece of it.)

- *Elision*, applied to 5 per cent of the generated genotypes. One integer in a random position of the vector is eliminated. The last two operations make it possible to generate longer or shorter genotypes from the original eight element vectors.

5.1.2 Heavy tail distributions

Heavy tail distributions are probabilistic distributions which exhibit an asymptotic hyperbolic decrease, usually represented as

$$\Pr\{|X| > x\} \sim Cx^{-\alpha}, \tag{5.5}$$

where α is a positive constant. Distributions with this property have been used to model ramdom variables whose extreme values are observed with a relatively high probability.

Work on these probability distributions can be traced to Pareto's work on the earning distribution (1965) or to Levy's work on the properties of stable distributions (1937). A fundamental advance in the use of heavy tail distributions for was provided by Mandelbrot's work [111, 112] on the application of fractal behavior and self-similarity to the modeling of real-world phenomena, which he used to introduce stable distributions to model price changes in the stock exchange. Heavy tail distributions have also been used in areas such as statistical physics, wheather prediction, earthquake prediction, econometrics and risk theory [59, 113]. In more recent times, these distributions have been used to model waiting times in the World Wide Web [152] or the computational cost of random algorithms [73, 74, 75, 76].

For many purposes, the only relevant parameter of a heavy tail distribution is its *characteristic exponent* α, which determines the ratio of decrease of the tail and the probability of occurrence of extreme events. In this thesis we only consider heavy tail distributions where α belongs to the $(0, 2)$ interval, with positive support ($\Pr\{0 \leq X < \infty\} = 1$).

The existence or inexistence of the different moments of a distribution is fully determined by the behavior of its tail: α can also be regarded as the exponent of the maximum finite moment of the distribution, in the sense that moments of X of order less than α are finite, while moments of order equal or greater are infinite. For instance, when $\alpha = 1.5$, the distribution has a finite average and an infinite variance, while for $\alpha = 0.6$ both average and variance are infinite.

Estimation of the characteristic exponent

Many procedures have been used to estimate α [88, 11, 43]. Two of them have received the most extensive usage. The first uses a maximum likelihood estimator, the second applies a simple regression method.

An important issue while estimating α is how to tackle censored observations when extreme data are not available. Consider, for instance, physical phenomena such as

wind velocity or earthquake magnitude, where heavy tail distributions have been considered appropriate. In these cases, extreme measures are non-observable, since very strong hurricanes or highly destructive earthquakes will damage the measuring instruments. In the process of financial data, such as stock exchange rates, heavy tail models have also been used [52]. In moments of high volatility, when extreme data usually appear, many stock exchange markets introduce rules to limit transactions or even close the market, to prevent them from taking place. Consider finally the case of random algorithms: the computational costs of some problems are so high, that the algorithms have no alternative but to interrupt the execution and start again with a different random seed. In those cases, computational costs are not observable beyond a certain threshold [77]. Thus the censorship of extreme values needs to be considered by available estimators.

Let $X_{n1} \leq X_{n2} \leq \ldots X_{nn}$ be the ordered statistics, i.e. the ordered values in the sample $X_1, X_2, \ldots, X_n$. Let $r < n$ be the truncation value which separates normal from extreme observations.

The adapted Hill-Hall estimator for censored observations is:

$$\hat{\alpha}_{r,u} = (\frac{1}{r}\sum_{j=1}^{r-1} \ln X_{n,n-r+j} + \frac{u+1}{r} \ln X_{n,n} - \frac{u+r}{r} \ln X_{n,n-r})^{-1}. \tag{5.6}$$

In this notation, n is the number of observed data, $r+1$ is the number of larger observations selected and u is the number of non-observed extreme values. If all the data are observable, $u = 0$ and equation 5.6 becomes the classic Hill-Hall estimator.

In heavy tail distributions, the ratio of decay of the estimated density is hyperbolic (slower than a exponential decay). Thus the one-complement of the accumulated distribution function, $\overline{F}(\cdot)$, also shows a hyperbolic decay.

$$\overline{F}(x) = 1 - F(x) = \Pr\{X > x\} \sim Cx^{-\alpha}. \tag{5.7}$$

Therefore, for a heavy-tail variable, a log-log graph of the frequency of observed values larger than x should show an approximately linear decay in the tail. Moreover, the slope of the linear decaying graph is in itself an estimation of α. This can be contrasted

with a exponentially decaying tail, where a log-log graph shows a faster-than-linear tail decay.

This simple property, besides giving visual evidence of the presence of a heavy tail, also gives place to a natural regression estimator based on equation 5.7, the least-squares estimator [11], which can be expressed in terms of a selected number of extreme observations. Assume that we have a sample of $k = n + u$ independent identically distributed random variables. Assume also that we only observe the n smallest values of random variable X and therefore have the ordered statistics $X_{n1} \leq X_{n2} \leq \ldots \leq X_{nn}$. Assume that, for $X_{n,n-r} \leq X \leq X_{nn}$, the tail distribution has a hyperbolic decay. The least-square regression estimator for the α exponent is

$$\hat{\alpha} = -\frac{\sum l_i \log X_{ni} - \sum l_i \sum \log X_{ni}/(r+1)}{\sum (\log X_{ni})^2 - \sum (\log X_{ni})^2/(r+1)}, \tag{5.8}$$

where $l_i = \log \frac{n+u-i}{n+u}$ and the sums go from $i = n - r$ to $i = n$. If all the values in the sample $k = n + u$ can be observed, then $u = 0$ and $k = n$.

Leptokurtic distributions vs. heavy tail distributions

The name heavy tail, applied to a class of distributions, expresses their main property: the large proportion of total probability mass concentrated in the tail, which reflects its (hyperbolic) slow decay and is the reason why all the moments of a heavy tail distribution are infinite, starting at a given order.

The concept of *kurtosis* is also related to the tail heaviness. The kurtosis of a distribution is the amount μ_4/μ_2^2, where μ_2 and μ_4 are the second and fourth centralized moments (μ_2 is the variance). The kurtosis is independent of the localization and scale parameters of a distribution. Kurtosis is high, in general, for a distribution with a high central peak and long tails.

The kurtosis of the standard normal distribution is 3. A distribution with a kurtosis higher than 3 is called *leptokurtic* as opposite to *platokurtic* (see fig. 5.2). In a similar way to heavy tail distributions, a leptokurtic distribution has long tails with a considerable concentration of probability. However, the tail of a leptokurtic distribution decays faster than that of a heavy tail distribution: all the moments

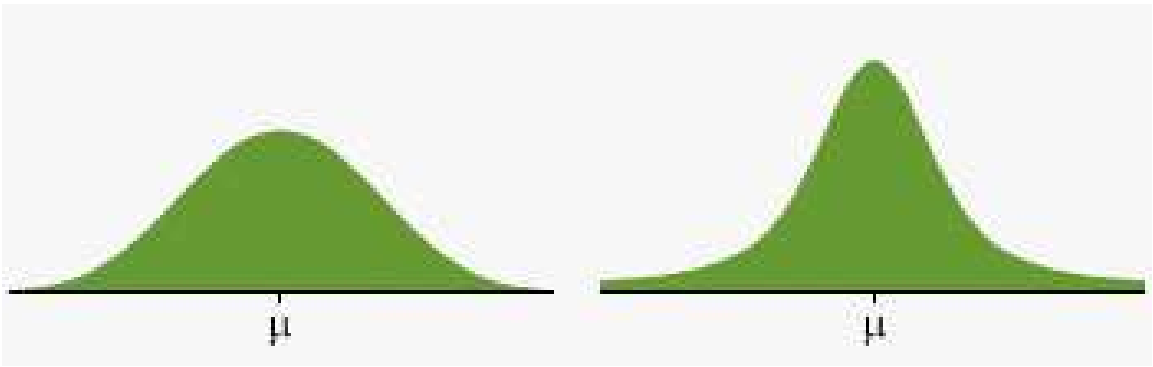

Figure 5.2: Low kurtosis (platokurtic distribution) vs. high kurtosis (leptokurtic distribution). The probability density function on the right has a higher kurtosis than the one on left: its center part has a higher peak and its tails are heavier.

in a leptokurtic distribution can be finite, in a strong contrast with a heavy tail distribution where, at most, the first two moments are finite.

5.1.3 Heavy tails in Grammatical Evolution

Randomized algorithms with a high execution time variability are suspetc of hiding a heavy tail distribution. In the present subsection we provide empirical evidence that our GE algorithm for the automatic generation of fractal curves may exhibit a heavy tail behavior which can be exploited to improve the performance.

[125] provides data about different executions of the same algorithm to generate fractal curves with the same dimensions, using different random seeds. The numbers of generations needed to reach the target differ in up to two orders of magnitude (see table 5.1).

Figure 5.3 shows the empirical distribution of the number of generations needed to find a solution, i.e.

$$F(x) = \Pr\{\text{number of generations to reach a solution} \leq x\} \tag{5.9}$$

for four different fractal dimensions: 1.3, 1.5, 1.8 and 2. The empirical distribution

dimension	angle (degrees)	# experiments	# generations range
1.1	45	10	[37, 9068]
1.1	60	4	[119, 72122]
1.2	45	8	[188, 11173]
1.2	60	10	[21, 750]
1.3	45	9	[50, 18627]
1.3	60	4	[14643, 66274]
1.25	60	2	[1198, 3713]
1.255	60	15	[1, 2422]
1.2618595...	60	4	[1, 2]
1.4	45	10	[79, 781]
1.4	60	10	[33, 1912]
1.5	45	11	[52, 11138]
1.5	60	8	[12, 700]
1.6	45	5	[275, 3944]
1.6	60	1	[116, 913]
1.7	45	2	[585, 1456]
1.7	60	8	[18, 1221]
1.8	45	2	[855, 2378]
1.8	60	13	[69, 3659]

Table 5.1: Number of generations needed to generate a fractal curve with a given dimension in a set of experiments.

functions have been obtained by running 1,000 executions with 1,000 different independent random seeds. At the end of each execution, the number of generations needed to reach a solution is recorded. We took a *censorship* value equal to $\tau = 5,000$ generations, meaning that, if an execution needs over 5,000 generations, it is stopped and marked as non-observable.

dimension	1.3	1.5	1.8	2
observable	80.5%	88.3%	66.2 %	100%
non-observable	19.5%	12.7%	38.2 %	0%

Table 5.2: Percentages of observable and non-observable executions for a censorship value $\tau = 5,000$ generations.

Table 5.2 shows the percentages of non-observable executions in our experiments. This percentage is quite high, specially for dimensions 1.8 and 1.3. The empirical

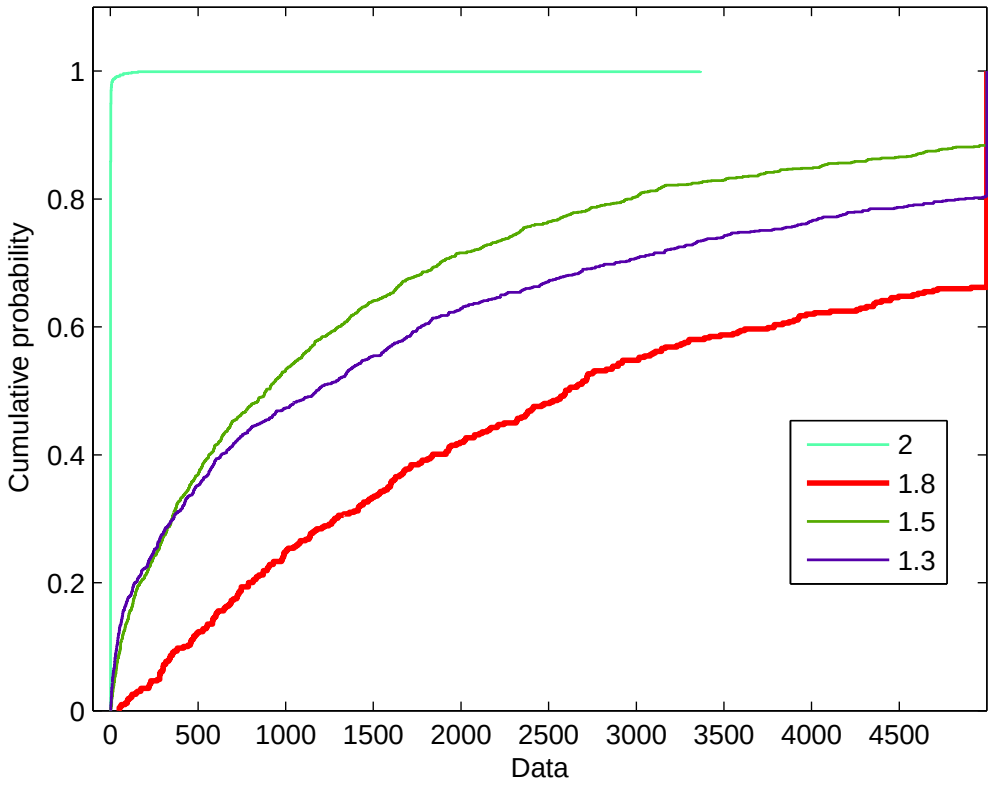

Figure 5.3: Empirical distribution function of the number of generations needed to reach a solution for several fractal dimensions: 1.3, 1.5, 1.8 and 2.

distribution functions may be used to test whether the distribution has a heavy tail.

In the previous subsection (definition 5.5) we saw that a random variable has a heavy tail behavior if it shows an *asymptotic* hyperbolic decay, although that behavior can also be shown in its whole support. In the figures displayed in this subsection, only the extreme values are shown, therefore we had to choose a parameter r to truncate the non-extreme observations. Usually r takes values in the $[1\%, 25\%]$ interval; we will use the set $\{1\%, 2.5\%, 5\%, 10\%, 15\%, 20\%\}$, as recommended by [43].

Figure 5.4 shows the log-log graphs of the distribution tails for fractal dimensions 1.3, 1.5, 1.8 and 2. Notice the linear decay of function $\log \overline{F}(x)$, in contrast with exponential decay distributions, where the decay of $\log \overline{F}(x)$ is faster than linear.

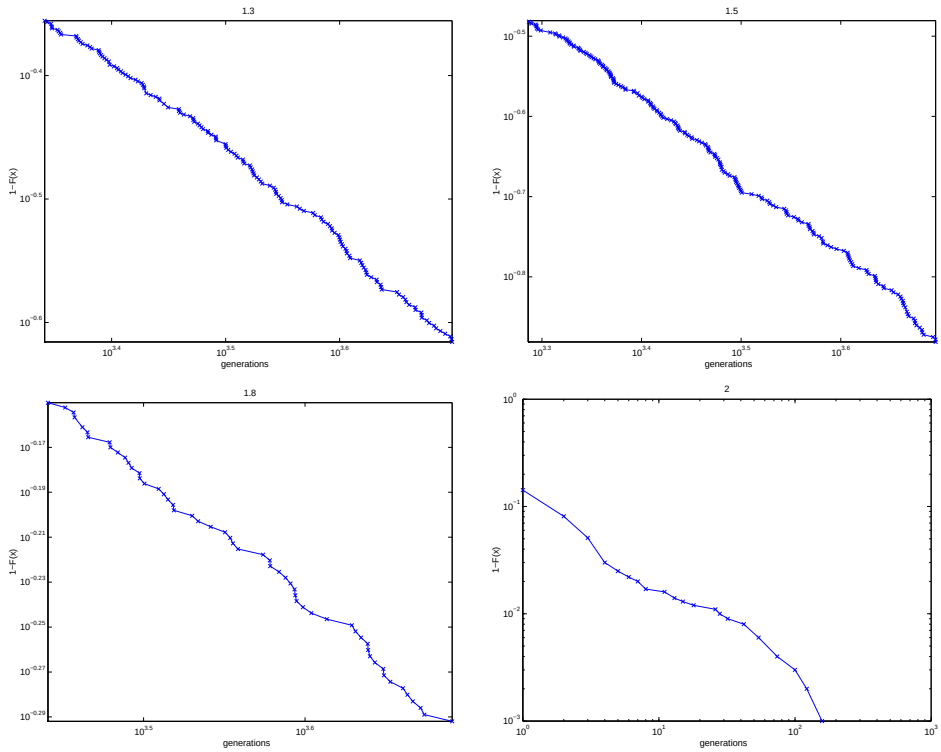

Figure 5.4: Log-log graph of the tail of (r=20%) distributions for dimensions 1.3, 1.5, 1.8 and 2.

The averages for dimensions 1.3, 1.5 and 1.8 are $E(X_{1.3}) = 1,173$, $E(X_{1.5}) = 1,108$ and $E(X_{1.8}) = 1,721$. It can be seen that, with a number of generations almost 5 times above their averages, respectively over 10%, 20% and 30% executions have not finished.

Figure 5.5 displays four *box-and-whisker* graphs, which give rise to three remarkable conclusions:

- The median (the dashed line within the box in fig. 5.5) is much smaller than the average (the cross '+' within the box) for dimensions 1.3, 1.5 and 1.8. This suggests that the average of these distributions is biased by the size of the sample, which means that they may have an infinite asymptotic average typical of heavy tail distributions.

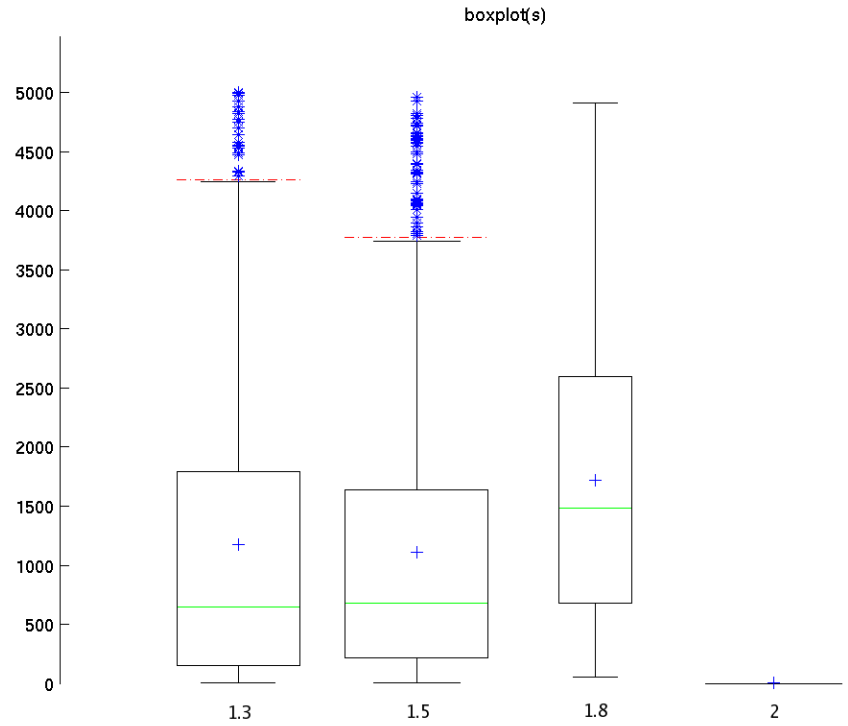

Figure 5.5: *Box-and-whisker* type graphs for dimensions 1.3, 1.5, 1.8 and 2.

- The sample distribution is strongly biased towards high execution times, indicating a right-hand-side heavy tail. This can be seen in the fact that the lower interquartilic distance (the difference between the first quartile - the lower segment of the box - and the the median - the green line) is shorter than the upper interquartilic distance (the difference between the median and the third quartile - the upper segment of the box). Besides this, the distance between the minimum and the first quartile is much less than the distance between the maximum (the highest point of the graph) and the third quartile.

Estimating the characteristic exponent

The preceding subsection provides visual evidence for a heavy tail behavior in dimensions 1.3, 1.5, 1.8. Evidence for this behavior is weaker in dimension 2, but also present in, for instance, the linear decay observed in figure 5.4. In this subsection we estimate the characteristic exponent for these distributions, using the estimators presented in subsection 5.1.2.

First we compute the Hill-Hall estimator adapted for censored observations, (equation 5.6).

dimension	r					
	1%	2.5%	5%	10%	15%	20%
1.3	0.7827	0.6796	0.8312	0.7953	0.7634	0.7084
1.5	1.1765	1.2400	1.0952	1.0595	0.9952	0.9418
1.8	0.3649	0.4855	0.6746	0.5759	0.5657	0.5705
2	0.7656	0.6043	1.0403	1.0463	0.7732	1.0309

Table 5.3: Estimations of α for dimensions 1.3, 1.5, 1.8 and 2 using the adapted Hill-Hall estimator.

Table 5.3 confirms that these distributions are heavy tailed, since all the values in the table are less than 2, the limit for heavy tail distributions.

For dimension 1.3, all the estimations (for all values of r) are less than 1, which means that this distribution does not have neither a finite average nor a finite variance. The same happens for dimension 1.8 even in a stronger way, as the values of α are even smaller (all are below 0.7).

Dimensions 1.5 and 2 provide examples of heavy tail distributions with a characteristic exponent α between 1 and 2. These distributions have a finite average, but an infinite variance, indicating that their right heavy tail is lighter than in the other two distributions.

Figure 5.6 displays the erratic behavior of the sample average as a function of the sample size.

To verify the reliability of our characteristic exponent estimation, table 5.4 shows the estimations obtained using the regression estimator described in a previous subsection (equation 5.8), which is considered slightly less robust than the maximum

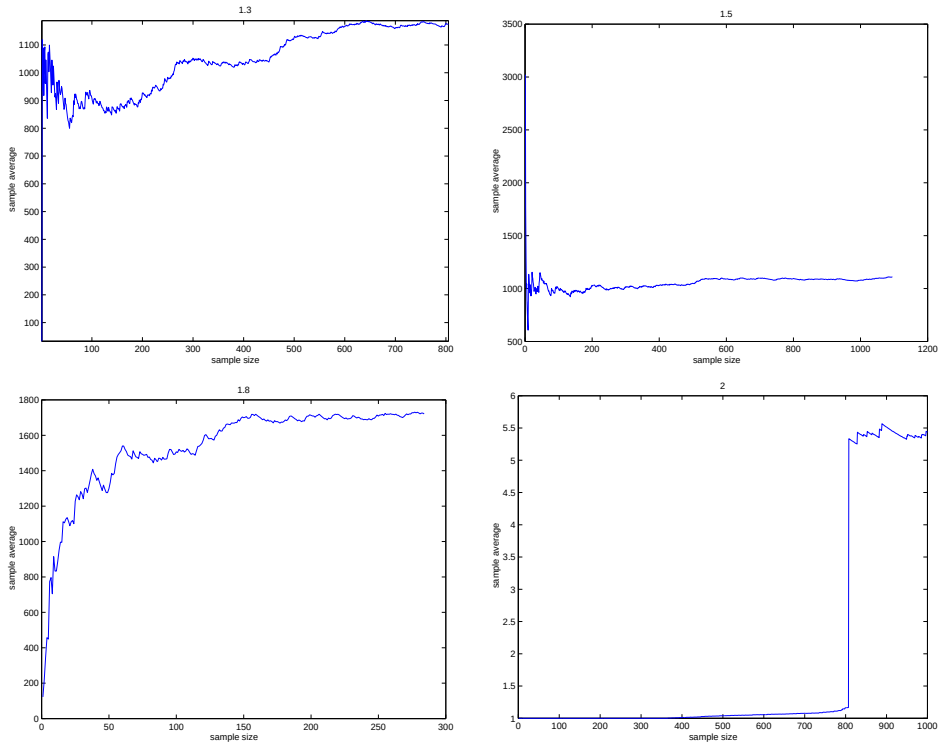

Figure 5.6: Evolution of the sample average as a function of the sample size for dimensions 1.3, 1.5, 1.8 and 2.

likelihood estimator (adapted Hill-Hall). The results of this estimator can be seen to be consistent with those of the adapted Hill-Hall estimator.

Tail truncation

As mentioned before, in practice one has to select the GE maximum number of generations for specially difficult problems. In other words, an appropriate censorship value τ must be chosen, so that the algorithm does not become stagnated in the extreme values of the distribution tail. As a consequence, the tail is truncated. The selection of the value of τ depends on the problem and the algorithm. Ideally, only a small portion of tail should be truncated, but this may be prohibitive from the computational point of view.

If the truncation is set at a small number of generations, it will be harder to distinguish between heavy tail and leptokurtic distributions. From a practical point

dimension	r					
	1%	2.5%	5%	10%	15%	20%
1.3	0.6952	0.7528	0.7715	0.7904	0.7692	0.7345
1.5	1.1318	1.3790	1.0886	0.9664	0.9786	0.9721
1.8	0.3310	0.5220	0.7285	0.6424	0.5762	0.5762
2	≈ 0	≈ 0	0.2554	0.4821	0.6008	0.6667

Table 5.4: Estimations of α for fractal dimensions 1.3, 1.5, 1.8 and 2, using the regression estimator.

of view, this is not a problem, if there are strong indications that the tail exhibits at least one of the two behaviors. A heavy tail behavior is not a necessary condition to accelerate randomized search methods. In fact, it has been proved that the efficiency of the search in leptokurtic distributions can be improved by randomized backtracking [73]. However, with a heavy tail distribution, the occurrence of long executions will be more frequent than with a leptokurtic distribution, making it possible to obtain a higher potential acceleration.

dimension	1.3	1.5	1.8	2
μ_2	1.6171e+06	1.3532e+06	1.5496e+06	69.9039
μ_4	9.0640e+12	7.6389e+12	6.0015e+12	9.6330e+05
kurt(x)	3.4575	4.1642	2.4817	197.1335

Table 5.5: Kurtosis computation for dimensions 1.3, 1.5, 1.8 and 2.

Table 5.5 shows the kurtosis for the 4 fractal dimensions considered. Remember that, if this value is greater than 3 (the kurtosis for a normal distribution) the distribution is leptokurtic (with abrupt peaks and heavy tails), otherwise it is platokurtic (with smooth peaks and light tails). In our case, fractal dimensions 1.3, 1.5 and 2 are seen to be leptokurtic, while dimension 1.8 is platokurtic. Figure 5.7 shows the histograms built for the execution samples for dimensions 1.3, 1.5, 1.8 and 2.

This subsection ends with the conclusion that there exists an application of GE, automatic fractal generation, whose distributions exhibit a heavy tail behavior, besides being leptokurtic in many cases. In the next subsection we show that it is possible to take advantage of this probabilistic characterization to increase the performance of GE and yield a fast fractal generation algorithm.

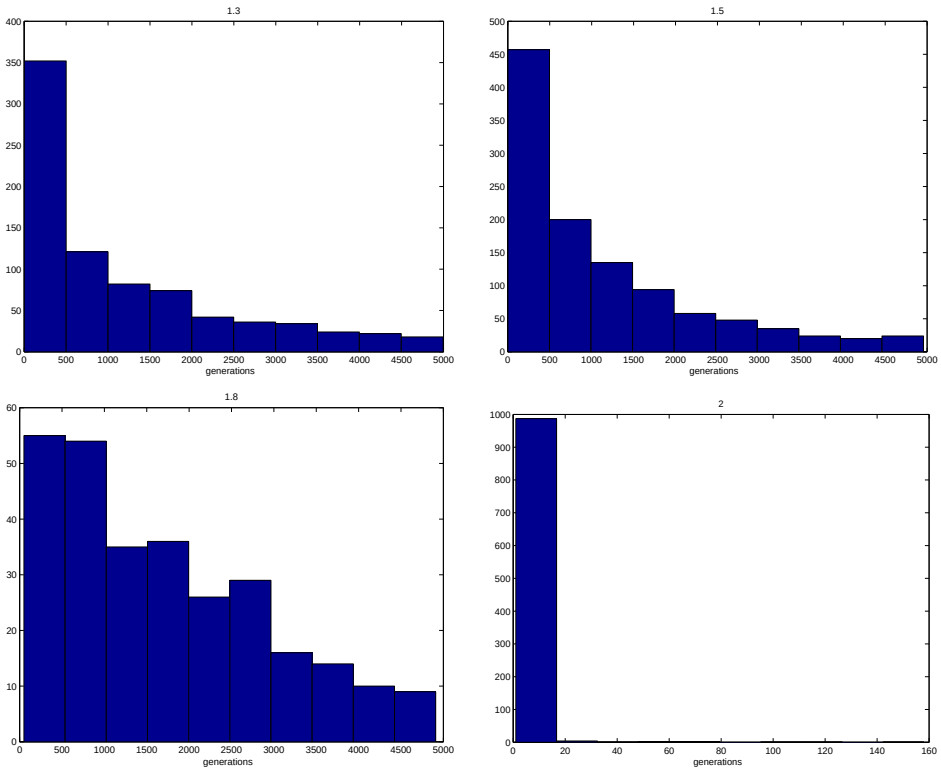

Figure 5.7: Histograms with the execution samples obtained for fractal dimensions 1.3, 1.5, 1.8 and 2.

5.1.4 Restart strategies

We have shown that our algorithm may give rise to computational efforts with a leptokurtic or heavy tail distribution. This may be due to the fact that the algorithm makes *bad choices* more frequently than expected, leading the search to a dead-end in the search space, where no solution of the required fitness exists.

The algorithm seems to be more efficient at the beginning of the search, which suggests that a sequence of short executions, compared to a single long execution, may give rise to a better use of the computational resources. In this subsection we show that the algorithm may be accelerated by the use of several *restart strategies*.

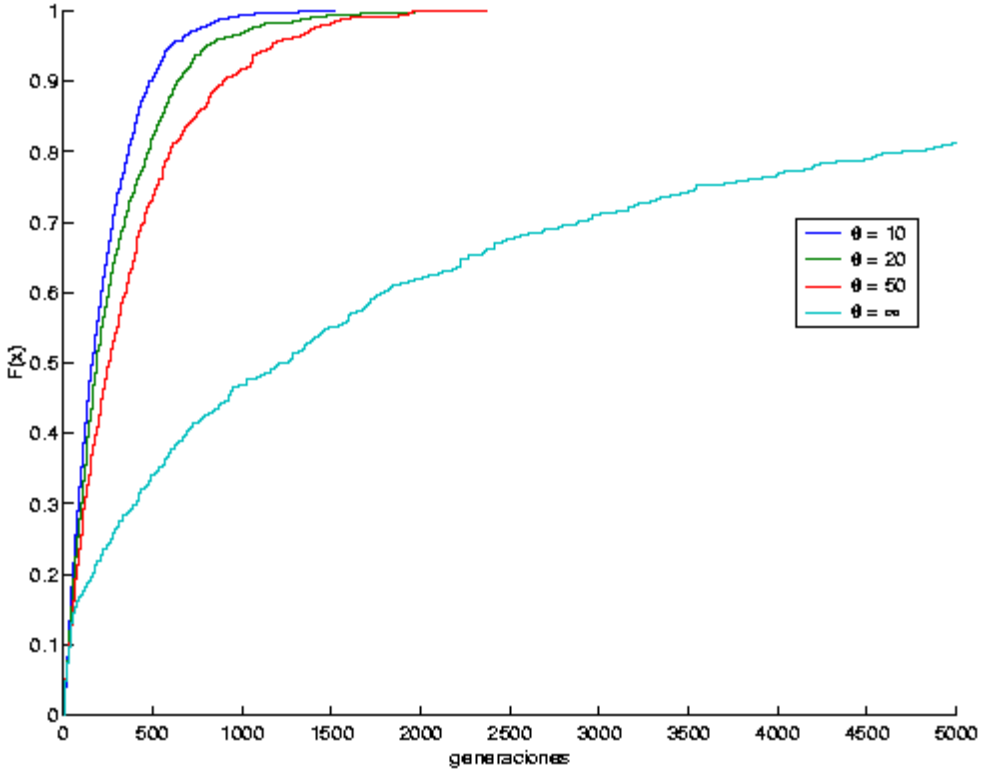

Figure 5.8: Function $F(x)$ for several values of the restart threshold $\theta \in \{10, 20, 50, \infty\}$ applied to fractal dimension 1.3.

Restarts with a fixed threshold

Figure 5.8 displays the result of a *restart strategy with a fixed threshold* applied to the generation of a fractal curve with dimension 1.3. This is the simplest strategy: once the algorithm has been working for a predefined number of generations θ, without reaching the desired goal, a new execution is started with a different random seed. As the figure shows, the failure rate after 500 generations is 70% ($F(500) = 0.3$), while this percentage falls to 10% using restarts with a threshold $\theta = 10$ generations.

Such an improvement is typical of heavy tail distributions. The fact that the experimental curve has been so dramatically moved towards the beginning of the support is a clear indication that the heavy tail character of the original distribution

has disappeared in the modified algorithm.

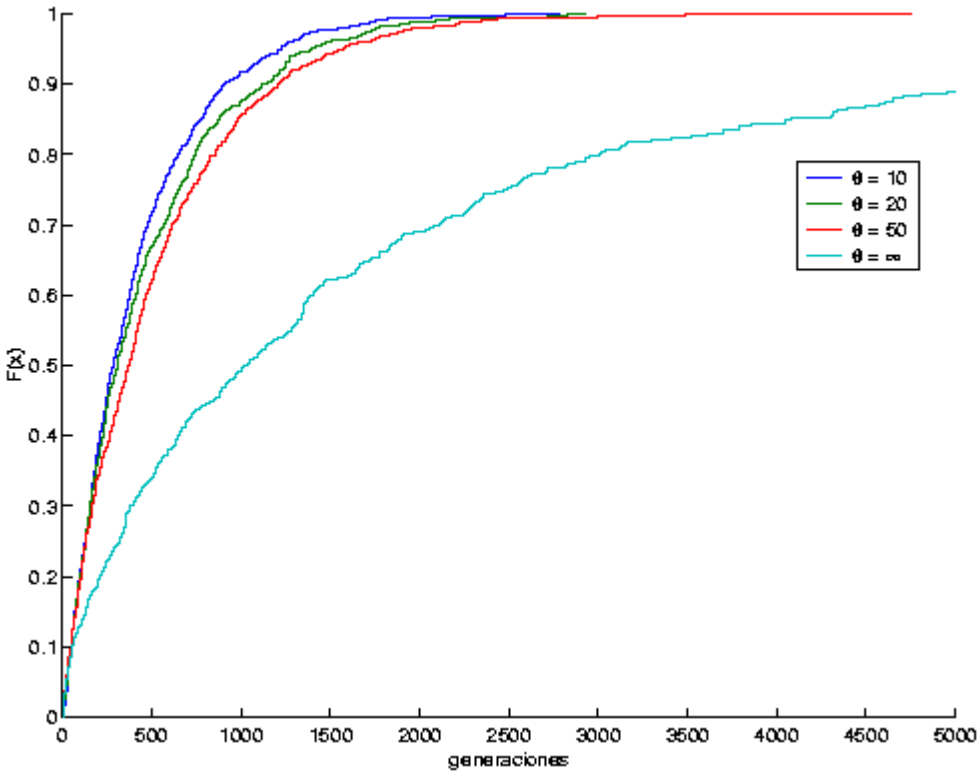

Figure 5.9: Function $F(x)$ for several values of the restart threshold $\theta \in \{10, 20, 50, \infty\}$ applied to fractal dimension 1.5.

Figures 5.9, 5.10 and 5.11 clearly show that the restarts make the tail of the distributions *lighter*, thus providing a mechanism to handle heavy tail and leptokurtic distributions.

Different fixed thresholds give rise to different average times needed to reach a solution. Table 5.6 and figure 5.12 show that the threshold value $\theta = 6$ minimizes the expected cost for fractal dimension 1.3, making it the optimal threshold θ^*. For threshold values larger than the optimal, the heavy tail behavior at the right of the median dominates the average cost, while below the optimal value the success

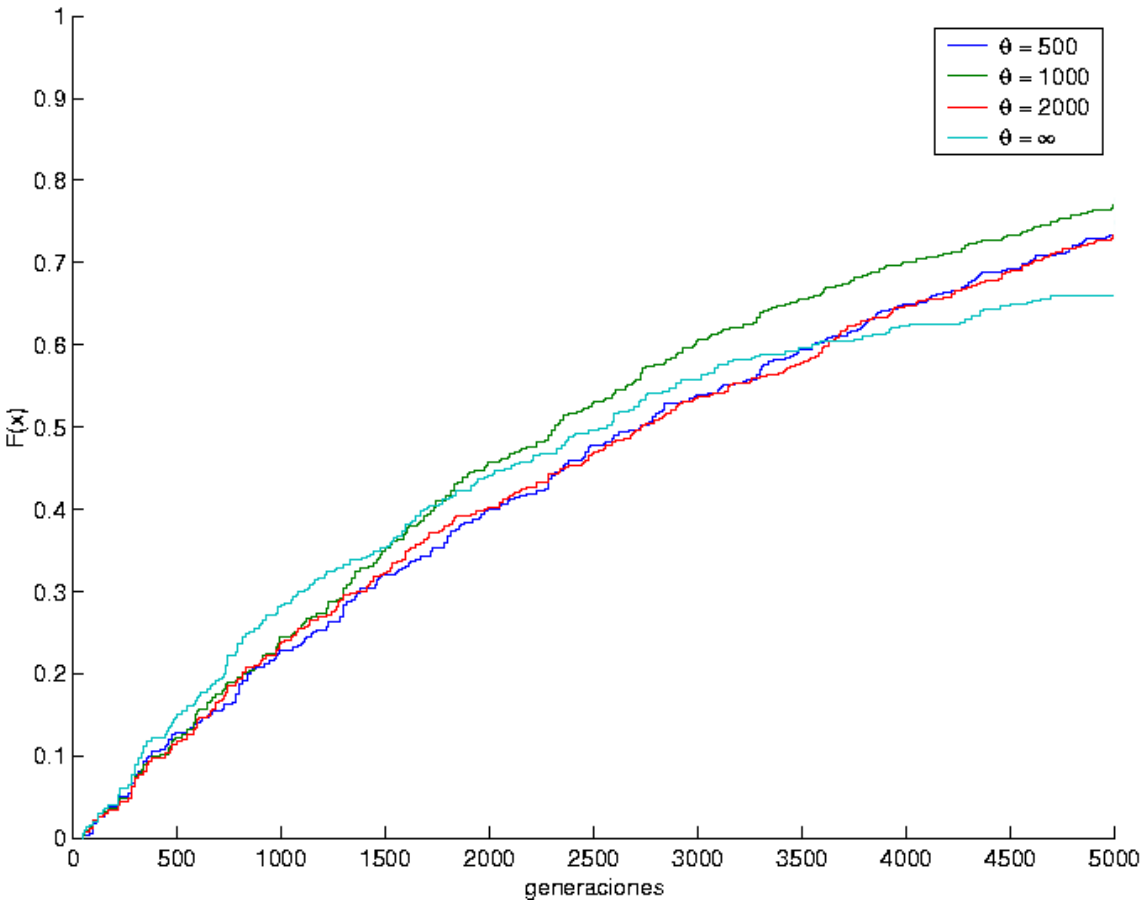

Figure 5.10: Function $F(x)$ for several values of the restart threshold $\theta \in \{500, 1000, 2000, \infty\}$ applied to fractal dimension 1.8.

percentage is too small and too many restarts are required. Anywhere, many non-optimal choices provide a considerable acceleration of the algorithm.

It has been proven that the use of a fixed restart threshold θ with a heavy tail distribution eliminates this behavior in such a way that all the moments of the new distribution become finite [74].

Restart sequences

The idea of a fixed threshold comes from theoretical results in [109], which describe optimal restart policies. It can be proven that if the time distribution of the execution

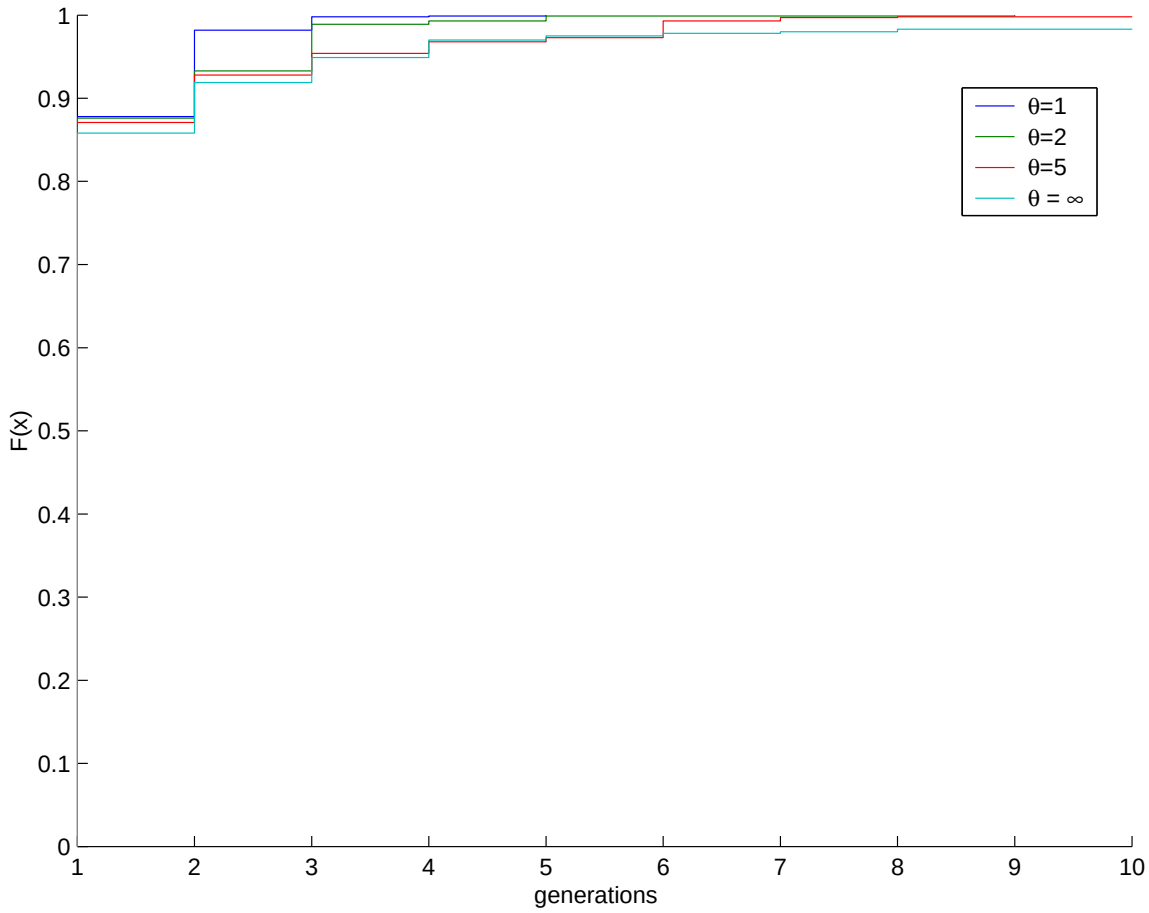

Figure 5.11: Function $F(x)$ for several values of the restart threshold $\theta \in \{1, 2, 5, \infty\}$ applied to fractal dimension 2.

is completely known and therefore θ^* can be calculated a priori, restarting every θ^* generations yields the minimum average execution time.

[109] also provide a strategy (a *universal strategy* applicable to every distribution) to minimize the expected cost of random procedures in the case where no *a priori* knowledge is available. It consists of sequences of executions whose values are powers of two. After two executions with a given threshold, the threshold is changed to its double value. Let t_i be the number of generations of the i-th execution; the universal

θ	% solved	average cost
2	100%	382.6740
4	100%	277.5730
8	100%	207.8240
16	100%	271.3980
32	100%	345.2680
64	100%	407.2460
128	100%	621.1770
256	99.8%	830.4220
512	98.5%	985
1024	96.4%	1,367
2048	93.7%	1,909

Table 5.6: Percentage solved and average cost for several threshold values in the fractal dimension 1.3 experiment.

strategy is defined as:

$$t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \leq i < 2^k - 1, \end{cases}$$

yielding strategies of the form

$$(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots).$$

[109] presents two theorems which together prove the asymptotic optimality of this procedure for an unknown distribution.

Table 5.7 summarizes the results of the application of both strategies. The average time using restarts with the universal strategy is approximately twice the time needed using fixed restarts with the optimal threshold. Both yield a considerable acceleration against the algorithm without restarts.

In several problems whose execution times had heavy tail distributions, the universal strategy was found to grow 'too slowly.' This happens because, in those problems, the restart sequence takes too many iterations to reach a value near θ^* [30, 94]. A

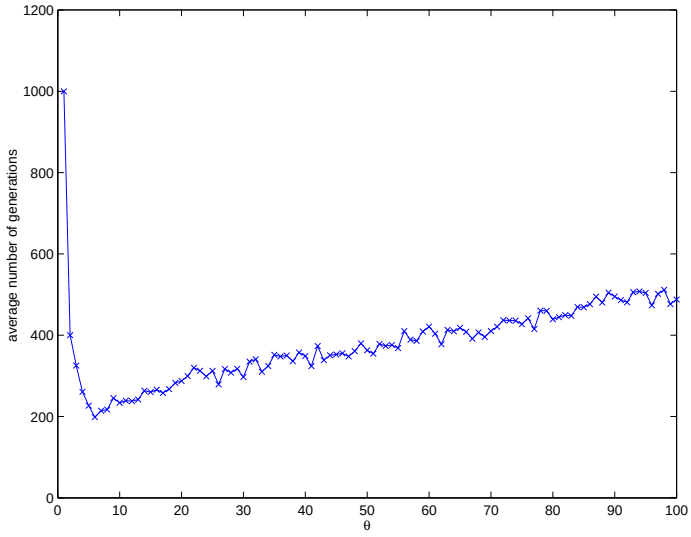

Figure 5.12: The effect of restarts with fixed θ on the solution costs for fractal dimension 1.3.

correction was proposed by [149], with a new restart strategy which was applied successfully to constraint satisfaction problems. In this simple strategy, each new restart is a constant factor γ greater than the preceding value:

$$(1, \gamma, \gamma^2, \gamma^3, \ldots), \quad \gamma > 1. \tag{5.10}$$

This strategy has a high probability of success when the restart value $t_i = \gamma^{i-1}$ is near the optimal restart threshold value. Increasing the restart threshold geometrically makes sure that the optimal value will be reached in a few generations. The

dimension	no restart	optimal fixed threshold	universal
1.3	1,173	164.9655 ($\theta^* = 6$)	294.867
1.5	1,108	374.2069 ($\theta^* = 10$)	622.181
1.8	1,443	248.5263 ($\theta^* = 17$)	625.334
2	5.4360	1.1655 ($\theta^* = 1$)	1.1701

Table 5.7: A comparison between average execution times for each dimension without restarts, with an optimal fixed threshold strategy and with the universal strategy.

solution is expected to be found within a few restarts after the value of t_i has surpassed the optimal. This strategy has the advantage of being less sensitive to the actual distribution it is applied to.

Figure 5.8 displays the average execution times using Walsh strategy for several values of γ. It can be seen that $\gamma = 1.2$ provides the fastest acceleration; with this parameter, fractal dimension 2 reaches the performance of fixed restarts with optimal threshold. The average times for fractal dimensions 1.3 and 1.5 are approximately double of those obtained with the universal strategy, although much less than those without any restart strategy. A special case is fractal dimension 1.8, where Walsh strategy worsens the performance.

5.1.5 Discussion

Heavy tail probability distributions have been used to model several real world phenomena, such as weather patterns or delays in large communication networks. In this section we have shown that these distributions may be also suitable to model the execution time of an algorithm which uses Grammatical Evolution for automatic fractal generation. Heavy tail distributions help to explain the erratic behavior of the mean and variance of this execution time and the large tails exhibited by the distribution.

We have proved that restart strategies mitigate the inconveniences associated with heavy tail distributions and yield a considerable acceleration on the previous algorithm. These strategies exploit the non-negligible probability of finding a solution in short executions, thus reducing the variance of the execution time and the possibility that the algorithm fails, which improves the overall performance.

dimension	Walsh				
	$\gamma = 1.2$	$\gamma = 1.4$	$\gamma = 1.6$	$\gamma = 1.8$	$\gamma = 2$
1.3	441.5138	639.0714	846.0743	773.4603	898.4630
1.5	654.5434	773.9020	845.0908	905.0780	938.3790
1.8	3,115	2,695	2,527	2,437	2,372
2	1.167	1.1827	1.1729	1.1979	1.1783

Table 5.8: Average execution times using the Walsh strategy for several values of γ.

We have given evidence that several restart strategies are of practical value, even in scenarios with no a priori knowledge about the probability distribution of the execution time.

So far, we have considered situations of complete or inexistent knowledge. In real situations, the execution time or the resources are bounded, so that some *partial knowledge* about the execution time is available. In this scenario, we suspect that our algorithm would take advantage of *dynamic restart strategies* based on predictive models, which have been used successfully to tackle decision and combinatorial problems [86, 94, 139]. Further research along this line would be focused on pinpointing the real time knowledge about the behavior of the algorithm which would make it possible to build predictive models for its execution time, thus providing a further acceleration.

Finding the conditions for the execution time of a particular Grammatical Evolution experiment to exhibit a heavy tail distribution would also make an interesting research line: is the fractal generation optimization exhibiting a typical behavior or is it just an exception?

5.2 Accelerated Training of Multilayer Perceptrons

The training time of a Multilayer Perceptron (MLP), understood as the time needed to obtain some required training error, is a random variable which depends on the random initialization of the MLP weights.

These weights are commonly initialized according to a given probability distribution, having this choice a significant impact on the training time distribution (see [53, 56, 101]). To address this problem, some weight initialization methods have been proposed (e.g. [56, 150]). They attempt to reduce the training time by applying different probability distributions on the initial weights of the MLP based on knowledge about the training set.

In this work, a simpler and more general approach which does not make use of the mentioned information is presented. To do this, we model the learning process of a MLP as a *las Vegas* algorithm [109], i.e. a randomized algorithm which meets

three conditions: (i) it stops when some pre-defined training error δ is obtained, (ii) its only measurable observation is the training time, and (iii) it only has either full or null knowledge about the training time probability distribution.

Using this modeling, we perform a case study with the UCI Thyroid Disease database[1], revealing that the time distribution for learning this pattern recognition benchmark belongs to the *heavy tail* distribution family.

This work adapts restart estrategies to the MLP context: the MLP is trained during a number of epochs t_1. If the required training error δ is achieved before t_1, then the execution finishes. Otherwise, we initialize again the weights in a randomized way, and re-train the MLP during t_2 epochs. The process is iteratively repeated until the training error δ is reached.

Two different strategies are applied for the determination of optimal restarting times. The first assumes full knowledge of the distribution yielding a 40% cut down in expected time with respect to the training without restarts. The second assumes null knowledge, yielding a reduction ranging from 9% to 23%.

The rest of the section is organized as follows. Subsection 5.2.1 presents the Thyroid Disease database and provides evidence of heavy tail behavior when a MLP is trained on it. Subsection 5.2.2 tests the condition to be satisfied by the probability distribution to profit from restart strategies, providing an empirical evaluation of two strategies on the particular case study. Finally, some discussion and future research lines are given in subsection 5.2.3.

5.2.1 A case study: the UCI Thyroid Disease Database

To motivate the use of restarts in MLP learning, we firstly present the existence of a high variability in its training time, indicative of an underlying heavy tail behavior. The evaluation was performed using the UCI Thyroid Disease database, as a case study.

Table 5.9 shows the expectations, deviations (and its ratio) of the numbers of epochs T spent in building a single hidden layer MLP with $n = 1, \ldots, 8$ units. The

[1]The UCI Repository of Machine Learning Databases, available online at http://www.ics.uci.edu/~mlearn/MLRepository.html

MLP was trained using the well-known Back-Propagation technique with a target training error $\delta = 0.02$. The results shown were computed using 10-fold 10-cross validation.

n	1	2	3	4	5	6	7	8
$E[T]$	8551.7	5516.8	888.5	2339.7	1680.2	587.6	482.4	490.5
$\sigma[T]$	2547.5	3885.6	1565.5	2848.8	1355.6	55.1	296.9	464.1
$\sigma[T]/E[T]$	30%	70%	156%	106%	79%	10%	60%	95%

Table 5.9: Expectation, deviation (and its ratio) of the number of epochs T spent in the building of a MLP with n hidden units and training error $\delta = 0.02$.

The obtained deviations are very large respect to the expectations for most of the architectures. For the rest of the experiments, we shall use a MLP with $n = 3$ hidden units, which has the highest relative variability. This will serve as a proof of concept, although the same behavior is observed in MLPs with other number of hidden units.

In the following, we give visual evidence that T is heavy tailed.

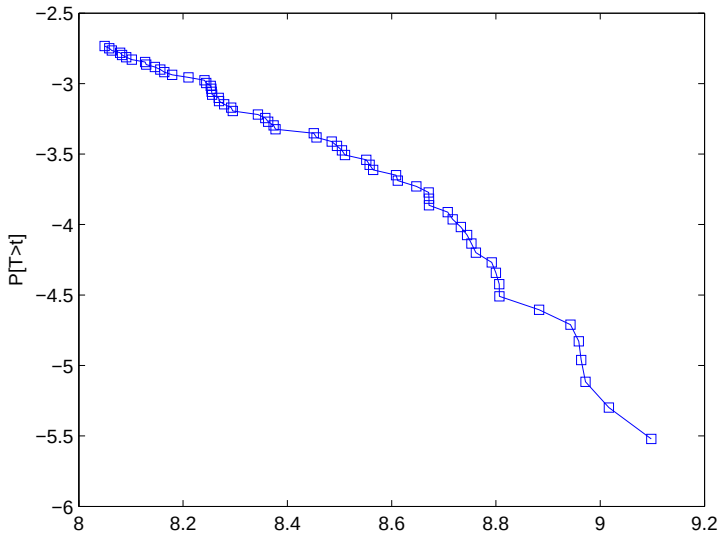

Figure 5.13: A log-log plot of $P[T > t]$ as a function of t (in epochs).

Figure 5.13 presents a log-log plot of $P[T > t]$ for the 10% largest values ($t > 3,000$). The plot confirms the polynomial decay by displaying a straight line with

slope $-\alpha$. As we informally mentioned in last chapter, for sufficiently large t, $\log P[T > t] = -\alpha \log C.t \Rightarrow \log P[T > t]/\log C.t \approx -\alpha$.

Finally, we verify that α belong to the $(0, 2)$ interval by computing the (this time non-adapted) Hill estimator (see [82]):

$$\hat{\alpha}_r = \left(r^{-1} \sum_{j=1}^{r} \ln T_{m,m-j+1} - \ln T_{m,m-r} \right),$$

where $T_{m,1} \leq T_{m,2} \leq \ldots \leq T_{m,m}$ are the m ordered training completion times, and $r < m$ is a cutoff that allows to observe only the highest values (the tail). We use the typical cutoff $r = 0.1m$ and obtain $\hat{\alpha}_r = 1.942$, which is consistent with our hypothesis.

This polynomial decay, which yields a big probability mass for long executions, is due to the fact that certain initial weights entail a convergence to local minima of the target function, requiring very long (even infinite) training periods, while others yield a convergence to global minima in a few epochs.

5.2.2 Restart strategies

A las Vegas algorithm may profit from restarting if, at some point of the execution τ, the expected completion time conditioned to the already employed execution time ($E[T-\tau|T > \tau]$) is larger than the (unconditioned) expected completion time ($E[T]$), i.e. if $\exists\tau,\ \ E[T] < E[T - \tau|T > \tau]$ (see [148]).

Figure 5.14 shows that the majority of τ values met the condition for the MLP to profit of restart strategies.

Restart strategies when the distribution is known

Luby el at. prove the existence of an optimal restart strategy for a Las Vegas algorithm which minimizes the expected running time when the execution time distribution $q(t) = \Pr(T < t)$ is assumed known (see [109]).

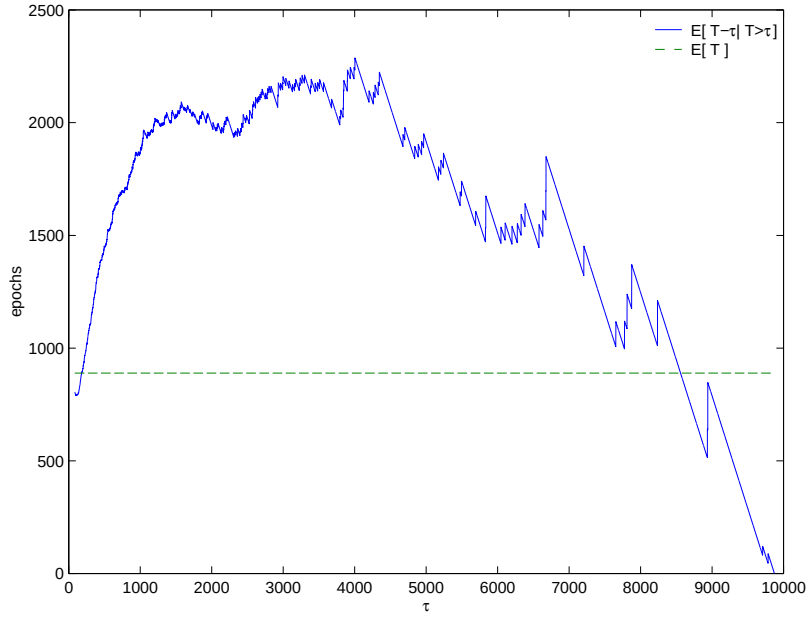

Figure 5.14: $E[T-\tau|T>\tau]$ as a function of τ, $E[T]$ serves as the baseline.

This optimal strategy is of the form $t_i = t^* \ \forall i$, where

$$t^* = \arg\min_t E[S_t] = \arg\min_t \frac{1}{q(t)}\left(t - \sum_{t'<t} q(t')\right) \tag{5.11}$$

and S_t is the restart strategy where $t_i = t \ \forall i$ for some t. This strategy is thus a *fixed strategy*, as it is presented in the previous chapter.

Simple calculations yield $t^* = 418$, with an optimal expected time $E[S_t^*] = 546.876$. This provides a 40% cut down in expected time with respect to the training without restarts (see Table 5.9). Figure 5.15 displays the expected time for strategies of the form S_t with $t \in [100,\ 10,000]$. As it can be seen, many non-optimal t choices provide a time reduction as well.

Restart strategy when the time distribution is unknown

In some scenarios it is not possible to assume full knowledge of the distribution, e.g. if the MLP is to be trained a single time. In this subsubsection we assume null

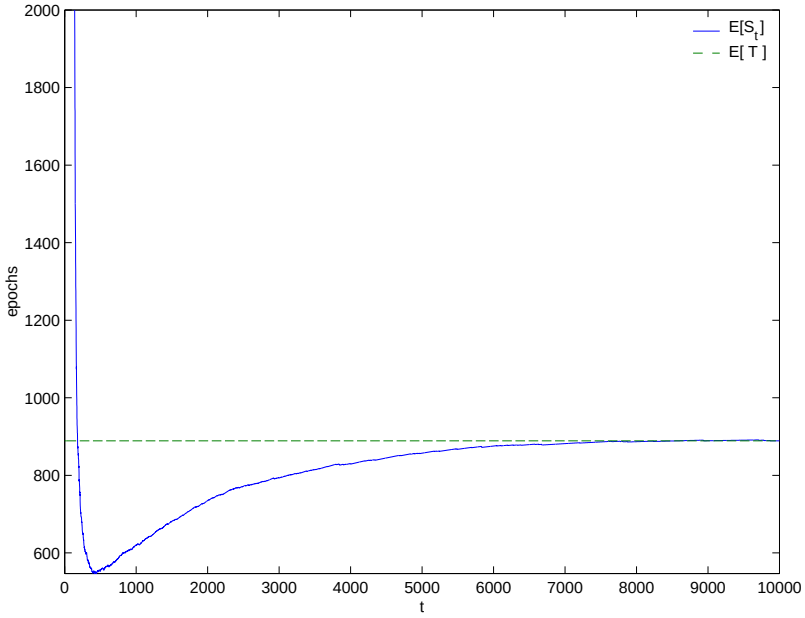

Figure 5.15: Expected training time using the strategy S_t with $t \in [100,\ 10,000]$, $E[T]$ serves as the baseline.

knowledge.

We now use the Walsh strategy S_W, described in the previous chapter.

Figure 5.16 displays the expected values of S_W using several standard γ values $\gamma = 2, 3, \ldots, 10$. Training is speeded with all choices, with improvements ranging from 9% ($\gamma = 2$) to 23% ($\gamma = 8$). The expected times were computed running $1,000$ times the training algorithm for each γ.

5.2.3 Discussion

In this work, MLP training algorithm is modelled as a Las Vegas algorithm, performing a case study on the UCI Thyroid Disease Database. We give statistical evidence that the probability distribution of the training time belongs to the heavy tail family, meaning a polynomial probability decay for long executions. This property is exploited to reduce the training time cost by two simple restart strategies. The first assumes full knowledge of the distribution yielding a 40% cut down in expected time with respect to the training without restarts. The second, assumes null knowledge,

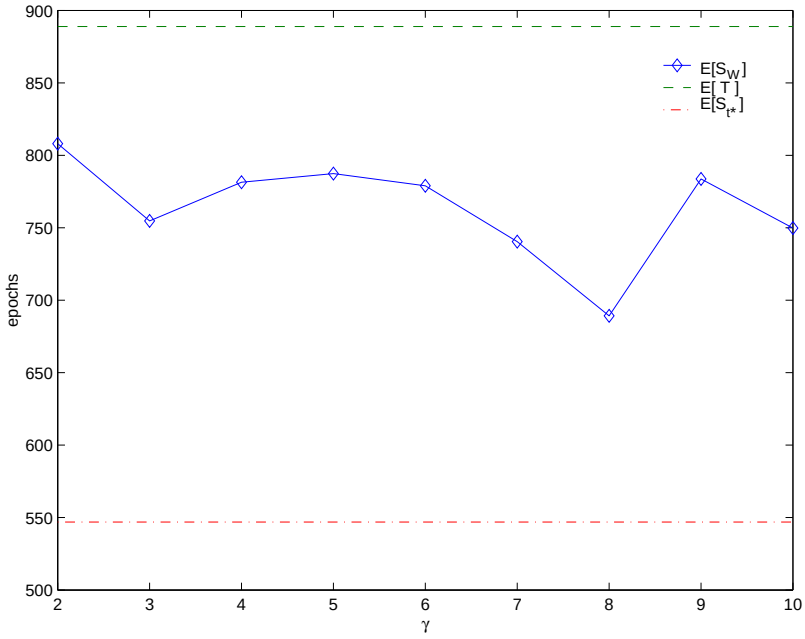

Figure 5.16: Expected training time using the Walsh strategy $E[S_W]$ for $\gamma = 1, 2, \ldots, 10$, $E[S_t^*]$ and $E[T]$ serve as baselines.

yielding a reduction ranging from 9% to 23%.

As a future research, we plan to determine whether further improvements can be obtained by relaxing the two Las Vegas algorithms assumptions (see the introducion of this section). This could make it possible to incorporate dynamic restart strategies (see [94]) capable of exploiting epoch-by-epoch information about the training time distribution, using various algorithm behavior measurements besides the execution time.

5.3 GRASP-Evolution for Constraint Satisfaction Problems

Random binary CSPs is a widely used benchmark within the constraint programming community as an efficiency test for several algorithms and solvers. However, we can also find a wide spectrum of evolutionary approaches for solving random binary CSPs.

In [42] we find a comprehensive comparison of those methods, such as SAW [41], Glass-box [114], MID, CCS, among others.

This section presents the hybrid evolutionary algorithm GA-GRASP$_{Vo}$ for solving random binary CSPs. The algorithm is conceptual, simple and uses a key modeling based on the ideas in [38]. GA-GRASP$_{Vo}$ specifically applies the idea of a GRASP-like mechanism to perform genotype-to-phenotype mapping for solving random binary CSPs.

The main difference between our algorithm and that of [38], in terms of introducing a GRASP-like genotype-to-phenotype mapping, is that the genotypes represent two completely different aspects. In [38], the genotype represents values to assign to variables, while in our algorithm the genotype represents the order in which the variables will be tentatively assigned. This is a novel approach to solve CSPs, and it will be explained in more detail in the next subsections.

We provide a comparison with two of the most successful state-of-the-art evolutionary algorithms as shown in [42]. Our simple algorithm outperforms the best approach in terms of effectiveness (measured by success rate, mean error at termination and average champion error) while outdoing it also in terms of efficiency (measured by the average number of evaluations to find a solution). Furthermore, it compares with the best approach in terms of efficiency, while outperforming it in terms of effectiveness.

The main contributions of this section are:

- A novel representation which focuses on finding an optimal variable ordering, and that borrows ideas from [38] for a GRASP-like genotype-to-phenotype mapping.

- A general evolutionary algorithm which can be easily suited to solve any kind of CSP problem without considerable implementation effort.

- Outstanding results that outperform and compare with the best evolutionary algorithms which usually involve complex heuristics and fitness adjustment functions.

- Showing that a simple algorithm can yield outstanding results if an appropriate modeling is chosen; therefore, stating the importance of representation in evolutionary strategies.

The rest of the section is organized as follows: first we briefly introduce constraint satisfaction problems for Evolutionary Algorithms (EAs). We then introduce the GRASP framework and present our hybrid algorithm. The following subsections are devoted to the experimental comparison with other methods: in section 5.3.4 we define our test-suite problem (random binary CSPs), in section 5.3.5 we introduce the methods against which our algorithm will be compared, section 5.3.6 describes the measures we use for the comparison, and section 5.3.7 shows the experimental results and the comparison itself. The section ends with some discussion and future work.

5.3.1 Constraint Satisfaction Problems and EAs

In a *constraint satisfaction problem* (CSP) we are given a set of variables, where each variable has a domain of values, and a set of constraints acting between variables. The problem consists of finding an assignment of values to variables in such a way that the restrictions imposed by the constraints are satisfied.

We can also define a CSP as a triplet $< X, D, C >$, where $X = \{x_1, \ldots, x_n\}$ is the set of variables, $D = \{D_1, \ldots, D_n\}$ is the set of nonempty domains for each variable x_i, and $C = \{C_1, \ldots, C_m\}$ is the set of constraints. Each constraint is defined over some subset of the original set of variables $\{x_1, \ldots, x_n\}$ and specifies the allowed combinations of these variable values. Thus, solving the CSP is equivalent to finding a complete assignment for the variables in X with values from their respective domain set D, such that no constraint $C_i \in C$ is violated.

The evolutionary framework presents the issue of constraint handling: constraints can either be handled directly or indirectly [57].

- **Indirect handling** involves transforming the constraint into an optimization objective which the EA will pursue; while,

- **Direct handling** leaves the constraint as it is, and enforces it somehow during the execution of the algorithm.

Direct handling is not oriented for EA due to the lack of an optimization function in the CSP, which would result in no guidance towards the objective. Thus, indirect handling is the best suited approach for EA, although a mixed strategy where some constraints are enforced and some are transformed into an optimization criteria is suited as well.

5.3.2 Greedy Randomized Adaptive Procedures

```
procedure GRASP(maxIt,seed)
  1. Read_Input()
  2. for k=1,..., maxIt do
  3.     Solution ← Greedy_Randomized_Construction(seed);
  4.     Solution ← Local_Search(Solution);
  5.     Update_Solution(Solution);
  6. end;
  7. return Best_Solution;
  end GRASP
```

Figure 5.17: The GRASP pseudocode

The GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic can be viewed as an iterative process, each iteration consisting of two phases: construction and local search [62]. The construction phase builds a solution whose neighborhood is investigated by the local search procedure. During the whole process, the best solution is updated and returned at the end of a certain number of iterations. Figure 5.17 illustrates the basic GRASP procedure.

Any local search algorithm can be incorporated to improve a solution: tabu search and simulated annealing [54, 108], large neighborhoods [13] or variable neighborhood search [118]. However, we are interested in the greedy construction phase, where a tentative solution is built in a greedy fashion.

Randomly generated solutions are usually of a poor quality, while greedy generated solutions tend to be attracted by local optima, due to the less amount of variability. A *greedy randomized heuristic* [61] adds variability to the greedy algorithm. A certain greedy function yields a ranked candidate list, which is called *restricted candidate list* (RCL). An element from that list is randomly selected and added to the solution.

procedure Greedy_Randomized_Construction(seed)
1. Solution $\leftarrow \emptyset$
2. Evaluate the incremental costs of candidate elements
3. **While** Solution is not complete **do**
4. Build the restricted candidate list RCL
5. Select element s from RCL at random
6. Solution $\leftarrow$ Solution $\cup \{s\}$;
7. Reevaluate the incremental costs;
8. **end**;
9. **return** Solution;
end Greedy_Randomized_Construction

Figure 5.18: The Greedy Randomized Construction pseudocode

The procedure to construct the *greedy randomized* solution is depicted in Figure 5.18. A key step in this pseudocode is the selection of an attribute from the RCL. This can be performed using a qualitative or quantitative criterion. In the former, the element is selected among the k best elements; while in the latter, the element is selected among the elements with a quality $\alpha\%$ of the greedy value. Note that $k = 1$ or $\alpha = 100$ yields a pure greedy selection.

Reactive GRASP

As can be seen in the procedure described below, the selection of the k parameter is problematic. The use of a fixed value for this parameter could hinder high quality solutions [128]. A learning-based strategy named *reactive* GRASP was introduced in [129], selecting a different value in each iteration from a finite set of values. The selection of a certain value in a given iteration can be chosen on the basis of the

goodness of the best solution generated by this parameter. A possibility is to maintain a vector of parameter values to use in each iteration, where a position p_i denotes the value of the parameter that serves to choose the $i-th$ candidate. From now on we will refer to this vector as *GRASP parameters vector*.

For example, a certain position of the GRASP parameters vector $p_i = 3$ makes us choose a random candidate among the four best candidates, for the $i-th$ decision, in the RCL list (from now on we will consider that the first value in the RCL is in position 0 and the last one $n-1$, where n would be the length of the RCL).

5.3.3 The Hybrid Evolutionary Algorithm

We now turn to the hybrid evolutionary algorithm for solving CSP problems. The algorithm maintains a population of GRASP parameters and performs a number of iterations until a solution is found. In each iteration it selects two individuals of the population and, with some probabilities, crosses and/or mutates them. The next population will be obtained in an elitist fashion.

Following the framework presented in [42], our algorithm consists of a generational evolutionary model with an elitist selection of the new generations, a one-point crossover recombination operator, and a mutation operator which selects, for each s_i (each single GRASP parameter in the vector), a uniformly random new value (subject to a given probability); the parent selection is performed in a binary tournament fashion and the constraint handling is purely indirect by using GRASP parameters. It presents neither fitness adjustments nor use of heuristics. Each of these characteristics is now reviewed in more detail.

Evolutionary model

The algorithm consists of a generational strategy where new populations are selected in an elitist fashion, which means that in each new generation, the population is calculated by maintaining the best individuals among the previous population and the offspring.

Fitness function

In order to calculate the fitness of a certain individual, we will take into account how many variables are in conflict with the rest. Thus, the fitness function would be as follows:

$$f(s) = \sum_{j=1}^{n} \upsilon(s, C_j), \text{where,}$$

$$\upsilon(s, C_j) = \begin{cases} 1, & \text{if } s \text{ violates at least one } c \in C_j; \\ 0, & \text{otherwise.} \end{cases}$$

Where s is a complete assignment of values to variables , C_j is the set of constraints containing variable v_j and n is the number of variables.

Note that, even though we name it *fitness function*, we want to minimize its value, since this is equivalent to reducing the number of violated variables in the problem. Indeed, if $f(s) = 0$ then the assignment s produces no violations, and, therefore, it is a solution.

Crossover

The hybrid EA uses a one-point crossover for crossing two individuals σ_1 and σ_2. It selects a random number r in $1..n$ and the child is obtained by selecting the first r genes from σ_1 and the remaining $n - r$ genes from σ_2.

Mutation

The mutation here is achieved by, with a given probability, randomly selecting a new value for each single GRASP parameter in the vector that defines the individual.

Parent selection

In each iteration the algorithm selects two individuals in the population (the parents). This is performed in a binary tournament fashion: randomly selecting two individuals from the population and choosing the best of them; the same is carried out for the second parent.

Representation

This is the most important feature in our algorithm, in fact, the rest of the characteristics are common in simple evolutionary schemes. However, the representation of the CSP is a key factor in the algorithm efficiency. In order to define our implementation, we must introduce some basic concepts.

GRASP parameters vector Our population is then, a set of GRASP parameter vectors. In [38] a Hybrid GRASP - Evolutionary algorithm for finding Golomb rulers is introduced. Our representation makes the same use of the GRASP features as in the mentioned algorithm. In the same manner, the value of each parameter defines the exact candidate to select, instead of a range for a random selection as defined in the GRASP section.

The value of each parameter reflects the decision to take in this step, forcing us to make the decision ranked in the position indicated by the value. Decisions are ranked according to some quality criteria, thus, a parameter value 0 will involve making the "best" decision. A vector with all parameters set to 0 corresponds to a plain greedy strategy.

Parameters concordance Solving a CSP usually implies assigning values to variables iteratively until either a consistent solution has been reached, or the problem has been proved to be unsolvable. Every time a variable is *instantiated* with a value, a consistency test is performed to ensure that the rest of the variables will have consistent values to be assigned to. If this test fails, the procedure will backtrack to the previous decision (to the last consistent variable instantiation) and try to assign a different value to the current variable. If the test is positive the procedure will choose a new variable to instantiate.

However, there is a crucial element on the efficiency of this solving procedure: the order in which the variables are chosen for instantiation. This is called the *variable ordering heuristic*. According to [69], the ordering heuristic for assigning variables is a key factor in quickly finding a feasible solution. Based on that, we will assume that it is possible to assign the variables in a certain order such that we will be able

to find a solution assigning values that do not generate conflicts. This can seem a fuzzy assumption, and perhaps an optimistic statement; however, it is true. If you could know beforehand the search tree of a solution, you could reproduce it by assigning a value that yields no violations (the value that appears in the solution) to every variable in the order that the search tree indicates. This will produce a valid solution.

Let us exemplify this with a toy CSP. Imagine we have three variables x, y and z. The common domain of the variables is $D = 0, 1, 2$. The variables are subject to two constraints:

$$c_0 :: x + y = 2$$
$$c_1 :: z + y = 1$$

Now we are going to try to solve the problem (finding a consistent assignment of values to variables) by instantiating the variables in lexicographic order. Very briefly, we will assign $x = 0$, then $y = 2$ to be consistent with c_0 and we would not be able to assign a value to z that satisfies c_1. However, if we would have ordered the variables y, x, z, we would have assigned $y = 0$, then $x = 2$ to satisfy c_0, and finally $z = 1$ satisfying c_1. Note that the order in which we assign the values is also important. In our approach we assume a static lexicographic ordering, and we will always assign the first value that yields no violations.

Therefore, we are going to transform the problem of finding values for the variables (approach followed in [38]) into finding an optimal ordering for the variables that will yield a feasible solution. Our vector of GRASP parameters will allow us to choose, among the ranked variables, which one we want to instantiate next. The variables will be dynamically ranked using the dom/degree ordering heuristic [69] (quality criteria), which gives more weight to variables with few available values in its domain, and that take place in a greater amount of constraints (Note that this heuristic will yield the ordering introduced in the example above where we were able to immediately find a solution). The values that the parameters can take, will fall within the range $[0, n - pos_i]$, where $pos_i \in 1..n$ is the position of the given parameter within the vector. In this case, the last parameter will always be 0, since there is just one variable left to assign. Once we have selected a variable we will instantiate it with the best value

possible (the value that yields the least amount of constraint violations). In Figure 5.19 this process is explained in 6 steps.

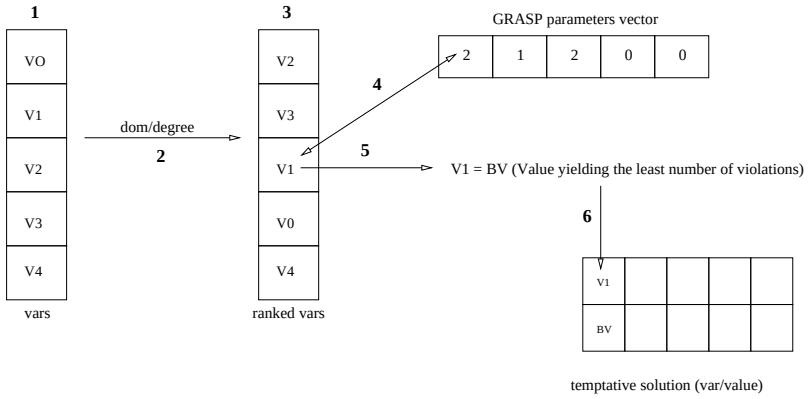

Figure 5.19: Assigning the first variable using the GRASP parameters vector in 6 steps: Step 1 shows the variables available to select. Step 2 applies the dom/degree heuristic to these variables. Step 3 shows the resultant RCL list. Step 4 selects the candidate variable that the GRASP parameters vector indicates. In Step 5 this variable is instantiated with the best value possible and the last step reflects this selection and instantiation in the first position of a vector that represents an actual tentative solution of the problem.

It is worth mentioning that, opposite to [38], we allow non-feasible instantiations. This follows immediately from the fact that we are considering a feasibility problem, instead of an optimization problem. In the latter we are searching for the best feasible solution, hence, we can restrict the search to feasible solutions; while in the former we are searching for the best unfeasible solution, which corresponds to a feasible solution–a valid solution which does have zero constraint violations.

The Hybrid Algorithm

We are now ready to present the hybrid EA GA-GRASP$_{Vo}$ which is depicted in Figure 5.20. Lines 2-4 perform the initializations. In particular, the population is randomly generated in lines 2-3 and the generation counter g is initialized in line 4.

The core of the algorithm is in lines 5-21. They generate new generations of individuals for a number of iterations or until a solution is found. The new generation is initialized in line 7, while lines 8-19 create the new generation. The new individuals

```
1.  GA-GRASP_{Vo}(csp)
2.      forall i ∈ 1..populationSize
3.          Σ ← Σ ∪ {randomConfiguration(csp.n)};
4.      g ← 0;
5.      while g ≤ maxGen  &  v(Σ) > 0 do
6.          i ← 0;
7.          Σ+ ← ∅;
8.          while i ≤ populationSize do
9.              select (σ1, σ2) ∈ Σ;
10.             with probability P_c
11.                 σ* ← crossover(σ1, σ2);
12.                 if v(σ*) == 0
13.                     return σ*;
14.             with probability P_m
15.                 σ* ← mutate(σ*);
16.                 if v(σ*) == 0
17.                     return σ*;
18.             Σ+ ← Σ+ ∪ {σ*};
19.             i ← i + +;
20.         Σ ← select(Σ+, Σ);
21.         g ← g + 1;
```

Figure 5.20: Algorithm GA-GRASP$_{Vo}$ for CSP problems.

are generated by selecting the parents in line 9, applying a crossover with probability P_c (lines 10-11), and applying a mutation with probability P_m (lines 14-15). The new individuals are added to the new population in line 18. The current population is selected among the previous and the new population in line 20. Note that after crossover and mutation we need to calculate the cost of the individual in order to detect solutions and/or keep track of the cost in order to properly select parents and next population.

5.3.4 Our benchmark: random binary CSPs

In this work we consider random binary constraint satisfaction problems, since their properties in terms of difficulty to be solved are well-understood and hence such

problems have been used for testing the performance of algorithms for solving binary CSPs. In [138] it was shown that any CSP can be equivalently transformed to a binary CSP, thus without a loss of generality.

Various problem instance generators have been developed for the class of binary CSPs, based on several theoretical models. All of these models are parameterized by n, m, D, and k, where n is the number of variables, m is the number of constraints, D is the number of values in each domain and k is the arity of each constraint. In a binary constraint network, the value of k is fixed to 2.

There are four traditional models, called A, B, C and D developed from a general framework presented in [131] and [141], all of which are unsolvable with high probability. In our work we use the E model proposed by Achioptlas *et al.* [8], which has the advantage that it generates solvable benchmarks. This model is usually specified as $E(n, p, D, k)$ with p defined as $p = m[\binom{n}{k} D^k]^{-1}$ and works by choosing uniformly, independently and with repetitions, conflicts between two values of two different variables. We are aware of the existence of another good random binary CSPs generator, Model F [40], and we plan to use it as a benchmark generator in future experiments.

Our test suit consists of 250 solvable problem instances available on the Web

http://www.cs.vu.nl/~bcraenen/resources/csps_modelE_v20_d20.tar.gz

and used also as a benchmark in [42]; they are generated using the model $E(20, p, 20, 2)$ with 25 solvable instances for each value of p in {0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.30, 0.31, 0.32, 0.33}. By using the conjecture of Smith [141] we show that the range for p in model E actually runs through the mushy region. The term mushy region is used to indicate the region where the probability that a problem is soluble changes from almost zero to almost one. Within the mushy region, problems are in general difficult to solve or to prove unsolvable. In Table 5.10 it can be seen that the predicted number of solutions drops bellow one when moving from $p = 0.31$ to $p = 0.32$, precisely what defines the mushy region.

p	E(solutions)
0.24	1707299.07
0.25	258652.614
0.26	38984.6092
0.27	5600.99655
0.28	838.870129
0.29	125.400589
0.30	19.6420135
0.31	2.79148238
0.32	0.42173145
0.33	0.06618763

Table 5.10: The Smith's conjecture prediction of the number of solutions as a function of p.

5.3.5 Related work

There are several evolutionary algorithms focused on solving random binary CSPs [42]. Most of them use knowledge of the problem, either to develop heuristics or to implement a fitness adjustment technique. The nice feature of our algorithm is that no knowledge about the problem is taken advantage of, hence, harnessing generality without a loss in efficiency or effectiveness.

In this section we are going to briefly introduce the two most successful approaches according to [42], which will be later used to compare against our algorithm.

SAW

The basic idea behind the SAW (Stepwise Adaptation of Weights) algorithm lies in the way that the fitness function is evaluated. Each k evaluations[2] the variables causing the constraint violations in the best individual of the current population are given a high weight (penalty), because they are considered to be harder than the others. These weighted-up variables will have a greater impact in the fitness of the following evaluations. A comprehensive study of different parameters and genetic operators of SAW can be found in [41].

[2]In [42] the period k is set to 25 evaluations.

Glass-box

Glass-box works by decomposing complex constraints in two steps: elimination of functional constraints and decomposition of the CSP into primitive constraints, usually of the form $\alpha \cdot p_i - \beta \cdot p_j \neq \gamma$ where p_i and p_j are the values of variables v_i and v_j. A common repair rule used is the following

$$\text{if} \quad \alpha \cdot p_i - \beta \cdot p_j = \gamma \quad \text{then change} \quad v_i \quad \text{or} \quad v_j \tag{5.12}$$

Repairing a violated constraint can result in the production of new violated constraints, thus at the end of the repairing process, the chromosome will not in general be a solution. An extensive work on this constraint processing technique is presented in [114] and [147].

5.3.6 Measures of effectiveness and efficiency

Genetic algorithms are random algorithms, therefore the behavior of the optimization in a problem instance varies from execution to execution. In order to obtain a more accurate idea of the performance of an algorithm in a concrete binary CSP instance, we are going to run it 10 times for each problem instance, thus having 250 executions for each p value (10 executions for each 25 problem instances belonging to a concrete p value). The set of executions for each p value is denoted by S_p.

An execution is finished when the genetic algorithm finds the solution or when a given number of evaluations is reached. An evaluation is the calculation of the fitness of an individual (in this case the number of variables violating a constraint).Thus, we define θ as the maximum number of evaluations for each execution. Now, we are ready to define some effectiveness and efficiency measures as a function of p.

Effectiveness

Effectiveness is measured by the success rate (SR), the mean error at termination (ME) and the average champion error (ACE).

We define S_p^+ as the executions of S_p that found a solution before θ evaluations,

and $S_p^- = S_p - S_p^+$. The SR over p is the percentage of runs that find a solution in no more than θ evaluations.

$$SR(p) = 100\frac{|S_p^+|}{|S_p|}$$

The error at termination (ET) is defined for a single run as the number of constraints violated by the best candidate solution in the population when the execution reaches θ evaluations. If an execution finishes before θ evaluations, then its ET is considered 0, thus the mean error at termination (ME) is defined as

$$ME(p) = \frac{1}{|S_p|}\sum_{s \in S_p^-} EAT(s)$$

We use another effectiveness measure that focuses on the convergence speed of the algorithm. We define the champion error (CE) as the number of constraints violated by the best individual found up to a given time (measured in evaluations) during a run, thus the average champion error is defined as

$$ACE(p,t) = \frac{1}{|S_p|}\sum_{s \in S_p} CE(s,t)$$

If s has finished before t evaluations, then $CE(s,t) = 0$.

Efficiency

In our experiments, we use the average number of evaluations to find a solution (AES) in order to measure efficiency. The AES is the average number of evaluations to find a solution (ES) over the successful runs S_p^+.

$$AES(p) = \frac{1}{|S_p^+|}\sum_{s \in S_p^+} ES(s)$$

It is important to note that if $S_p^+ \ll S_p^-$ then AES is statistically unreliable.

Another interesting efficiency measurement is the average conflict checks needed to find a solution, used in [42]. Unfortunately, it is not possible to find a correspondence

between the typical conflict check and the way in which our algorithm computes constraint violations.

5.3.7 Experimental results

We have chosen the SR, ME, ACE and AES measures in order to compare the measures obtained from our algorithm GA-GRASP$_{Vo}$ with same measures of the best ones of the algorithms analyzed in [42]. The test suite (the 250 instances generated from model E), the limit of evaluations ($\theta = 100000$)[3] and our mutation probability parameter (set to 0.3 in all the experiments) are also the same.

In the following two subsubsections we compare our algorithm with the winners of effectiveness and efficiency of the analysis performed in [42].

Comparing against the most effective algorithm: SAW

The most important measure in evaluating effectiveness is the success rate, because the main goal of an algorithm for solving CSPs is to obtain a solution. In [42] it is shown that the overall winner regarding success rates is the SAW algorithm.

In Figure 5.21 we give a comparison of the SR measures between the GA-GRASP$_{Vo}$ and SAW algorithms. SAW is outperformed by GA-GRASP$_{Vo}$ in all p values. Moreover, if we consider the global success rate for all p's, we obtain an overall SR of 55% for GA-GRASP$_{Vo}$ and 44% for SAW, which implies more than a 10% of successful executions.

Unfortunately, due to the way in which the fitness function is computed in SAW, its ME and ACE measures cannot be compared to those from GA-GRASP$_{Vo}$. This is explained because the fitness of the SAW algorithm is not the number of constraint violations, but a weights-scaled function.

We include an efficiency comparison in Figure 5.21, the average number of evaluations to a solution. In all but two values of p the GA-GRASP$_{Vo}$ requires less

[3]This limit is achieved by a population size of 1000 individuals, a maximum generations limit of 50 and two evaluations for each individual and generation: after crossover and after mutation. Similar results can be achieved with a smaller population size and with a larger amount of generations by means of a simple restarting policy. Thus, diversity is very important for the performance of the algorithm in this benchmark.

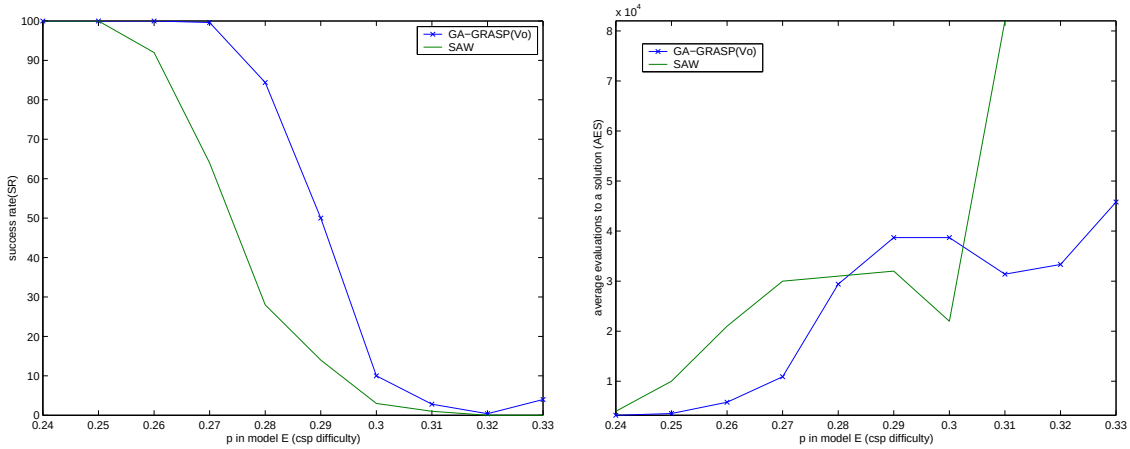

Figure 5.21: SR and AES measures for the GA-GRASP$_{Vo}$ and SAW algorithms.

evaluations than SAW. Curiously, from $p = 0.24$ to $p = 0.30$ the two algorithms seem to converge, but from that point onwards the AES for SAW growths exponentially while having only a slight increase for GA-GRASP$_{Vo}$.

Comparing against the most efficient algorithm: Glass-box

In [42] it is shown that the winner regarding Efficiency, measured by the number of fitness evaluations, is the Glass-box genetic algorithm.

In Figure 5.22 we give a comparison of the efficiency measure (AES) between the GA-GRASP$_{Vo}$ and Glass-box algorithm. In the easy region (0.24 to 0.27) GA-GRASP$_{Vo}$ needs a comparable number of evaluations, but is surpassed in terms of efficiency by Glass-Box in the mushy region. The average number of solutions over all p's is 24077 for GA-GRASP$_{Vo}$ and 7889 for Glass-box, being the last a 32% more efficient.

In efficacy terms, the two algorithms are more balanced. GA-GRASP$_{Vo}$ outdoes Glass-box in the whole easy region and in the beginning of the mushy region (0.24 to 0.31), with an equilibrium in the rest of the values. In overall terms the Glass-Box successfully finishes a 40% of the executions, a 15% less than GA-GRASP$_{Vo}$.

Observing the ME and ACE of Figure 5.22 it can be see that the quality of the partial solutions during the execution is slightly better for Glass-box, specially in the

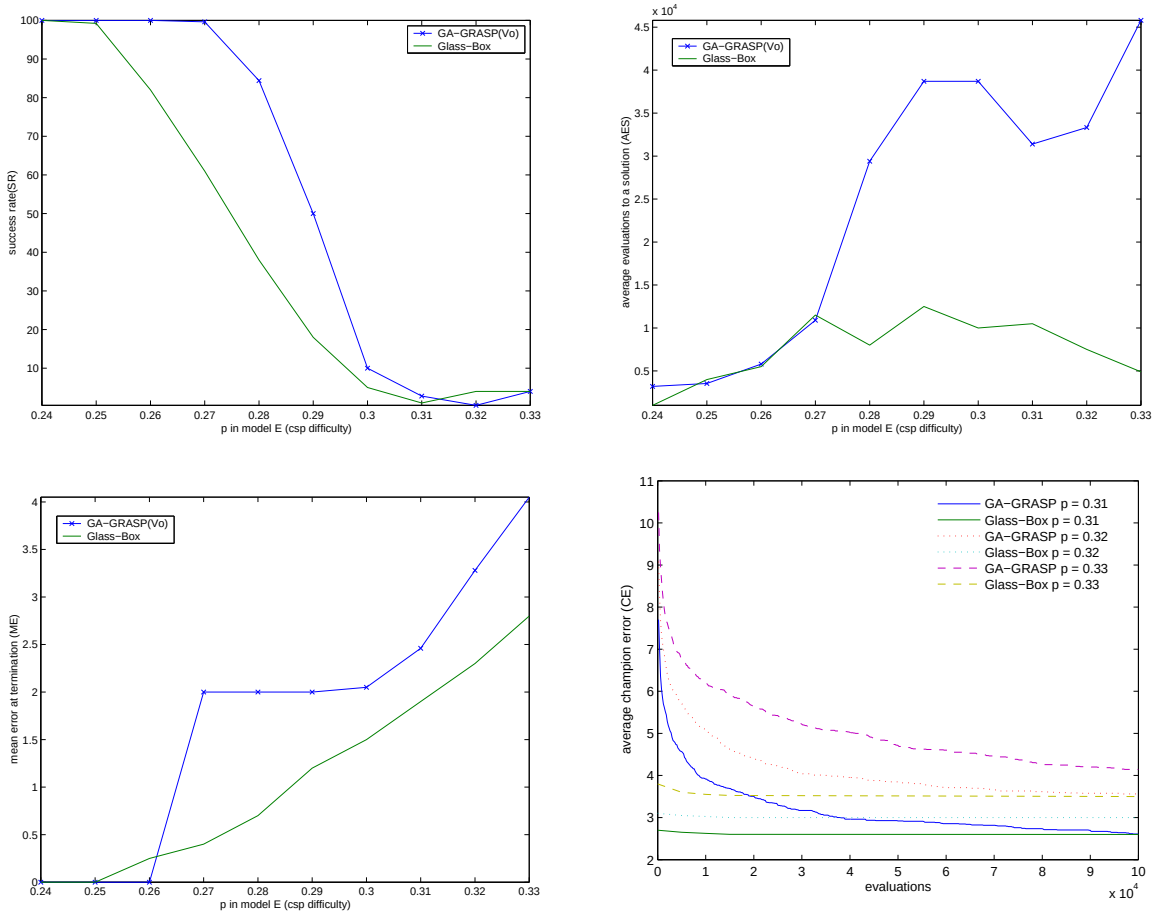

Figure 5.22: Efficacy and efficiency measures from the GA-GRASP$_{Vo}$ and Glass-Box algorithms.

mushy region, where it has 1 less violated constraint on average at the end of the run.

For all efficiency measures we have used the evaluation as the unit of computational effort. In order to obtain a more realistic picture about the efficiency of the algorithms, we have computed an average of the CPU time consumed by an evaluation: 0.0062 ± 0.0028 seconds on a Pentium IV at 2.8 GHz with 512 MBytes of RAM.

Finally, we display in figure 5.23 the average time to solution (in seconds) of GA-GRASP$_{Vo}$ for several values of p .

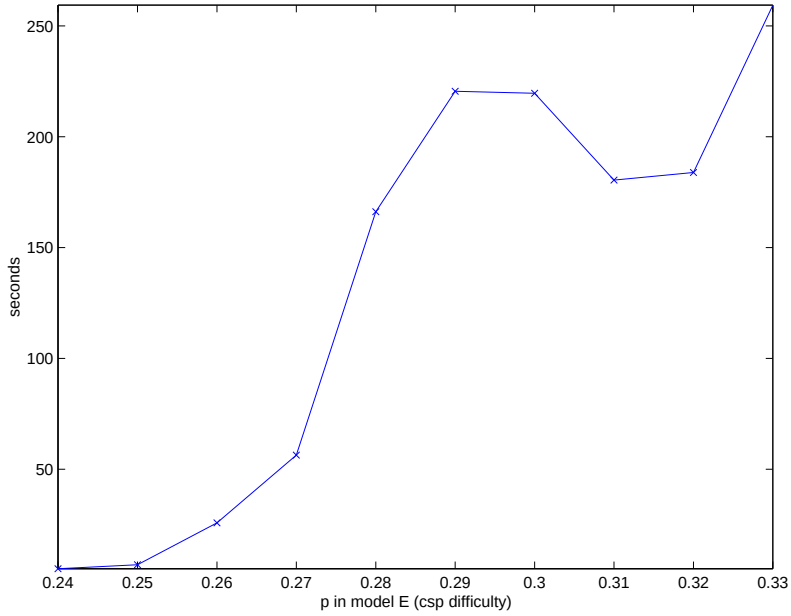

Figure 5.23: Average time to solution of the GA-GRASP$_{Vo}$ for several values of p.

5.3.8 Discussion

In this section we have presented an hybrid evolutionary algorithm for solving random binary CSPs, which yields outstanding results, as it outperforms the best previous approach in terms of effectiveness, and compares with the best strategy in terms of efficiency.

Our hybrid algorithm incorporates features of GRASP, in a similar way as in [38], where a GRASP-like mechanism is applied to genotype-to-phenotype mapping. However, our approach features a novel representation which focus on finding a variable ordering instead of a value to variable assignment.

The rest of the algorithm is conceptual and simple, making no use of information regarding the problem, which harnesses generality. It also demonstrates that modeling (or representation) is a key factor in evolutionary strategies.

Moreover, we believe there is a large space for improvement. Learning techniques and restart policies should be introduced and tested. We are also studying hybridizations with local search techniques that are already yielding very promising results.

Finally, we are interested in using real life CSP benchmarks in order to compare

results with constraint programming techniques and other evolutionary approaches available. Binary CSPs are not very common in real life applications and even though any CSP can be transformed into a binary CSP in polynomial time [138], we plan to generalize our solver to deal with n-ary CSPs.

Chapter 6

Conclusions and future work

Throughout this thesis we have adressed questions of theoretical nature in Algorithmic Information Theory, and developed new or improved applications for Classification and Evolutionary Computation, which borrow the ideas from this theory and from Algorithmic Stochastic Modeling. This chapter gives a summary of the conclusions and research lines proposed in each one.

6.1 Advances in Algorithmic Information Theory

One of the most interesting advances of Algorithmic Information Theory is the development of an absolute measure of similarity between objects. This measure can only be estimated, as it is incomputable by definition. The typical estimation relies on the use of data compression algorithms, being this estimation known as the compression distance. The two theoretical contributions of chapter 3 analyze the quality of this estimation. The first quantifies the estimation robustness when the information contained in the objects is noise-altered, concluding that it is considerably resistant to noise. The second studies the impact of the compression algorithm implementation on the estimation, yielding some practical recipes for making this choice.

6.1.1 The Normalized Compression Distance in the presence of noise

When the Normalized Compression Distance (NCD) is used to compute the distance between two different files, the second file can be considered as a noisy version of the first. Therefore, the effect on the NCD of the progressive introduction of noise in a file can provide information about the measure itself. In this work, we forward a theoretical reasoning of the expected effect of noise introduction, which explains why the NCD can get values greater than 1 in some cases.

A first batch of our experiments confirm the theoretical model. A second batch explores the effects of noise on the precision of clusterings based on the use of the NCD. It can be noticed that the clustering process is qualitatively resistant to noise, for the results do not change much with quite large amounts of it. Different types of files are differently affected, however, which is not surprising: mtDNA files, for instance, which are built on a 4-letter alphabet, are degraded faster than human text, which uses a larger alphabet.

In the future, we intend to tackle a quantitative demonstration of the NCD resistance to noise. We shall also try other metrics and clustering procedures, appropriate to the different file types, to compare their resistance to noise with our NCD results.

6.1.2 Compressors requirements for the use of the Normalized Compression Distance

We have analyzed the impact on the NCD quality of some features of two compressors: the block size in bzip2, and the sizes of the two windows (sliding and lookahead) used by gzip. The well-known Calgary Corpus has been used as a benchmark. Any similarity distance should measure a 0 distance (or, at least, a very small value) between two identical objects. The empirical results obtained with both compressors for the Calgary Corpus reveal that the NCD is biased by the size of the objects, independently of their type. For object sizes smaller than certain values (related to the block and window sizes in the compressors), the distance between two identical objects is usually quite small, which proves that the NCD is a good tool for this

purpose. However, for larger sizes, when the inner limitations of the compressors are violated, obviously the distance between two identical objects grows to very high values, making the NCD practically unusable. Other widely used compressors (such as winzip and pkzip) also show the same limitations.

The use of block and window sizes in the compressors aims to increasing the computation speed at the expense of the compression ratio. Our experiments prove that this balance between quality and speed should be treated carefully for clustering, where quality is tantamount. When considering clustering problems, all considerations about speed should be left apart if they imply exceeding the system parameters. The proper use of this powerful distance depends on selecting compression algorithms without limiting factors related to the size of the objects, such as the high compression Markov predictive coder PPMZ [91], which does not set any window or block limit, but is much slower than those mentioned above. The results of using PPMZ in our experiments are shown in Figure 3.18 (page 58 and are coherent with our conclusions: the distance computed with PPMZ does not depend on the size of the objects and is always between zero and a very small value (0.1043). On the other hand, this also confirms that the NCD is a very good distance measurement, when used in the proper way.

In the case of bzip2 and gzip, the block, the sliding window and the lookahead window should be at least as large as the sum of the sizes of the objects to be compared. The table in Figure 3.3 (page 57) summarizes the results obtained for all three compressors under different circumstances, both as regards the compression ratio obtained and the size limits where the use of the NCD is acceptable for each.

6.2 New applications of Algorithmic Information Theory

In chapter 4 we used variants of the compression distance to develop two applications for classification and one for evolutionary computation. The first application addresses the problem of detecting similarities in objects which have been generated

by a predecessor common source, independently of whether they use or not the same coding scheme: this includes detecting document translation and reconstructing phylogenetic threes from genetic material. We make use of the already proved usefulness of compression based similarity distances for educational plagiarism detection to develop our second application: AC, an integrated source code plagiarism detection environment. The third application makes use of this distance as a fitness function, which is used by evolutionary algorithms to automatically generate music in a given pre-defined style.

6.2.1 Common Source Data Detection

The experimental data confirm the hypothesis that by using Lempel-Ziv inspired structures, we can detect similarities in the texts even if the alphabet is different and we can also detect the type of the similarities, namely if the similarities are from common concepts with well-defined structure or from common data predecessor with poorly expressed or missing structure.

The measure M_1 (page 67) works reliably in texts larger than 10 KB. However, when using shorter texts, of length inferior to 5 KB we can not achieve good results. It would be interesting to find a representation of short texts that can give reliable similarity measure. As a first attempt the measure is good enough. Further consideration shows that measures, based on dynamical programming are more precise.

The comparison of correlations in order to judge the similarity between text as the only similarity measure is of course, prone of errors. First of all we must choose the parameters L and B in a way that allows as to have sufficient, but not too many arcs of the graphs within one bin. In practice, we are looking for some 5-15 arcs in 500 bins. But this implies that having some reasonable L for texts (about the length of one word, e.g. 4-8 symbols), we need significant length of the text. Actually this is observed also empirically. The method works well with text of length 5-15 KB, which is the range we have proved experimentally. It may be supposed to work with larger texts as well.

To avoid this limited text length range, one can use much more sophisticated

methods, based in general on statistical physics conformation analysis, that is beyond the scope of this thesis and subject of ongoing work.

The challenging question is whether the grammar can also be captured using similar methods. The grammar is an element may be in use when a text is compressed. As an example, let us consider some arbitrary Spanish phrase, for example "las chicas altas son buenas bailarinas" (the tall girls are good dancers). The gender/plural information is carried by the ending of the word "-as" and the string "as" should be coded according to this.

6.2.2 Souce Code Plagiarim Dectection

The problem of plagiarism detection is a difficult one. The frontier between, on one side, random similarity or simple inspiration from anothers' work and, on the other side, blind cut+paste plagiarism is not clear-cut, and certain cases will always require a human grader to distinguish between what is acceptable and what is not. However, different algorithms and heuristics can be used to identify suspects of blatant plagiarism and flag the more complex cases, greatly simplifying the grader's task.

This thesis has presented AC, a plagiarism detection tool which also doubles as a framework for research into source code plagiarism detection. Even though AC was initially designed as an enviroment to increase the NCD applicability to plagiarism detection, it has suffered a great development, currently offering many improvements over other tools described in current literature: the use of rich visualization greatly simplifies the task of analyzing the result of similarity tests; its stand-alone, cross-platform implementation does not raise privacy concerns found in web-based systems; and preparation of assignment submissions for automated plagiarism detection, overlooked by many systems, can be automated with a graphical user interface.

The use of statistical methods has opened different lines for further research. Greater insight into the *Hampel identifier* threshold choice could be obtained from a student controlled experiment or by further work on an ongoing project where assignments corpora are artificially developed [29]. A second line of research is concerned

with the surprisingly high accuracy of the normal distribution in outlier identification, even when confronted with different corpora and different similarity distance algorithms.

Once two assignments have been deemed to be "very similar" to each other, a human grader is currently expected to visually compare both for evidence of plagiarism. In other systems, similar fragments from both assignments are highlighted for side-by-side inspection, but AC currently lacks this feature. The *Substring* similarity distance test is a good candidate to identify such areas. Further refinement of the test itself and an extension to facilitate identification of *hot* areas during visual inspection is pending.

Although source code plagiarism detection is a relatively veteran field of research, few systems have undergone experimental validation. A typical approximation is to use a corpus of already-graded set of assignments (where cases of plagiarism were manually identified) and compare these cases to those found using the tool to be tested. However, both the original grader and the tool may fail to correctly identify all cases of plagiarism; and false positives are also possible. A better approach would require asking a group of students to write plagiarized versions of randomly selected assignments within a carefully selected corpus where no plagiarism had occurred. This would allow experiments to be performed with a fully annotated corpus. An alternative and less labor-intensive approach is to use automatic programming to generate artificial sets of assignments; initial steps using this method have been described in [29]. Whatever the approach, validation will continue to be one of the main lines of inquiry.

6.2.3 Music Generation

We have found that that the NCD is a promising fitness function for genetic algorithms used in automatic music generation. Some of the pieces of music thus generated recall the style of well-known authors, in spite of the fact that the fitness function only takes into account the relative pitch envelope. Our results have been qualitatively superior to those obtained previously with a different fitness function [124].

Several recombination operators have been tested to fine tune the genetic algorithm for this application, finding that mixed strategies which promotes diversity in the first generations and then change to a more exploitative strategy give the best results. This scheme of initial exploration and posterior exploitation is analogue to the idea behind Simulated Annealing [95].

In the future we intend to combine our results with those of other authors [104, 35] to use as the target for the genetic algorithm, not one or two pieces of music by a given author, but a cluster of pieces by the same author, thus trying to capture the style in a more general way.

Although we have introduced the information about note duration in the genetic process, we have ignored it so far. As the current design of our algorithm facilitates this (the NCD can easily deal with integers representing note lengths) we intend perform a new set of experiments to evolve the note length information along with the melody.

We shall also work with a standard and richer system of music representation, such as MIDI.

The results presented in this thesis serve as a proof-of-concept. As future research, we plan to provide a comparison with state-of-the-art music composition techniques from machine learning to reveal both the strengths and weaknesses of our proposal.

6.3 New applications of Algorithmic Stochastic Modelling

Another three new applications were derived in chapter 5, by means of Stochastic Modeling, two for evolutionary computation and one for classification. Two of them are intimately related and make use of the presence of Heavy Tail probability distributions in the optimization processes involved in the generation of fractals by an evolutionary algorithm, and in the training process of a multilayer perceptron. This discovery is used to improve the performance of both algorithms by means of restart strategies. The last application presented in this thesis is a successful story of the use

of a special randomized heuristic in a simple genetic algorithm to yield a state-of-the art evolutionary algorithm for solving Constraint Satisfaction Problems.

6.3.1 Accelerated Generation of Fractals of a Given Dimension

Heavy tail probability distributions have been used to model several real world phenomena, such as weather patterns or delays in large communication networks. In this thesis we have shown that these distributions may be also suitable to model the execution time of an algorithm which uses Grammatical Evolution for automatic fractal generation. Heavy tail distributions help to explain the erratic behavior of the mean and variance of this execution time and the large tails exhibited by the distribution.

We have proved that restart strategies mitigate the inconveniences associated with heavy tail distributions and yield a considerable acceleration on the previous algorithm. These strategies exploit the non-negligible probability of finding a solution in short executions, thus reducing the variance of the execution time and the possibility that the algorithm fails, which improves the overall performance.

We have given evidence that several restart strategies are of practical value, even in scenarios with no a priori knowledge about the probability distribution of the execution time.

So far, we have considered situations of complete or inexistent knowledge. In real situations, the execution time or the resources are bounded, so that some *partial knowledge* about the execution time is available. In this scenario, we suspect that our algorithm would take advantage of *dynamic restart strategies* based on predictive models, which have been used successfully to tackle decision and combinatorial problems [86, 94, 139]. Further research along this line would be focused on pinpointing the real time knowledge about the behavior of the algorithm which would make it possible to build predictive models for its execution time, thus providing a further acceleration.

Finding the conditions for the execution time of a particular Grammatical Evolution experiment to exhibit a heavy tail distribution would also make an interesting

research line: is the fractal generation optimization exhibiting a typical behavior or is it just an exception?

6.3.2 Accelerated Training of Multilayer Perceptrons

In this thesis, MLP training algorithm is modelled as a Las Vegas algorithm, performing a case study on the UCI Thyroid Disease Database. We give statistical evidence that the probability distribution of the training time belongs to the heavy tail family, meaning a polynomial probability decay for long executions. This property is exploited to reduce the training time cost by two simple restart strategies. The first assumes full knowledge of the distribution yielding a 40% cut down in expected time with respect to the training without restarts. The second, assumes null knowledge, yielding a reduction ranging from 9% to 23%.

As a future research, we plan to determine whether further improvements can be obtained by relaxing the two Las Vegas algorithms assumptions (see sect. 5.2). This could make it possible to incorporate dynamic restart strategies (see [94]) capable of exploiting epoch-by-epoch information about the training time distribution by using various algorithm behavior measurements besides the execution time.

6.3.3 GRASP-Evolution for Constraint Satisfaction Problems

In this thesis we have presented an hybrid evolutionary algorithm for solving random binary CSPs, which yields outstanding results, as it outperforms the best previous approach in terms of effectiveness, and compares with the best strategy in terms of efficiency.

Our hybrid algorithm incorporates features of GRASP, in a similar way as in [38], where a GRASP-like mechanism is applied to genotype-to-phenotype mapping. However, our approach features a novel representation which focus on finding a variable ordering instead of a value to variable assignment.

The rest of the algorithm is conceptual and simple, making no use of information regarding the problem, which harnesses generality. It also demonstrates that modeling (or representation) is a key factor in evolutionary strategies.

Moreover, we believe there is a large space for improvement. Learning techniques and restart policies should be introduced and tested. We are also studying hybridizations with local search techniques that are already yielding very promising results.

Finally, we are interested in using real life CSP benchmarks in order to compare results with constraint programming techniques and other evolutionary approaches available. Binary CSPs are not very common in real life applications and even though any CSP can be transformed into a binary CSP in polynomial time [138], we plan to generalize our solver to deal with n-ary CSPs.

Chapter 7

Conclusiones y trabajo futuro

A lo largo de esta tesis, hemos tratado cuestiones de naturaleza teórica en Teoría de la Información Algorítmica, y desarrollado aplicaciones novedosas (o que suponen una mejora a las ya existentes) para problemas de Clasificación y Computación Evolutiva, usando ideas de esta teoría y del Modelado Estocástico Algorítmico. Este capítulo resume las conclusiones y lineas futuras propuestas para cada avance.

7.1 Avances en Teoría de la Información Algorítmica

Una de las más interesantes aportaciones de la Teoría de Información Algorítmica es el desarrollo de una medida absoluta de similitud entre objetos. Esta medida sólo puede ser estimada, al ser no computable por definición. La estimación típica se basa en el uso de algoritmos de compresión de datos, siendo esta estimación conocida como la distancia de compresión. Las dos aportaciones teóricas presentadas en el capítulo 3 analizan la calidad de esta estimación. La primera cuantifica la robustez de la estimación cuando la información contenida en los objetos ha sido alterada por ruido externo, concluyendo que ésta es considerablemente resistente al mismo. La segunda, estudia el impacto de la implementación del algoritmo de compresión sobre la estimación, obteniéndose algunas recetas prácticas para realizar dicha elección.

7.1.1 La Distancia de Compresión Normalizada en presencia de ruido

Cuando usamos la distancia de compresión normalizada (NCD) para calcular la distancia entre dos ficheros diferentes, podemos considerar el segundo fichero como una versión ruidosa del primero. Por tanto, el efecto que la adición progresiva de ruido en un fichero puede tener sobre la NCD puede darnos información sobre el comportamiento mismo de esta medida. En este trabajo, damos un razonamiento teórico del efecto esperado de esta introducción de ruido, que logra explicar por qué la NCD da valores mayores que 1 en algunos casos.

Un primer conjunto de nuestros experimentos confirma el modelo teórico propuesto. Un segundo conjunto explora los efectos que el ruido tiene sobre el *clustering* basado en NCD. Es posible concluir que los procesos de *clustering* son cualitativamente resistentes al ruido, ya que no se aprecia que los *cluster* resultantes tengan cambios sustanciales respecto al *cluster* sin ruido, incluso cuando se añade un gran nivel del mismo. Sin embargo, distintos tipo de datos se ven afectados de forma diferente por el ruido, lo cual no es sorprendente: el ADN mitocondrial, por ejemplo, que se construye sobre un alfabeto de 4 letras, se degrada más rápido (con respecto a la NCD) que los textos escritos por humanos, con un alfabeto mucho mayor.

Como trabajo futuro, planeamos dar una demostración cuantitativa de resistencia al ruido de la NCD. También debemos probar otras métricas y procedimientos de *clustering* específicos para cada tipo de fichero, y comparar su resistencia al ruido con nuestros resultados sobre la NCD.

7.1.2 Requisitos de los compresores para el uso de la Distancia de Compresion Normalizada

Hemos analizado el impacto sobre la precisión de la NCD de dos aspectos de dos compresores reales: el tamaño de bloque en bzip2 y los tamaños de las dos ventanas (*sliding* y *lookahead*) de gzip. El bien conocido *Calgary Corpus* se ha usado como *benchmark*. Usamos como referencia de precisión la siguiente propiedad: cualquier medida de similitud debe dar una distancia cercana a 0 entre dos objetos idénticos.

Los resultados empíricos que hemos obtenido muestran que la NCD tiene un sesgo con el tamaño de los objetos respecto a dicha propiedad, independientemente del tipo de datos que éstos contengan: para tamaños de objetos menores que ciertos valores (relacionados éstos con los tamaños de bloque y ventana de los compresores), la distancia entre dos objetos idénticos es normalmente baja, lo que prueba que la NCD es una buena herramienta para similitud en estas condiciones; sin embargo, para tamaños moderadamente grandes de ficheros, la distancia entre dos ficheros idénticos crece hasta valores muy altos (cercanos a 1, la máxima disimilitud), provocando que el valor devuelto por la NCD carezca de valor informativo. Otros compresores muy usados (como winzip y pkzip) muestran las misma problemática.

La implementación de compresores con limitaciones (e.g. de bloque y ventana) tiene como objetivo aumentar el rendimiento de dichos algoritmos, a expensas de una posible pérdida en la capacidad de compresión. Nuestros experimentos prueban que este compromiso entre calidad y velocidad debe tratarse con cuidado si estos compresores se usan en aplicaciones de *clustering*, donde la precisión es vital. En estos casos, toda consideración sobre rapidez de compresión deben dejarse de lado, si tal cosa supone una limitación sobre los ficheros involucrados en el *clustering*. El uso apropiado de la distancia de compresión depende fuertemente de saber seleccionar compresores sin factores limitantes sobre, al menos, el tamaño de los ficheros. Un ejemplo de dichos compresores es es el codificador predictivo marcoviano PPMZ [91], que no establece ningún limite de tipo bloque o ventana, a costa de obtener una velocidad de compresión considerablemente menor que bzip2 y gzip. Los resultados del uso de PMMZ sobre nuestro conjunto experimental se muestran en la figura 3.18 (página 58), siendo estos coherentes con nuestras conclusiones: distancias entre objetos idénticos calculadas usando este compresor siempre son menores que un valor próximo a 0 (0.1043). Estos exprimentos también confirman que la NCD es una distancia robusta si se usa de forma adecuada.

Si se desea usar compresores más rápidos como bzip2 y gzip, el tamaño de bloque, así como el de la *sliding* y *lookahead* window, debe ser tan grandes como la suma de los tamaños de los objetos a comparar. La tabla en la figura 3.3 (página 57) resume las regiones aceptables de funcionamiento usando distintas opciones con los

tres compresores, así como el ratio de compresión obtenido en cada región.

7.2 Nuevas aplicaciones de la Teoría de Información Algorítmica

En el capítulo 4 usamos variantes de la distancia de compresión para desarrollar dos aplicaciones para clasificación y una para computación evolutiva. La primera aplicación considera el problema de la detección de similitudes entre documentos que han sido generados por una fuente común predecesora, independientemente de si estos usan o no la misma codificación: esto incluye la detección de traducciones de documentos y la reconstrucción de árboles filogenéticos a partir de material genético. Hacemos uso de la ya demostrada utilidad de las distancias de similitud basadas en compresión en la detección de plagio (en el ámbito educacional) para desarrollar nuestra segunda aplicación: AC, un entorno integrado de detección de plagio en código fuente. La tercera aplicación hace uso de esta distancia como una función de fitness, que es usada por algoritmos evolutivos para generar de forma automática música con un estilo predefinido.

7.2.1 Detección de Información Proveniente de una Fuente Común

Los datos experimentales confirman la hipótesis de que es posible la dección de similitudes en información textual usando estructuras inspiradas en el algoritmo Lempel-Ziv, incluso si sus alfabetos son diferentes. También es posible detectar el tipo de similitud, esto es, si las similitudes vienen de conceptos comunes subyacentes con una estructura bien definida, o si estos vienen de una fuente de información común predecesora con una estructura pobremente expresada o incluso inexistente.

La medida M_1 (página 67) funciona fiablemente bien en textos mayores de 10 KB. Sin embargo, cuando se usan textos más cortos, de longitud inferior a 5 KB, no logramos obtener buenos resultados. Sería interesante encontrar una representación de estos textos cortos que pueda dar una medida de similitud igualmente fiable a

cuando son suficientemente largos. En cualquier caso, consideramos que nuestra medida sirve como una primera aproximación al problema. Un análisis más fino, todavía por confirmar, muestra que medidas relacionadas con la nuestra aumentadas con programación dinámica podrían ser más precisas.

La comparación de correlaciones para juzgar la similitud entre información textual tiene la ventaja de ser simple, pero precisamente por ello, puede estar sujeta a errores si no se hacen las elecciones deseadas. Primeramente, debemos elegir los parámetros L y B de forma que tengamos suficientes, (pero no demasiadas) aristas en cada *bin*. En la práctica, esto se traduce en tener de 5 a 15 aristas en unos 500 bins. Eso implica que para encontrar algún L razonable para textos (como por ejemplo L igual a la longitud típica de las palabras en el texto, e.g. de 4 a 8 símbolos para la mayoría de los idiomas) necesitamos que la longitud de estos sea significativamente grande. Es posible verificar estos requerimientos de forma empírica, encontrando que este método funciona bien para textos cuya longitud está entre 5 y 15 KB, que es rango que hemos manejado en los experimentos. Es de prever que tambin funcione con textos más largos.

Con el fin de evitar esta limitación en el rango de longitudes de los textos, planeamos usar métodos más sofisticados basados en *statistical physics conformation analysis*. Estos métodos están más allá del objetivo de esta tesis y son objeto de una investigación en curso.

También estamos interesados en una cuestión desafiante: ¿es posible capturar la gramática subyacente a un texto usando metodos análogos a los propuestos en este trabajo? La gramática es un elemento que puede usarse de forma implícita en la compresión de textos. Por ejemplo, consideremos la siguiente frase en español “las chicas altas son buenas bailarinas”. La información de género y de número está contenida en el sufijo “-as”, de manera que un algoritmo de compresión inteligente codificará la cadena “as” de acuerdo con dicha información.

7.2.2 Detección de Plagio en Código Fuente

La detección de plagio es un problema difícil. La frontera entre, por un lado, similitud casual o simple inspiración en el trabajo de otro y, por otro, el simple *corta y pega*, no está bien delimitada, de manera que ciertos casos requerirán siempre un experto humano que distinga lo que es aceptable de lo que no lo es. Sin embargo, se pueden usar varios algoritmos y heurísticas para identificar sospechosos de plagio evidente y señf otros casos más complejos, simplificando de forma evidente la tarea del profesor eliminando comparaciones futiles.

Esta tesis ha presentado AC, una herramienta de detección de plagio que también sirve como *framework* de investigación en detección de plagio en código fuente. A pesar de que AC fue inicialmente diseñado como un entorno para aumentar la aplicabilidad de la NCD para la detección de plagio, ésta ha experimentado un gran desarrollo, ofreciendo en la actualidad muchas mejoras sobre otras herramientas que conforman el estado del arte: el uso de visualizaciones enriquecidas simplifica enormemente el análisis de los resultados de los test de similitud; su implementación in situ e independiente de plataforma evita la problemática legal de privacidad existente en los sistemas basados en web. Asimismo, la preparación de ejercicios para su suministro al sistema puede ser automatizado mediante una intuitiva interfaz gráfica de usuario.

El uso de métodos estadísticos para detección de plagio ha abierto varias lineas de investigación. Un mejor uso del umbral proporcionado por el *identificador de Hampel* puede conseguirse a través de un experimento controlado con estudiantes y copias reales, o a través de nuestro proyecto en desarrollo para generar *benchmarks* artificiales de plagio [29]. Una segunda línea de trabajo consiste en explicar la sorprendente adecuación de la distribución normal para modelos de detección de *outliers*, incluso cuando nos encontramos con tipos muy distintos de ejercicios de programación (en tipo de tarea y lenguaje requerido) y distintas medidas de similitud.

Una vez dos ejercicios de programación han sido marcados como “muy similares” por alguna medida, se espera que el corrector humano compare visualmente ambos códigos en busca de una evidencia de plagio. En otras herramientas para detección de plagio es común que los fragmentos similares de código sean marcados de alguna forma especial para facilitar la inspección humana. AC carece de esta facilidad en

la actualidad. Consideramos que la medida de similitud *Substring* es una buena candidata para señalar dichas áreas de similitud. Por tanto, queda pendiente un mayor refinamiento de dicho test y la subsecuente extensión para facilitar la identificación de zonas de posible plagio.

A pesar de que la detección de plagio en código fuente es un campo relativamente veterano, muy pocas de estas herramientas han sido objeto de una verdadera validación experimental . El único esfuerzo en este sentido es el uso de varios corpus de ejercicios que contienne algunos plagios detectados en la fase de corrección por parte de los profesores. Estos corpus se suministran a la herramienta de plagio para verificar si ésta es capaz de detectar de nuevo los plagios ya detectados por el humano.

Desafortunadamente, es común que ambos, herramienta y corrector humano, fallen en la correcta identificación de plagio, o incluso que incurran en falsos positivos. Un enfoque más experimental requeriría involucrar a un grupo de estudiantes en un experimento de plagio real de algunos ejercicios cuidadosamente seleccionados, mientras que a otro grupo de estudiantes se le pediría soluciones originales. Esto permitiría a los experimentadores disponer de un corpus con autoría completamente conocida. Una alternativa a esto, menos costosa en términos económicos y humanos, es usar programación automática para generar conjuntos artificiales de ejercicios que resuelvan una tarea concreta: los primeros pasos en esta linea se describen en [29]. Sea cual sea el enfoque, la validación de herramientas continua siendo una de nuestras principales lineas de mejora.

7.2.3 Generación de Música

Hemos aportado evidencia de que la NCD es una prometedora función de *fitness* para que los algoritmos genéticos puedan generar música de forma automática. Algunas de las piezas generadas de esta forma recuerdan al estilo de los autores que se usaron como objetivo, a pesar del hecho de que dicha función de *fitness* sólo toma en cuenta el intervalo entre los tonos. Nuestros resultados mejoran otros obtenidos con anterioridad usando una función de fitness diferente [124].

Se han evaluado varios operadores de recombinación para refinar el algoritmo

genético en esta aplicación concreta, encontrando que los mejores resultados se obtienen mediante estrategias mixtas que promuevan la diversidad en las primeras generaciones y luego cambien a una estrategia más focalizada. Este esquema basado en exploración inicial y posterior focalización es análoga a la idea subyacente al *Simulated Annealing* [95].

Planeamos combinar nuestros resultados con los de otros autores (e.g [104, 35]) para usar como *target* del algoritmo genético no sólo una o dos piezas, sino un gran *cluster* de obras del mismo autor, intentando capturar el estilo de una forma más general.

A pesar del hecho de que ya hemos introducido información sobre la duración de los tonos en el algoritmo genético, no hemos hecho uso de ella hasta el momento. Como el diseño actual de nuestro algoritmo facilita dicha incorporación (la NCD puede manejar fácilmente los enteros que representan la duración de los tonos) pretendemos realizar un nuevo conjunto de experimentos para evolucionar la duración de los tonos junto a la melodía.

También planeamos trabajar con estándares de representación musical más ricos, tales como MIDI.

Los resultados presentados en estas tesis sirven como prueba de concepto. Como investigación futura, planeamos realizar una comparación de nuestro método con técnicas de *machine learning* que sean el estado del arte en composición automática de música, para así revelar los puntos fuertes y débiles de nuestro enfoque.

7.3 Nuevas aplicaciones del Modelado Estocástico Algorítmico

Las tres aplicaciones presentadas en el capítulo 5 derivan del uso de Modelado Estocástico, dos para computación evolutiva y una para clasificación. Dos de ellas están íntimamente relacionadas y hacen uso de la presencia de distribuciones de probabilidad de Cola Pesada en los procesos de optimización involucrados en la generación de fractales mediante un algoritmo evolutivo, y en el proceso de entrenamiento de

un perceptrón multicapa. Este descubrimiento se usa para mejorar el rendimiento de ambos algoritmos mediante el uso de estrategias de recomienzo. La última aplicación presentada en esta tesis es una historia exitosa del uso de una heurística aleatoria especial en un algoritmo genético simple, obteniéndose un algoritmo que equivale al estado del arte para la resolución de Problemas de Satisfacción de Restricciones (CSPs).

7.3.1 Generación Acelerada de Fractales de una Dimensión Dada

Las distribuciones de probabilidad de cola pesada ya han sido usadas para modelar fenómenos de la naturaleza tales como patrones climáticos o retrasos en grandes redes de comunicaciones. En esta tesis hemos mostrado que estas distribuciones pueden modelar el tiempo de ejecución de un algoritmo que usa Evolución Gramatical para la generación automática de fractales. Las distribuciones de cola pesada contribuyen a explicar el errático comportamiento de la media y la varianza del tiempo de ejecución de dicho algoritmo, así como las largas colas del que exhibe su función de distribución.

Hemos demostrado que las estrategias de recomienzo pueden mitigar los problemas asociados con las distribuciones de cola pesada y producir una considerable aceleración con respecto al algoritmo sin recomienzos. Estas estrategias explotan la probabilidad no despreciable de encontrar soluciones en ejecuciones cortas, reduciendo la variabilidad de tiempo de una ejecución a otra, así como la posibilidad de que el algoritmo no encuentre solución en grandes tandas de ejecuciones, mejorando por tanto su rendimiento de forma general.

Damos evidencia experimental del valor práctico de aplicar distintas estrategias de recomienzos, incluso en escenarios donde no se tiene conocimiento a priori sobre la distribución de probabilidad del tiempo de ejecución.

Hasta el momento sólo hemos considerado situaciones de conocimiento completo o nulo. En situaciones reales, el tiempo de ejecución o los recursos computacionales están limitados, de manera que sólo se posee *conocimiento parcial* sobre dicha variable. Conjeturamos que, en este escenario, nuestro algoritmo puede sacar provecho de

estrategias de recomienzo dinámicas basados en modelos predictivos, que ya han sido usadas de forma exitosa en problemas combinatorios y de decisión [86, 94, 139]. La futura investigación sobre esta linea estará focalizada en determinar qué conocimiento en tiempo real sobre el comportamiento del algoritmo haría posible construir modelos predictivos para el tiempo de ejecución que produzcan una aceleración aún mayor que la conseguida hasta el momento.

Encontrar las condiciones en las que el tiempo de ejecución de un algoritmo de Evolución Gramatical exhibe colas pesadas supone una línea de investigación desafiante ¿tiene la generación de fractales un comportamiento típico o se trata sólo de una excepción?

7.3.2 Entrenamiento Acelerado de Perceptrones Multicapa

En esta tesis se modela el entrenamiento de un perceptrón multicapa (MLP) como un algorimo de tipo Las Vegas, realizando un caso de estudio sobre la *UCI Thyroid Disease Database*. Damos evidencia estadística sólida de que la distribución de probabilidad del tiempo de entrenamiento pertenece a la familia de distribuciones de colas pesadas, significando esto un decaimiento (sólo) polinómico de probabilidad para ejecuciones largas. Esta propiedad es explotada para reducir el coste de entrenamiento usando dos estrategias simples de recomienzo. La primera supone conocimiento completo de la distribución, obteniendo una reducción del 40% en el tiempo esperado con respecto al entrenamiento sin recomienzos. La segunda supone conocimiento nulo, provocando un ahorro en tiempo que va del 9% al 23%.

Como investigación futura, queremos averiguar si es posible conseguir una mayor aceleración relajando los dos supuestos en los que trabajan los algoritmos Las Vegas (ver secc. 5.2). Esto permitiría incorporar las ya mencionadas estrategias dinámicas de recomienzo (ver [94]) capaces de explotar información época a época sobre la distribución del tiempo de entrenamiento, usando mediciones del comportamiento del algoritmo más allá del tiempo de ejecución.

7.3.3 Evolución de tipo GRASP para Problemas de Satisfacción de Restricciones

Presentamos en esta tesis un algoritmo híbrido evolutivo para la resolución de Problemas de Satisfacción de Restricciones Aleatorias Binarias (*Random Binary CSPs*) que obtiene resultados excepcionales, superando al enfoque más efectivo actual (en términos del número de problemas resueltos) y siendo comparable con el mejor enfoque en términos de eficiencia (en términos del tiempo medio para la resolución).

Nuestro algoritmo híbrido incorpora aspectos GRASP (*Greedy Randomized Adaptive Search Procedures*) de una manera similar a como se hace en [38], donde un mecanismo tipo GRASP se aplica para la traducción genotipo-fenotipo. Nuestro enfoque se diferencia de éste en su novedosa representación, focalizada en encontrar un ordenamiento de variables en lugar de asignaciones variable-valor.

El resto del algoritmo es conceptual y simple, además de no hacer uso de información específica de los problemas de Satisfacción de Restricciones Aleatorias Binarias, lo que aumenta su aplicabilidad en otros problemas. La solución propuesta muestra claramente que el modelado (o representación) es un factor clave en los algoritmos evolutivos.

Creemos que, a pesar de los excelentes resultados obtenidos, hay todavía un gran espacio para la mejora. Consideramos introducir y probar técnicas de aprendizaje y estrategias de recomienzos en nuestro algoritmo. También estamos estudiando hibridaciones con búsqueda local que están comenzado a dar resultados muy prometedores.

Finalmente, y a pesar de que los problemas de Satisfacción de Restricciones Aleatorias Binarios sean un *benchmark* altamente usado, estamos interesados en utilizar CSPs que modelen problemas más realistas, con el propósito de comparar nuestros resultados con otras técnicas tales como *Constraint Programming* u otros algoritmos evolutivos. Los CSPs binarios no son muy comunes en la vida real, y a pesar de que cualquier CSP se puede transformar a un CSP binario en tiempo polinómico [138], planeamos generalizar nuestro algoritmo para que maneje CSPs n-arios.

Appendix A

Mathematical proofs for the phenomenological model of translation

Let us suppose that some text with a histogram (d_2) is a translation of another text with a histogram (d_1), according to the model in the phenomenological section:

$$d_2 \equiv H_M(H_G(d_1; \sigma_G, \delta))$$

The degree of each pair of nodes, $d_1(x)$ and $d_1(x + dx)$ changes according to this model as:

$$\{d(x), d(x + dx)\} \rightarrow$$
$$\{d(x) + G(0, \sigma_M) + \delta G(dx, \sigma_G), d(x) + G'(0, \sigma_M) - \delta G(dx, \sigma_G)\}. \quad \text{(A.1)}$$

Note that due to the exchange nature of H_G, $G(dx, \sigma_G)$ means one and the same number for x and for $x + dx$. Of course we must sum over dx to get d_2 out of d_1.

We have to calculate:

$$\rho(d_2, d_1) = \rho(H_M(H_G(d_1; \sigma_G, \delta)), d_1) = \frac{\langle d_1 d_2 \rangle - \langle d_1 \rangle \langle d_2 \rangle}{\sqrt{\langle {d_2}^2 \rangle - \langle d_2 \rangle^2} \sqrt{\langle {d_1}^2 \rangle - \langle d_1 \rangle^2}}.$$

The variance of the initial text can not be calculated and we take it for given $D = \sigma_1 \equiv \sqrt{\langle {d_1}^2 \rangle - \langle d_1 \rangle^2}$.

The effect of the H_G and H_M in the covariation term vanishes because of the linear nature of the terms in (A.1) and the covariation is exactly equal to ${\sigma_1}^2$.

The only term left to calculate is actually the variance of d_2. For this term we obtain:

$$\begin{aligned} {\sigma_2}^2 &= \langle {d_2}^2 \rangle - \langle d_2 \rangle^2 = \\ &= \mathrm{var}[d(x) + G(0, \sigma_M) + \sum_{dx} \delta G(dx, \sigma_G)] = \\ &= {\sigma_1}^2 + 2\delta^2 {\sigma_G}^2 + {\sigma_M}^2. \end{aligned} \tag{A.2}$$

Note that the term corresponding to the grammar counts twice, because of its exchange nature. From Eq. A.3 the derivation of Eq. 4.2 (page 80) is straightforward.

Appendix B

Benchmarks for Plagiarism Detection Tools

B.1 Introduction

Every computer science lecturer knows that plagiarism detection (copy-catch) is a difficult and extremely time consuming. Several plagiarism detection tools have been implemented since the 1960s: MOSS [14], SIM [79], YAP [153], JPlag [130], SID [33] and AC (presented in this thesis), to name the most widespread in the academic community.

The problem we are interested in occurs when facing the assessment of such tools. Quoting Whale [151, p. 145]: "Assessing different techniques for similarity detection is possible only on a relative scale". The reason is very simple: it is almost impossible to determine whether an assignment solution is a plagiarism of another. What is more, in some cultures, a student will deny a plagiarism even in the most blatant cases. The decision of whether a solution is original is a matter of judgment and generally depends on the sensibility of the grader to find abnormally similar works. This subjectivity contaminates all benchmarks constructed in this way, thus little accuracy can be expected in the assessment.

Two main attempts have been done to ameliorate this issue. The first [71] consists of editing operations on a solution to obtain a plagiarized one: variable and function

name renaming, comment removal, inversion of adjacent statements, permutation of function, etc. The problem with this approach is that these modifications are usually done by researchers, who have a depth understanding of the assignment solution, while students have a very poor understanding of it. Another problem is due to the handicraft nature of this task, generally resulting in benchmarks of very small size. The second attempt (less ambitious) [33] builds plagiarized assignment solutions by means of random insertion of irrelevant statements into the original code in hopes that such insertions will confuse the detection mechanism.

We feel that a more principled approach is necessary to perform a fair comparison of detection tools. In this appendix we present a technique which, fed with some realistic specifications and the grammar of a programming language, is able to generate benchmarks of the desired size, made of a subset containing independent solutions to the specifications, coded *from scratch*, and another subset - the plagiarized solutions - built from one or two solutions taken from the original subset. Both the authentic and the plagiarized sets are built by means of evolutionary techniques adapted from Grammatical Evolution [123], whose suitability for automatic programming is well established.

In this appendix we try to show that having an arbitrary number of large solutions to an assignment, with a priori knowledge of their phylogeny, is the first step towards a benchmark for plagiarism.

The remainder of the appendix is organized as follows: in Sect. B.2 we detail the benchmark generation technique; in Sect. B.3 we give experimental evidence of the suitability of this technique by means of several examples. Sect. B.4 describes the reasons why we think that our approach to the generation of benchmarks is appropriate. Sect. B.5 proposes some conclusions and possibilities for improvement.

B.2 Automatic generation of benchmarks

Our benchmarks simulate the answers of different students to a practical assignment. In this work, each benchmark consists of APL2 functions which fit a set of points generated by applying one particular function to the set of inputs (values of x) $1, 2, 3, 4, 5$.

Four benchmarks have been generated, corresponding to the following toy problem functions: $x^2, 1 + x + x^2 + x^3, \cos(\log x)$ and $\log(x^3)$.

To mimic the solutions of the students to this assignment, two sets of programs are generated for each benchmark: the first is considered *original*, the second contains plagiarisms. Both sets are built by means of a genetic engine in two phases: in the first, 30 original programs are generated using grammatical evolution (GE)[123]. Then 14 solutions are generated by applying several selected genetic operators, trying to reproduce the basic plagiarizing techniques performed by students.

```
E:= O | oO  | OoO
O:= 0|1|2|3|4|5|6|7|8|9|
    X|X|X|Z|Z|Z|Z|Z|Z|Z|
    (E)|(E)|(E)|(E)|(E)|(E)|(E)|(E)|(E)|(E)
o:= +|-|×|÷|⌊|⌈|*|○|⊛|!
```

Figure B.1: Context free grammar to generate and modify the original APL2 functions. The repetition of a symbol affects the probability of its choice.

All the solutions consist of an APL2 function with the same header: the name of the function is F, their input is argument X, and they return the value of variable Z. The first instruction assigns the value of X to Z to guarantee F always returns a proper value. In the 'original' solutions, F contains a number of additional instructions between 0 and 255. Every one assigns the value of an expression to variable Z. These expressions are generated by means of GE. Figure B.1 shows the context free grammar used to generate the expressions. E is the axiom. A genotype consists of a number (between 100 and 200) of integers (codons) in the [0,255] interval. The first codon indicates the number of instructions to be added to the function. The genotype is mapped in the usual way, deriving the number of expressions indicated by the first codon from the initial word E. The alternate execution mechanism provided by APL2 has been used to intercept semantic errors in the generated expressions, thus avoiding program failures and unexpected end conditions. Each instruction is executed in the same way and occupies a single line, therefore the size of the generated

APL2 function is equal to the value of the first codon plus one.

The fitness function is the mean quadratic error of the generated APL2 function applied to the set of control points, as compared with the set of control results, scaled by a factor to punish long genotypes (size(genotype)/100), to favor parsimonial answers. The fitness optimum value is 0. The experiment stops when the solution found has a fitness value less than 1 or when the number of generations equals 1000. The genetic operators used are taken from *mutation with elision, mutation with elongation, genotypic recombination* and *phenotypic recombination.*

In the generation of the 30 original solutions we have used 30 different populations with one independently generated genotype each (corresponding to 30 different random seeds), which is equivalent to performing a hill-climbing local search. The genotype of the next population is obtained by applying mutation with elision to the previous individual, which is either mutated or shortened with the same probability (0.5). Elision deletes a codon in an arbitrary location on the genotype. The new genotype replaces the old one only if its fitness is better.

Mutation with elongation is similar to mutation with elision: an arbitrary codon is added in a random location on the genotype, rather than being deleted. Each time the genotype suffers this operator, the process is repeated 5 times.

One single point recombination is used in genotypic and phenotypic recombination. In our approach, only the child that begins like its first parent is taken into account. If we want to get two children, the same parents may be used in the opposite order, although in the second case the recombination point will usually be different. The procedure is performed 5 times and the child with the best fitness is selected as the result of the recombination.

Phenotypic recombination acts directly on the APL2 functions, so each child will contain the first lines of one parent and the remaining instructions of the other parent. We have included this approach to compensate the well-known tendency to phenotypic disruption caused by the ripple crossover operator used in GE program generation [123].

We have applied three different techniques to plagiarize one or two original functions. First the 5th, 10th, 15th, 20th, 25th and 30th original solutions are plagiarized

using mutation with elongation to generate 6 new APL2 solutions. This technique mimics plagiarism from a single source, where the source is changed by adding and replacing a few fragments. The second and third techniques simulate plagiarisms from two sources (two different originals are mixed to produce a new solution) by means of recombination. The second technique generates 4 new APL2 functions through the genotypic recombination of the following couples of originals: 5th and 10th, 10th and 5th, 15th and 20th and 20th and 15th. The third technique mixes the 20-15, 7-14, 5-22, and 30-1 couples using phenotypic recombination. Figure B.2 shows a graphic scheme of the whole process.

Figure B.3 shows the plagiarism relations existing in the benchmarks. Round vertices stand for original assignments, squares for plagiarism using a single source, rhomboids and octagons for the two different types of plagiarism using two sources. A black solid line between vertices A and B denotes that A has used B as the unique source of plagiarism; a red dashed lines between A and B denotes that A has used B as one of the two sources of plagiarism; a green dotted line denotes that they are indirect copies, i.e. they share a common source of plagiarism.

B.3 Experimental results

Summarizing: we have generated 4 benchmarks, each consisting of 44 assignments coded in APL2. Each benchmark is divided in the same manner:

- 30 original solutions, named P1 to P30.
- 6 *mutational plagiarized results*, named MPx, where x stands for the original source for the plagiarism (5, 10, 15, 20, 25 and 30).
- 4 *genotypic recombination plagiarized results*, named PxRGPy, where x and y represent the two source genotypes used as parents in genotypic recombination. y is considered the first parent.
- 4 *phenotypic recombination plagiarized results*, named PxRFPy, where x and y represent the two source genotypes used as parents in phenotypic recombination.

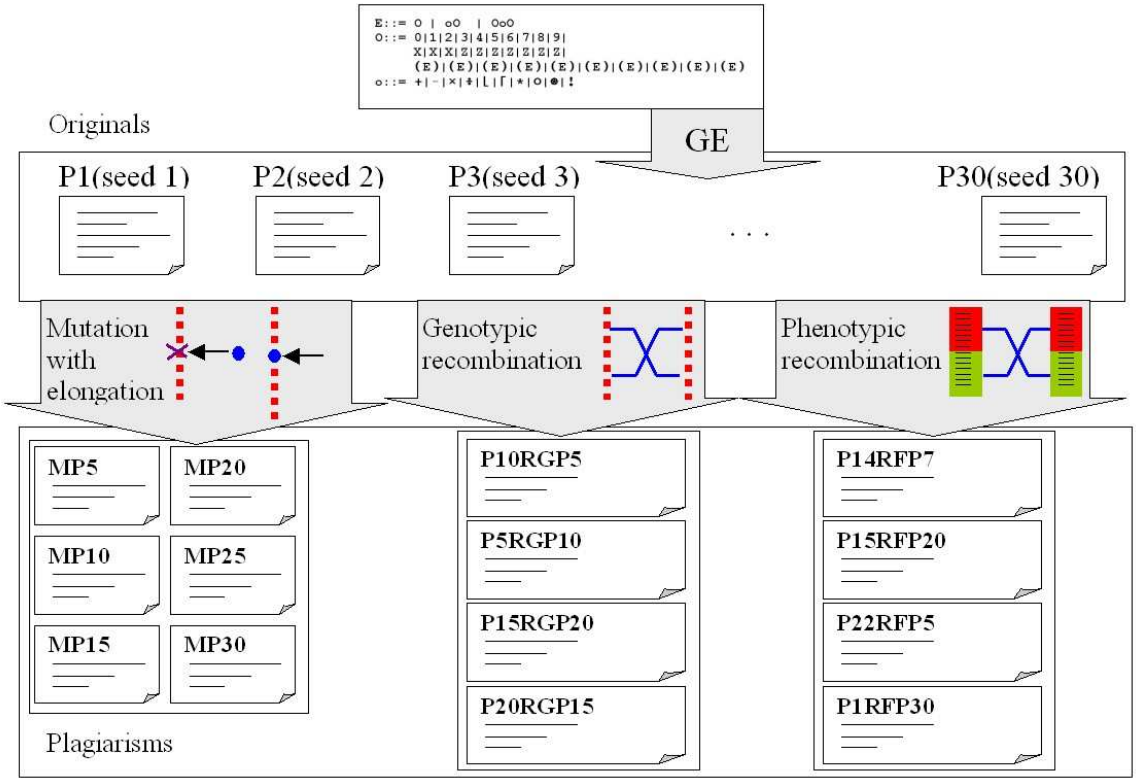

Figure B.2: Graphical scheme of the whole process

y is considered the first parent.

As indicated in the previous Section, the specifications of the 4 benchmarks were the functions x^2, x^3+x^2+x+1, $\cos(\log x)$ and $\log x^3$. Some statistic of the generation process are shown in Table B.1. Executions took about one hour per benchmark on a 2.5 GHz computer with 512 MBytes memory.

Now we want to check whether the sets generated with this process match our idea of typical plagiarism. To do this, we are going to feed our 4 benchmarks into the plagiarism detection tool AC [66], which works in two steps: in the first, one of the similarity metrics available must be selected, all giving results ranging between 0 (complete similarity) to 1 (complete dissimilarity). Then, once pairwise distances between all assignments have been obtained, several graphical interfaces are displayed to point abnormal low distances which could imply a plagiarism.

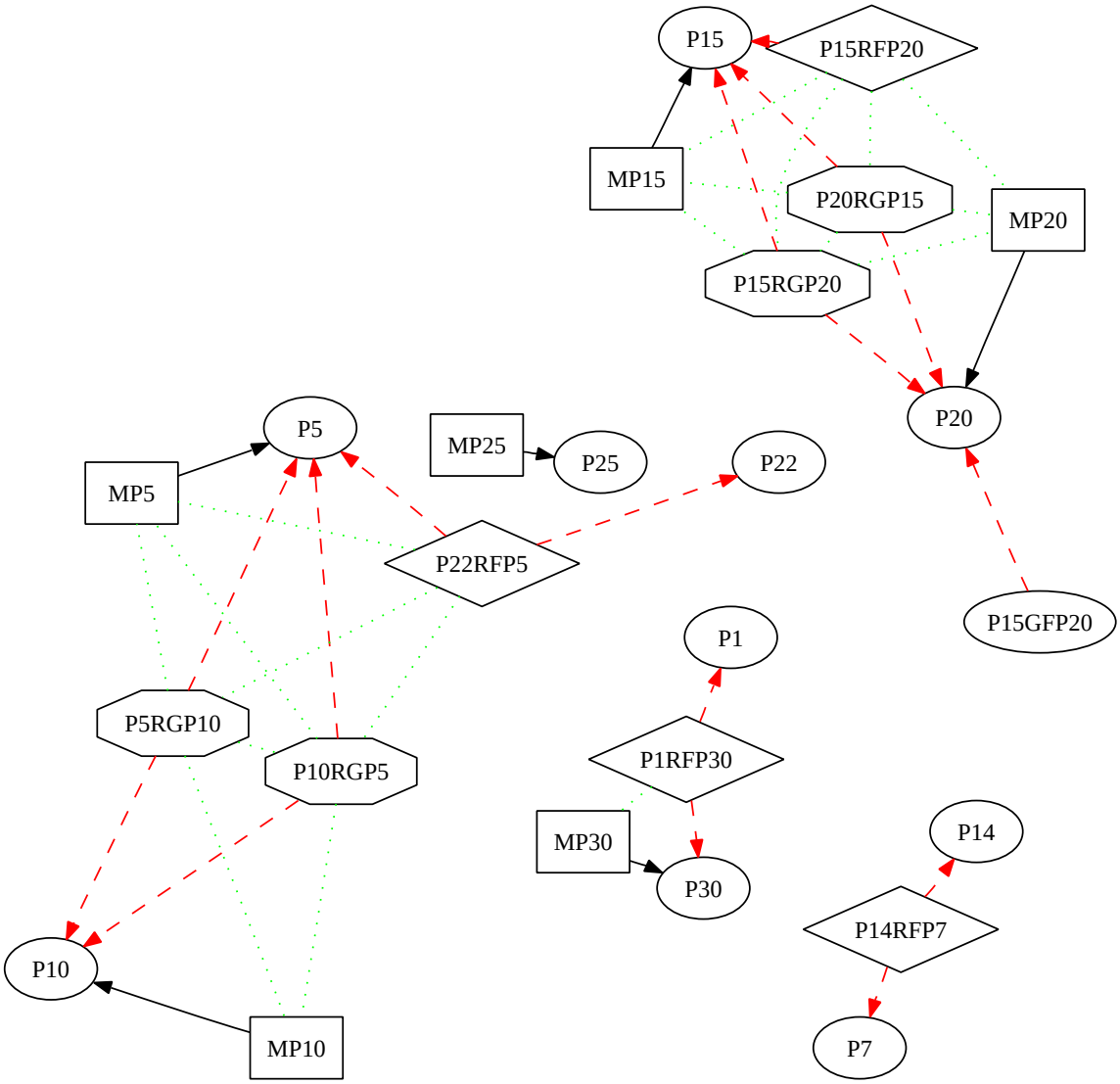

Figure B.3: Plagiarism relations of the benchmarks. Round vertices stand for original assignments, squares for plagiarism using a single source, rhomboids and octagons for the two different types of plagiarism using two sources. A black solid line between vertices A and B denotes that A has used B as the unique source of plagiarism; a red dashed lines between A and B denotes that A has used B as one of the two sources of plagiarism; a green dotted line denotes that they are indirect copies, i.e. they share a common source of plagiarism.

	ave. program size	ave. instructions
x^2	1889	120.25
$x^3 + x^2 + x + 1$	1954	126
$\cos(\log x)$	2349	140
$\log x^3$	1735	108

Table B.1: Statistics of the generation of the four benchmarks (the average program size is measured in bytes).

In Fig. B.4 we display a similarity graph obtained by computing a novel similarity distance on the benchmark x^2. This distance looks for the longest-most infrequent string which two assignments have in common; the longer and the more infrequent the string, the lower the distance between solutions. A graph is provided by the tool, whose vertices stand for each assignment solution and whose edges represent the distance between each couple of solutions. Only the distances less than a value chosen with a slider are shown. The bigger and hotter the edge, the smaller is the value (or the more similar are the sources). This graph constructs and displays minimum spanning trees (MSTs) built only with those distances below the threshold, 0.01 in this Figure. It can be seen that the obtained MSTs are exactly what one would desire: plagiarized versions clustered with their sources, in all but assignment P17, which is a paradigmatic case of an accidental coincidence. In Fig B.5, where the threshold has been increased to 0.02, the overwhelming majority of the plagiarized versions have been detected (13 out of 14), against only one additional non-plagiarized MST (P3-P28), i.e. plagiarized versions tend to appear long before non-plagiarized ones.

Fig. B.6 shows results for a different benchmark, function $\cos(\log x)$. The distance used is the normalized compression distance (NCD, see [103]) which, in simple terms, gives a low distance to sources which compress well together, i.e. which share a large amount of literal coincidence. Finally, the visualization is based on *individual temperature histograms*, meaning that the hotter the color, more elements are in this range. Each row displays the histogram of NCD distances between the assignment in the leftmost part of the row and the rest of the benchmark. It can be seen that plagiarized versions are nearer to their sources than to others at distances usually *outlying* from the rest of the sample.

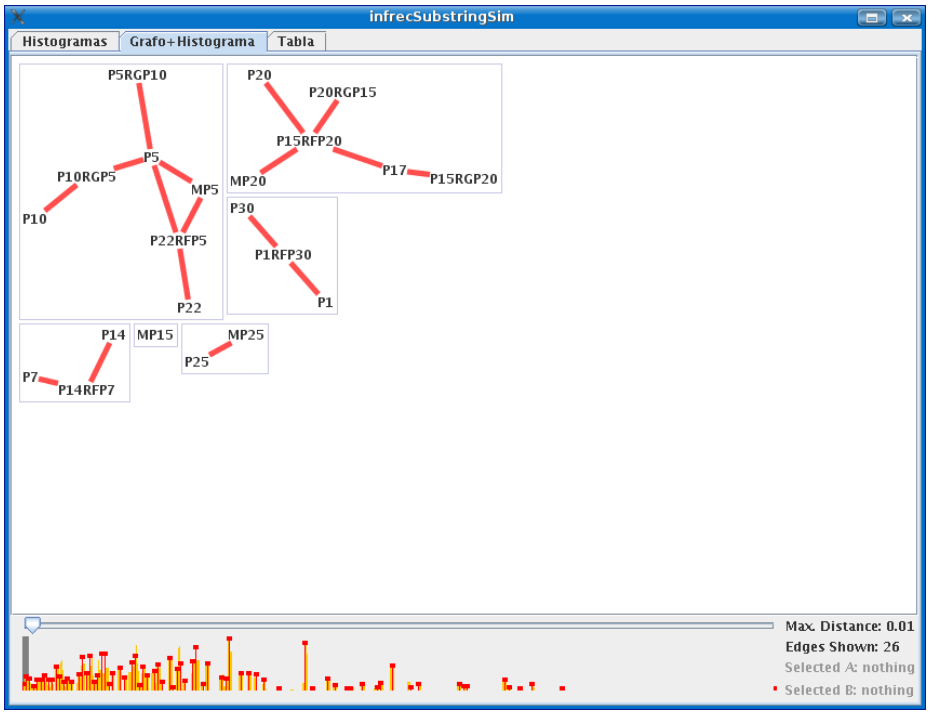

Figure B.4: The vertices of the graph stand for each assignment of the benchmark x^2 and the edges represent values of pairwise distances calculated using the longest-most infrequent similarity distance. Only the assignments whose pairwise distance is lower to the distance chosen by the slider (below) are shown. In this figure, the slider is set to 0.01. The bigger and hotter (more red) is the edge between two vertices (assignments), the smaller is the distance (or the more similar are the sources).

Another option available in AC provides a raw list of pairs sorted by their increasing chosen distance. In Tables B.2 and B.3 we display the 15 lowest distances for benchmarks $\log x^3$ and $x^3 + x^2 + x + 1$, where the NCD and the longest-most infrequent distances are used respectively. In both, authentic-plagiarized or plagiarized-plagiarized-from-the-sameassignment sources are generally top ranked, specially in the case of $\log x^3$, where no non-plagiarized pair appears in the table. Therefore, even if no graphical help is used, plagiarized pairs manifest by themselves.

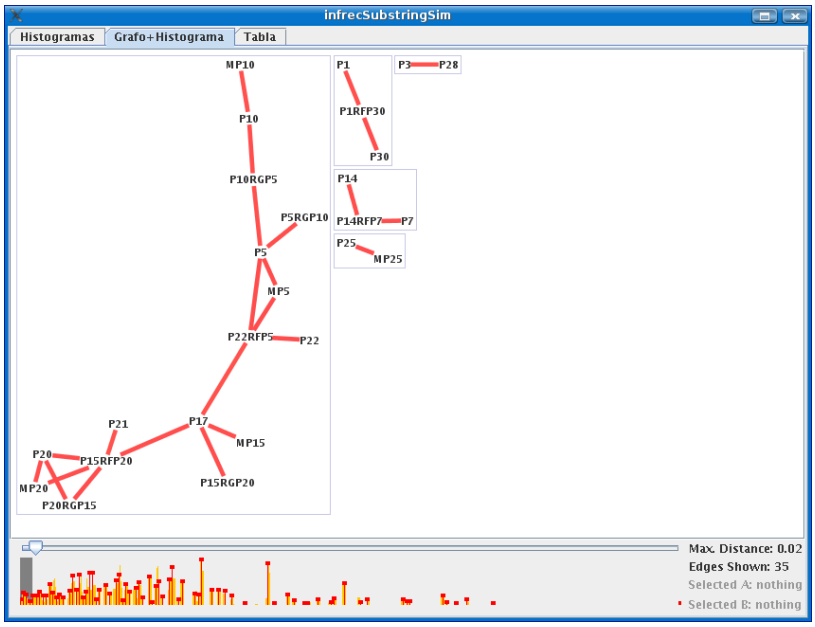

Figure B.5: Analog to Fig. B.4 but with threshold increased slider set to 0.02.

B.4 Discussion

Language APL2 has been selected to program our benchmarks for the following reasons:

- APL2 is a very powerful language, especially for the generation of expressions, with a large number of primitive functions and operators available.

- The APL2 expression grammar is very simple and can be implemented with just three non-terminal symbols, which simplifies the grammatical evolution process.

- APL2 instructions can be protected to prevent semantic and execution errors from giving rise to program failures. In this way, we can rest assured that all the programs in the benchmark will execute (although their results may not be a good answer to the assignment). grammatical evolution is also simplified, because we don't need to include any semantic information, such as attribute grammars or Chistiansen's grammars [50, 6].

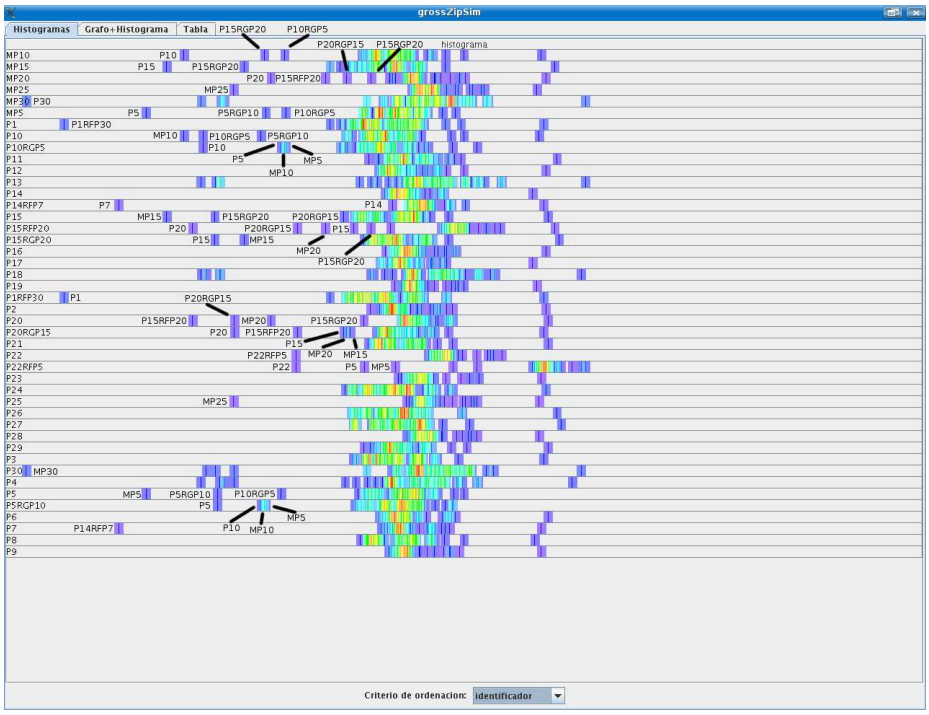

Figure B.6: We explain the first row, the next are analogue. We calculate the parwise distances between MP10 (leftmost part of the row) and the rest of submissions of the cos(log x) corpora. We then depict a 'hue histogram' of the distances, i.e. the more red (hotter) is the color at some point (distance), the higher is the number of submissions lying at that distance from MP10. The horizontal axis of the hue histogram ranges from 0 (leftmost part, complete similarity) to 1 (rightmost part, complete dissimilarity).

- APL2 makes it possible to define new programming functions in execution time, thus providing the feasibility of integrating the fitness computation with the genetic algorithms which generate the benchmark. With a compilable language, such as C, this would be very difficult. For a short introduction of the APL2 language see [18].

In Sections B.2 and B.3 we have tried, first conceptually and then empirically, to show that copies generated by our procedure match the intuitive idea of plagiarism: an improbable high similarity between works done by different authors. If we consider this definition in depth, we find that a philosophical problem shows up:

$x^3 + x^2 + x + 1$		
9.881421E-4	P10	MP10
0.0014822131	P15RFP20	P15
0.0014822131	P20RGP15	P15
0.0014822131	P20RGP15	P15RFP20
0.0019762842	P30	P1RFP30
0.0019762842	MP20	P20RGP15
0.0024703552	P15RGP20	P20RGP15
0.0029644263	MP30	P1RFP30
0.0029644263	MP30	P30
0.0039525684	P10RGP5	MP10
0.004446639	P18	P7
0.004446639	P18	P14RFP7
0.0049407105	P26	P4
0.0049407105	P26	P12
0.0049407105	P26	P18

Table B.2: Lowest 15 pairwise distances obtained using the longest-most infrequent distance on the benchmark $x^3 + x^2 + x + 1$.

Assume that students have some specifications for an assignment and there exists only an optimal way to code the solution. This is what we consider as optimal:

- Perfect functionality: for every input, the computer program must produce the specified output.
- Maximal parsimony: the program must be as simple as possible. During the generation process solutions with a high number of lines are penalized, although other measures of parsimony could have been used (e.g. [137]).

In this way, there may exist only one solution with perfect functionality and maximal parsimony. These conditions are not very restrictive if, for example, we consider the way in which programming challenges are qualified (see for example [1]).

In this situation, two students delivering the optimal solution to the grader could incur in the already mentioned definition of plagiarism: absolute coincidence. What could the grader do in this situation? It could be argued that it is highly improbable that two students end up with the same code and consider them plagiarisms,

$\log x^3$		
0.01538462	P1	P1RFP30
0.02339181	P15RFP20	P15
0.02339181	MP15	P15
0.02339181	MP15	P15RFP20
0.02469136	MP25	P25
0.13580246	P25	P20RGP15
0.13580246	MP25	P20RGP15
0.15789473	P20RGP15	P15
0.15789473	P20RGP15	P15RFP20
0.16374269	MP15	P25
0.16428572	P10RGP5	P5
0.16959064	P25	P15
0.16959064	P25	P15RFP20
0.16959064	MP15	P20RGP15
0.16959064	MP25	P15

Table B.3: Lowest 15 pairwise distances obtained using NCD on the benchmark $\log x^3$.

but the students can reject this argument with the easy explanation that they have optimized the program independently until no further improvement was possible. If the programmers are good enough, the probability of reaching the same optimal or quasi-optimal solution is very high.

The solution to this problem is provided by the experience of the grader at copy-catching: plagiarism is usually detected much more by observing abnormal coincidences in *trash code*, i.e. erroneous or spurious code, than for finding coincidences like similar variable or function names in correct portions of the code. The underlying idea is that there are few ways of doing things correctly, but many of doing it inaccurately, so why should two students have chosen the same way of making mistakes? Reported cases of copy-catching talk about shared lines of code that simply do nothing or two compiled codes which produce the same errors when executed. This happens because plagiarists have a poor understanding of the code and tend to incorporate *trash code* from the source into their code. Even those most daring who try to change some fragments of code usually fail to do it usually worsening that code.

```
Z←F X
Z←X
'' ⎕EA 'Z←7↓7' [trash]
'' ⎕EA 'Z←-Z'  [trash]
'' ⎕EA 'Z←((×7)+(6))×Z' [trash, shared]
'' ⎕EA 'Z←-Z' [trash, shared]
'' ⎕EA 'Z←⌈(⌈X)' [trash, shared]
'' ⎕EA 'Z←2×Z' [trash, shared]
'' ⎕EA 'Z←÷X'  [trash, shared]
'' ⎕EA 'Z←(Z↓4)' [trash, shared]
'' ⎕EA 'Z←5' [trash, shared]
'' ⎕EA 'Z←(⌊((Z)-(↓X)))×8' [trash, shared]
'' ⎕EA 'Z←3○(⌊Z)' [trash, shared]
'' ⎕EA 'Z←×6' [trash, shared]
'' ⎕EA 'Z←X' [trash, shared]
...
```

```
Z←F X
Z←X
'' ⎕EA 'Z←7↓9' [trash]
'' ⎕EA 'Z←9'   [trash]
'' ⎕EA 'Z←Z'   [trash]
'' ⎕EA 'Z←((×7)+(6))×Z'[trash, shared]
'' ⎕EA 'Z←-Z' [useless, shared]
'' ⎕EA 'Z←⌈(⌈X)' [trash, shared]
'' ⎕EA 'Z←2×Z' [trash, shared]
'' ⎕EA 'Z←÷X'  [trash, shared]
'' ⎕EA 'Z←(Z↓4)' [trash, shared]
'' ⎕EA 'Z←5'  [trash, shared]
'' ⎕EA 'Z←×(((Z)-(↓X))×8)' [trash, shared]
'' ⎕EA 'Z←3○(⌊Z)' [trash, shared]
'' ⎕EA 'Z←×6' [trash, shared]
'' ⎕EA 'Z←X' [trash, shared]
...
```

Figure B.7: Two fragments of code of P15 (left) and MP15 (right) from the $\cos(\log x)$ benchmark. Dots "..." stand for code not shown.

```
...
'' ⎕EA 'Z←Z○Z' [trash, shared]
'' ⎕EA 'Z←X'   [trash, shared]
'' ⎕EA 'Z←0×Z' [trash, shared]
'' ⎕EA 'Z←Z'   [trash, shared]
'' ⎕EA 'Z←-Z'  [trash, shared]
'' ⎕EA 'Z←⍟X'  [trash]
'' ⎕EA 'Z←×X'  [trash]
'' ⎕EA 'Z←Z'   [trash]
'' ⎕EA 'Z←Z'   [trash]
...
```

```
...
'' ⎕EA 'Z←Z○Z' [trash, shared]
'' ⎕EA 'Z←X'   [trash, shared]
'' ⎕EA 'Z←0×Z' [trash, shared]
'' ⎕EA 'Z←Z'   [trash, shared]
'' ⎕EA 'Z←-Z'  [trash, shared]
'' ⎕EA 'Z←Z'         [trash]
'' ⎕EA 'Z←-((Z⍟9))' [trash]
'' ⎕EA 'Z←Z⍟X'       [trash]
'' ⎕EA 'Z←((Z))'     [trash]
...
```

Figure B.8: Two fragments of code of P10RGP5 (left) and P5 (right) from the $\log x^3$ benchmark.

To simulate the plagiarism process, one has to take this into account. It turns out that there is a strong correspondence of these ideas with search and optimization: perfect solutions are equivalent to *global optima*, while approximate solutions, those which include trash code, are equivalent to *local optima*.

Our proposed generation process can be seen in this light. We perform a *light* optimization, i.e., we try to maximize functionality and parsimony, without seeking the global optimum. This is done by limiting the number of optimization steps. In a second step the counterfeits are created. Using genotypical mutation with elongation, a new solution is created which will share a big percentage of code with the original. The shared code will consist of both useful and trash code. On the other hand, the new code generated by the mutation/elongation will probably worsen the fitness of the assignment.

Figure B.7 shows code fragments of assignments P5 and MP5 from benchmark $\cos(\log x)$. Shared code and trash code are annotated to the right. Detection is

possible precisely due to the shared trash code, not the useful code, because the latter can be the same in both cases with a high probability. The same happens if we consider genotypical (Fig. B.8) or phenotypical (Fig. not shown) recombination. The obtained codes are mixtures of the sources where trash code has been inherited from both. As it can be seen in all examples, the trash code is the fingerprint for plagiarism detection.

B.5 Further work in automatic plagiarism benchmark generation

Copy-catching computer tools are difficult to evaluate, because actual work by real students is always subject to uncertainty. To help in their evaluation for the field of computer programming assignment plagiarism, we offer a procedure which automatically generates different benchmarks which may be useful for this purpose. A benchmark for a given assignment is made of a number of original solutions, together with another set of plagiarized solutions, generated in such a way to mimic the way in which students act. We have used these benchmarks to assess the performance of one state-of-the-art detection tool (AC) with satisfactory results. The APL2 programming language has been selected for the implementation of the benchmarks because of certain properties which make it very applicable as a first instance of this process.

Benchmarks for programming in other programming languages, such as C or Java, will be attempted in the next step of our research. We will also improve the generational mechanism, so that it can code bigger and more complex assignments, not just toy problems: for instance, assignments which imply building several functions or source files. This can be achieved by using smarter genetic operators and/or other different automatic programming techniques (classic GP trees [99], etc).

We also intend to perform direct comparisons between the tools by using benchmarks generated with our procedure. This could be done by making some statistical analysis of the number of plagiarized sources correctly detected by each tool. It would be also possible to weight the different types of plagiarism for that analysis because,

in real docent environments, the detection of single source plagiarism is usually less challenging that when several sources are mixed.

and will dedicate some effort to further study in depth the role of trash code in plagiarism identification.

The APL2 program used to generate the benchmarks and the four benchmarks themselves can be found at

http://www.eps.uam.es/~mcebrian/plagiarism-benchmark

Bibliography

[1] ACM International Collegiate Programming Contest [Online]. Available: http://icpc.baylor.edu/icpc/.

[2] BibleGateway: A searchable online Bible in over 50 versions and 35 languages [Online]. Available: http://www.biblegateway.com

[3] DEFLATE Compressed Data Format Specification [Online]. Available: ftp://ds.internic.net/rfc/rfc1951.txt.

[4] The Universal Declaration of Human Rights [Online]. Available: http://www.unhchr.ch/udhr/index.htm.

[5] Yale Face Database [Online]. Available: http://cvc.yale.edu/projects/yalefaces/yalefaces.html.

[6] M. de la Cruz A. Ortega and M. Alfonseca. Christiansen Grammar Evolution: grammatical evolution with semantics. *Accepted for publication in IEEE Transactions on Evolutionary Computation.*

[7] N. Abramson. *Information Theory and Coding.* McGraw-Hill, New York, 1963.

[8] D. Achlioptas, M.S.O. Molloy, L.M. Kirousis, Y.C. Stamatiou, E. Kranakis, and D. Krizanc. Random Constraint Satisfaction: A More Accurate Picture. *Constraints*, 6(4):329–344, 2001.

[9] R. Acín-Pérez, M.P. Bayona-Bafaluy, M. Bueno, C. Machicado, P. Fernández-Silva, A. Pérez-Martos, J. Montoya, MJ López-Pérez, J. Sancho, and J.A.

Enríquez. An intragenic suppressor in the cytochrome c oxidase I gene of mouse mitochondrial DNA. *Human Molecular Genetics*, 12(3):329–339, 2003.

[10] L.M. Adleman and M.D.A. Huang. Primality testing and abelian varieties over finite fields. *Lecture notes in mathematics*, 1992.

[11] R. Adler, R. Feldman, and MS Taqqu. A Practical Guide to Heavy Tails: Statistical Techniques for Analyzing Heavy Tailed Distributions, Birkaüser. 2000.

[12] K. Agrawal and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160:781–793, 2004.

[13] R.K. Ahuja, J.B. Orlin, and D. Sharma. New neighborhood search structures the capacitated minimum spanning tree problem. Technical report, University of Florida, 1998.

[14] A. Aiken et al. Moss: A system for detecting software plagiarism [Online]. *University of California–Berkeley.* Available: http://www.cs.berkeley.edu/aiken/moss.html

[15] M. Alfonseca, M. Cebrian, and A. Ortega. Evolving Computer-Generated Music By Means of the Normalized Compression Distance. *WSEAS Transactions on Information Science and Applications*, 2(9):1367–1372, 2005.

[16] M. Alfonseca and A. Ortega. A study of the representation of fractal curves by L systems and their equivalences. *IBM Journal of Research and Development*, 41(6):727–736, 1997.

[17] M. Alfonseca and A. Ortega. Determination of fractal dimensions from equivalent L systems. *IBM Journal of Research and Development*, 45(6):797–805, 2001.

[18] M. Alfonseca and D. Selby. APL2 and PS/2: the Language, the Systems, the Peripherals. *APL Quote Quad (ACM SIGAPL)*, 19(4):1–5, Aug. 1989.

[19] U. Arnason, A. Gullberg, and A. Janke. Mitogenomic analyses provide new insights into cetacean origin and evolution. *Gene*, 333:27–34, 2004.

[20] U. Arnason, A. Gullberg, and B. Widegren. The complete nucleotide sequence of the mitochondrial DNA of the fin whale, Balaenoptera physalus. *Journal of Molecular Evolution*, 33(6):556–568, 1991.

[21] U. Arnason, A. Gullberg, and B. Widegren. Cetacean mitochondrial DNA control region: sequences of all extant baleen whales and two sperm whale species. *Molecular Biology and Evolution*, 10(5):960–970, 1993.

[22] J.E. Baker. Reducing bias and inefficiency in the selection algortihm. In *2nd Int. Conference on Genetic Algorithms and Their Applications*, pages 14–21, 1987.

[23] H-G. Beyer and H-P. Schwefel. Evolution strategies. *Natural Computing*, 1:3–52, 2002.

[24] J.A. Biles. GenJam: A Genetic Algorithm for Generating Jazz Solos. *Proceedings of the 1994 International Computer Music Conference*, pages 131–137, 1994.

[25] E. Bilotta and P. Pantano. Synthetic Harmonies: An Approach to Musical Semiosis by Means of Cellular Automata. *LEONARDO*, 35(2):153–159, 2002.

[26] B.F. Braumoeller and B.J. Gaines. Actions Do Speak Louder than Words: Deterring Plagiarism with the Use of Plagiarism-Detection Software. *PS: Political Science and Politics*, 34(04):835–839, 2002.

[27] J. Bull, C. Collins, E. Coughlin, S. Developer, D. Sharp, and P. Square. Technical review of plagiarism detection software report [Online]. *CAA University of Luton*. Available: http://online.northumbria.ac.uk/faculties/art/information_studies/Imri/ Jiscpas/docs/jisc/luton.pdf.

[28] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Systems Research Center of Digital Equipment Corporation, Technical Report 124, Palo Alto, California, 1994.

[29] M. Cebrian, M. Alfonseca, and A. Ortega. Automatic Generation of Benchmarks for Plagiarism Detection Tools using Grammatical Evolution. In *Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO'2007)*, London, UK, July 2007. ACM SIGEVO. arXiv:cs.NE/0703134.

[30] Manuel Cebrián and Iván Cantador. Exploiting heavy tails in training times of multilayer perceptrons. a case study with the uci thyroid disease database. arXiv:0704.2725, 2007.

[31] G.J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

[32] J. Chen and J.Y. Nie. Web parallel text mining for Chinese-English cross-language information retrieval. *International Conference on Chinese Language Computing, Chicago, Illinois*, 2000.

[33] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker. Shared information and program plagiarism detection. *Information Theory, IEEE Transactions on*, 50(7):1545–1551, 2004.

[34] R. Cilibrasi and PMB Vitani. Clustering by Compression. *Information Theory, IEEE Transactions on*, 51(4):1523–1545, 2005. Software available: http://www.complearn.org.

[35] R. Cilibrasi, P. Vitanyi, and R. de Wolf. Algorithmic clustering of music. *Web Delivering of Music, 2004. WEDELMUSIC 2004. Proceedings of the Fourth International Conference on*, pages 110–117, 2004.

[36] J. Clare. Computer plagiarism threatens the value of degrees. *Daily Telegraph*, 3(7):2000, 2000.

[37] M.D. Coble, R.S. Just, J.E. OCallaghan, I.H. Letmanyi, C.T. Peterson, J.A. Irwin, and T.J. Parsons. Single nucleotide polymorphisms over the entire mtDNA genome that increase the power of forensic testing in Caucasians. *International Journal of Legal Medicine*, 118(3):137–146, 2004.

[38] C. Cotta and A.J. Fernández. A hybrid grasp - evolutionary algorithm approach to golomb ruler search. In Xin Yao et al., editors, *Parallel Problem Solving From Nature VIII*, number 3242 in Lecture Notes in Computer Science, pages 481–490. Springer, 2004.

[39] T.M. Cover and J.A. Thomas. *Elements of information theory*. Wiley New York, 1991.

[40] B. G. W. Craenen. Javacsp: a random binary constraint satisfaction problem instance generator in java [Online]. Available: http://www.xs4all.nl/˜bcraenen/JavaCsp/download.html.

[41] BGW Craenen and AE Eiben. Stepwise adaption of weights with refinement and decay on constraint satisfaction problems. *Proceedings of Genetic and Evolutionary Computation Conference*, pages 291–298, 2001.

[42] B.G.W. Craenen, A.E. Eiben, and J.I. van Hemert. Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 7(5):424–445, 2003.

[43] N. Crato. Estimation Of The Maximal Moment Exponent With Censored Data. *Communications in Statistics–Simulation and Computation, Vol29 No4*, pages 1239–1254, 2000.

[44] K. Culik II and S. Dube. L-systems and mutually recursive function systems. *Acta Informatica*, 30(3):279–302, 1993.

[45] F. Culwin, A. MacLeod, and T. Lancaster. Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools. *South Bank University Technical Report SBUCISM-01-02*, 2001.

[46] Miguel de Cervantes. Don kihot [Online]. Translated by N. Ljubimov, Hudozhestvenaya literatura, Moskva, 1988, Available: http://lib.ttknet.ru/koi/INOOLD/SERVANTES/donkihot1.txt.

[47] Miguel de Cervantes. Don quichotte [Online]. Translated by L. Viardot. Available: http://www.gutenberg.org/files/16066/16066-8.txt.

[48] Miguel de Cervantes. Don quijote [Online]. Available: http://cvc.cervantes.es/obref/quijote/edicion/parte1/parte01/cap01/default.htm .

[49] Miguel de Cervantes. Don quixote [Online]. Translated by J. Ormsby. Available: http://www.gutenberg.org/etext/996.

[50] A. de la Cruz, M. Ortega and m. Alfonseca. Attribute grammar evolution. *Lecture notes in computer science*, pages 182–191.

[51] K. de Leeuw, E.F. Moore, C.E. Shannon, and N. Shapiro. Computability by probabilistic machines. *Automata Studies, Ann. Math. Studies*, 34:183–212.

[52] PJF de Lima. On the robustness of nonlinearity tests to moment condition failure. *Journal of Econometrics*, 76(1):251–280, 1997.

[53] WH Delashmit and MT Manry. Enhanced robustness of multilayer perceptron training. In *Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers*, pages 1029–1033, 2002.

[54] H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Reactive grasp and tabu search based heuristics for the single source capacitated plant location problem. *INFOR*, 37:194–225, 1999.

[55] M. Dietzfelbinger. *Primality Testing in Polynomial Time: From Randomized Algorithms to 'Primes are in P'*. Springer, 2004.

[56] W. Duch, R. Adamczak, and N. Jankowski. Initialization and optimization of multilayered perceptrons. In *Proceedings of the 3rd Conference on Neural Networks and their Applications*, pages 99–104, Kule, Poland, Oct. 1997.

[57] AE Eiben. Evolutionary algorithms and constraint satisfaction: Definitions, survey, methodology, and research directions. *Theoretical Aspects of Evolutionary Computing*, pages 13–30, 2001.

[58] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[59] P. Embrechts, T. Mikosch, and C. Klèuppelberg. *Modelling Extremal Events for Insurance and Finance*. Springer, 1997.

[60] K.J. Falconer. Fractal Geometry: Mathematical Foundations and Applications. *Mathematical Foundations and Applications*, 1990.

[61] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[62] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[63] DB Fogel. *Evolutionary Computation–The Fossil Record 1998*. IEEE Press, 1998.

[64] L.J. Fogel, A.J. Owens, M.J. Walsh, et al. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[65] DR Foran, JE Hixson, and WM Brown. Comparisons of ape and human sequences that regulate mitochondrial DNA transcription and D-loop DNA synthesis. *Nucleic Acids Res*, 16(13):5841–5861, 1988.

[66] M. Freire, M.Cebrian, and E. del Rosal. Ac: An integrated source code plagiarism detection environment, 2007. arXiv:cs.IT/0703136.

[67] P. Gacs. On the symmetry of algorithmic information. *Soviet Math. Dokl*, 15(1477-1780):1–3, 1974.

[68] G. Gadaleta, G. Pepe, G. De Candia, and C. Quagliariello. Sibisa. E. & Saccone, C.(1989). The complete nucleotide sequence of the Rattus norvegicus mitochondrial genome: cryptic signals revealed by comparative analysis between vertebrates. *Journal of Molecular Evolution*, 28:497–516.

[69] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. An Empirical Study of Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem. *Principles and Practice of Constraint Programming*, pages 179–193, 1996.

[70] J. Giles. Preprint analysis quantifies scientific plagiarism. *Nature 444*, pages 524–525, 2006.

[71] D. Gitchell and N. Tran. Sim: a utility for detecting similarity in computer programs. *Technical Symposium on Computer Science Education*, pages 266–270, 1999.

[72] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[73] C. Gomes. *Constraint and Integer Programming: Toward a Unified Methodology*, chapter Complete randomized backtrack search, pages 233–283. Kluwer Academics, 2003.

[74] C. Gomes, B. Selman, and N. Crato. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1–2):67–100, 2000.

[75] C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the 15th National Conference onon Artificial Intelligence*, pages 431–437, 1998.

[76] C.P. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. *Principles and Practice of Constraint Programming*, pages 121–135, 1997.

[77] C.P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning*, 24(1):67–100, 2000.

[78] C.P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437, 1998.

[79] D. Grune and M.H. Vakgroep. Detecting copied submissions in computer science workshops. *Informatica Faculteit Wiskunde & Informatica, Vrije Universiteit*, 1989.

[80] J.K. Harris. Plagiarism in computer science courses. *Proceedings of the conference on Ethics in the computer age*, pages 133–135, 1994.

[81] M. Hart. The Gutemberb Project [Online]. Available: http://www.gutenberg.org.

[82] B.M. Hill. A Simple General Approach to Inference About the Tail of a Distribution. *The Annals of Statistics*, 3(5):1163–1174, 1975.

[83] J.H. Holland. *Adaptation in natural and artificial systems*. MIT Press Cambridge, MA, USA, 1992.

[84] S. Horai, Y. Satta, K. Hayasaka, R. Kondo, T. Inoue, T. Ishida, S. Hayashi, and N. Takahata. Man's place in hominoidea revealed by mitochondrial DNA genealogy. *Journal of Molecular Evolution*, 35(1):32–43, 1992.

[85] D. Horowitz. Generating rhythms with genetic algorithms. *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2) table of contents*, 1994.

[86] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A Bayesian approach to tackling hard computational problems. *Proceedings the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, 2001.

[87] D. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE*, volume 40, 401952.

[88] RL Hughey. A survey and comparison of methods for estimating extreme right tail-area quantiles. *Communications in statistics. Theory and methods*, 20(4):1463–1496, 1991.

[89] N.C. Ingle. Language Identification Table. *Technical Translation International*, 1980.

[90] R. Irving. Plagiarism Detection: Experiences and Issues. *JISC Fifth Information Strategies Conference, Focus on Access and Security*, 2000.

[91] A. C. Calgary J. Cleary, I. Witten. Data compression using adaptive coding and partial matching. *IEEE Trans. Communications*, 32:396–402, 1984.

[92] B.L. Jacob. Composing with genetic algorithms. *Proceedings of the 1995 International Computer Music Conference*, pages 452–455, 1995.

[93] E.L. Jones. Metrics based plagiarism monitoring. *Proceedings of the Sixth Annual CCSC Northeastern Conference, Middlebury, Vermont*, pages 1–8, 2001.

[94] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the 18th American Association on Artificial Intelligence*, pages 674–681, 2002.

[95] S. Kirkpatrick, CD Gelatt Jr, and MP Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671, 1983.

[96] A. Kolmogorov. Logical basis for information theory and probability theory. *Information Theory, IEEE Transactions on*, 14(5):662–664, 1968.

[97] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7, 1965.

[98] S.R. Kosaraju and G. Manzini. Some entropic bounds for Lempel-Ziv algorithms. *Proceedings of the Conference on Data Compression*, 1997.

[99] J.R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. Bradford Books, 1992.

[100] P. Laine and M. Kuuskankare. Genetic algorithms in musical style oriented generation. *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 858–862, 1994.

[101] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Mueller. Efficient backprop. *Lecture Notes in Computer Science*, 1524:5–50, 1998.

[102] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitányi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004.

[103] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi. The Similarity Metric. *Information Theory, IEEE Transactions on*, 50:12, 2004.

[104] M. Li and R. Sleep. Melody Classification using a Similarity Metric Based on Kolmogorov Complexity. *Sound and Music Computing*, 2004.

[105] M. Li and PMB Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

[106] D. Lidov and J. Gabura. A melody writing algorithm using a formal language model. *Computer Studies in the Humanities*, 4(3-4):138–148, 1973.

[107] A. Lindenmayer. Mathematical models for cellular interactions in development. I. Filaments with one-sided inputs. *J Theor Biol*, 18(3):280–99, 1968.

[108] X. Liu, P.M. Pardalos, S. Rajasekaran, and M.G.C. Resende. A grasp for frequency assignment in mobile radio networks. *Mobile networks and computing*, 52:195–201, 2000.

[109] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. In *Proceedings of the 2nd Israel Symposium on the Theory and Computing Systems*, pages 128–133, 1993.

[110] X. Ma and M. Liberman. BITS: A Method for Bilingual Text Search over the Web. *Machine Translation Summit VII*, 1999.

[111] B. Mandelbrot. The Pareto-Lévy Law and the Distribution of Income. *International Economic Review*, 1:79–106, 1960.

[112] B. Mandelbrot. The Variation of Certain Speculative Prices. *The Journal of Business*, 36(4):394–419, 1963.

[113] B.B. Mandelbrot and J.A. Wheeler. The Fractal Geometry of Nature. *American Journal of Physics*, 51:286, 1983.

[114] E. Marchiori. Combining constraint processing and genetic algorithms for constraint satisfaction problems. In *7th International Conference on Genetic Algorithms*, pages 330–337, San Francisco, CA, 1997.

[115] B. Martin. Plagiarism: a misplaced emphasis. *Journal of Information Ethics*, 3(2):36–47, 1994.

[116] J. McCormack. Grammar based music composition. *Complex Systems*, 1996.

[117] M. Mitzenmacher, E. Upfal, et al. *Probability And Computing: an introduction to randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[118] M. Mladenovic and P. Hansen. Variable neighborhood search. *Computers in Operations Research*, 24:1097–1100, 1997.

[119] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. *Handbook of Metaheuristics*, pages 105–144, 2003.

[120] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[121] M.A. Nilsson, A. Gullberg, A.E. Spotorno, U. Arnason, and A. Janke. Radiation of Extant Marsupials After the K/T Boundary: Evidence from Complete Mitochondrial Genomes. *Journal of Molecular Evolution*, 57:3–12, 2003.

[122] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

[123] M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico. Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines*, 4(1):67–93, 2003.

[124] A. Ortega, R.S. Alfonso, and M. Alfonseca. Automatic composition of music by means of grammatical evolution. *Proceedings of the 2002 conference on APL: array processing languages: lore, problems, and applications*, pages 148–155, 2002.

[125] A. Ortega, A.A. Dalhoum, and M. Alfonseca. Grammatical evolution to design fractal curves with a given dimension. *IBM Journal of Research and Development*, 47(4):483–493, 2003.

[126] L. Paninski. Estimation of Entropy and Mutual Information. *Neural Computation*, 15(6):1191–1253, 2003.

[127] S. Papert. Mindstorms: children, computers, and powerful ideas. 1980. Basic Books, New York.

[128] M. Prais and C.C. Ribeiro. Parameter variation in grasp procedures. *Investigación Operativa*, 9:1–20, 2000.

[129] M. Prais and C.C. Ribeiro. Reactive grasp: an application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

[130] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016–1038, 2002.

[131] P. Prosser. An Empirical Study of Phase Transitions in Binary Constraint Satisfaction Problems. *Artificial Intelligence*, 81(1-2):81–109, 1996.

[132] M.O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980.

[133] I. Rechenberg. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. *Stuttgart: Fromman-Holzboog*, 1973.

[134] P. Resnik. Parallel strands: A preliminary investigation into mining the web for bilingual text. In *Proceedings of the Third Conference of the Association for Machine Translation in the Americas on Machine Translation and the Information Soup*, pages 72–82, 1998.

[135] P. Resnik. *Mining the Web for bilingual text*. Association for Computational Linguistics Morristown, NJ, USA, 1999.

[136] P. Resnik and N.A. Smith. The Web as a parallel corpus. *Computational Linguistics*, 29(3):349–380, 2003.

[137] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[138] F. Rossi, C. Petrie, V. Dhar, and L.C. Aiello. On the Equivalence of Constraint Satisfaction Problems. *ECAI'90: Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, 1990.

[139] Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: A dynamic programming approach. *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002)*, pages 573–586, 2002.

[140] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1998.

[141] BM Smith and ME Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1):155–181, 1996.

[142] NA Smith. Detection of translational equivalence, 2001 Undergraduate Honors Thesis. *University of Maryland, College Park*.

[143] N.A. Smith. From words to corpora: recognizing translation. *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 95–102, 2002.

[144] R.J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7(1):1–22, 1964.

[145] R. Solovay and V. Strassen. A Fast Monte-Carlo Test for Primality. *SIAM Journal on Computing*, 6:84, 1977.

[146] I. H. Witten T. C. Bell, J. G. Cleary. *Text compression*. Prentice Hall,

[147] P. Van Hentenryck, V. Saraswat, and Y. Deville. Constraint processing in cc (FD). *Constraint Programming: Basics and Trends, A. Podelski (Ed.), LNCS*, 910, 1991.

[148] A. van Moorsel and K. Wolter. Analysis and algorithms for restart. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems*, pages 195–204, 2004.

[149] T. Walsh. Search in a small world. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1172–1177, 1999.

[150] N. Weymaere and J. P. Martens. On the initialization and optimization of multilayer perceptrons. *IEEE Transactions on Neural Networks*, 5:738–751, 1994.

[151] G. Whale. Identification of Program Similarity in Large Populations. *The Computer Journal*, 33(2):140, 1990.

[152] W. Willinger, M.S. Taqqu, W.E. Leland, and D.V. Wilson. Self-Similarity in High-Speed Packet Traffic: Analysis and Modeling of Ethernet Traffic Measurements. *Statistical Science*, 10(1):67–85, 1995.

[153] M.J. Wise. Detection of similarities in student programs: YAP'ing may be preferable to plague'ing. *ACM SIGCSE Bulletin*, 24(1):268–271, 1992.

[154] M. Yamaguti, M. Hata, and J. Kigami. Mathematics of fractals. *AMS Transl. Math. Monographs*, 1997.

[155] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 23(3):337–343, 1977.