Statistical Inference Through Data Compression

Rudi Cilibrasi

Statistical Inference Through Data Compression

ILLC Dissertation Series DS-2007-01

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam
phone: +31-20-525 6051
fax: +31-20-525 5206
e-mail: illc@science.uva.nl
homepage: http://www.illc.uva.nl/

Statistical Inference Through Data Compression

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.mr. P.F. van der Heijden
ten overstaan van een door het college voor
promoties ingestelde commissie, in het openbaar
te verdedigen in de Aula der Universiteit
op vrijdag 23 februari 2007, te 10.00 uur

door

Rudi Langston Cilibrasi

geboren te Brooklyn, New York, Verenigde Staten

Promotiecommissie:

Promotor:	Prof.dr.ir. P.M.B. Vitányi
Co-promotor:	Dr. P.D. Grünwald
Overige leden:	Prof.dr. P. Adriaans Prof.dr. R. Dijkgraaf Prof.dr. M. Li Prof.dr. B. Ryabko Prof.dr. A. Siebes Dr. L. Torenvliet

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Printed and bound by PRINTPARTNERS IPSKAMP.

ISBN: 90–6196–540–3

My arguments will be open to all, and may be judged of by all.

– Publius

Contents

List of Figures

List of Tables

Acknowledgements

The author would like to thank first and foremost Dr. Paul Vitányi for his elaborate feedback and tremendous technical contributions to this work. Next I thank Dr. Peter Grünwald for ample feedback. I also thank my colleagues John Tromp and Ronald de Wolf. I thank my friends Dr. Kaihsu Tai and Ms. Anna Lissa Cruz for extensive feedback and experimental inputs. This thesis is dedicated to my grandparents, Edwin and Dorothy, for their equanimity and support. It is further dedicated in spirit to the memories of my mother, Theresa, for her compassion, and to my father, Salvatore for his deep insight and foresight.

This work was supported in part by the Netherlands BSIK/BRICKS project, and by NWO project 612.55.002, and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778.

Papers on Which the Thesis is Based

Chapter 2 is introductory material, mostly based on M. Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer–Verlag, New York, second edition, 1997.

Chapter 3 is based on R. Cilibrasi and P. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523-1545, 2005, as well as M.Li, X. Chen, X. Li, B. Ma and P. Vitányi. The similarity metric. *IEEE Trans. Information Theory*, 50(12):3250–3264, 2004. Section 3.6 is based on unpublished work by R. Cilibrasi.

Chapter 4 is based on R. Cilibrasi and P.M.B. Vitányi. A new quartet tree heuristic for hierarchical clustering, IEEE/ACM Trans. Comput. Biol. Bioinf., Submitted. Presented at the *EU-PASCAL Statistics and Optimization of Clustering Workshop*, London, 2005, `http://arxiv.org/abs/cs.DS/0606048`

Chapter 5 is based on unpublished work by R. Cilibrasi.

Chapter 6 is based on

R. Cilibrasi and P. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523-1545, 2005;

R. Cilibrasi, P.M.B. Vitányi, and R. de Wolf. Algorithmic clustering of music based on string compression. *Computer Music Journal*, pages 49-67. A preliminary version appeared as

R. Cilibrasi, R. de Wolf, P. Vitányi, Algorithmic clustering of music, *Proc IEEE 4th International Conference on Web Delivering of Music* (WEDELMUSIC 2004), IEEE Comp. Soc. Press, 2004, 110-117.

This work was reported in among others: "Software to unzip identity of unknown composers, *New Scientist*, 12 April 2003, by Hazel Muir. "Software sorts tunes," *Technology Research News*, April 23/30, 2003, by Kimberly Patch; and "Classer musiques, langues, images, textes et genomes," *Pour La Science*, 317(March 2004), 98–103, by Jean-Paul Delahaye , (Pour la Science = Edition francaise de Scientific American).

Chapter 7 is based on

R. Cilibrasi, P.M.B. Vitanyi, Automatic meaning discovery using Google, http://xxx.lanl.gov/abs/cs.CL/0412098 (2004); followed by conference versions

R. Cilibrasi and P.M.B. Vitányi, Automatic Extraction of Meaning from the Web, *2006 IEEE International Symposium on Information Theory (ISIT 2006)*, Seattle, 2006; and

R. Cilibrasi, P.M.B. Vitányi, Similarity of objects and the meaning of words, *Proc. 3rd Conf. Theory and Applications of Models of Computation (TAMC)*, 15-20 May, 2006, Beijing, China. Lecture Notes in Computer Science, Vol. 3959, Jin-Yi Cai, S. Barry Cooper, and Angsheng Li (Eds.), 2006; to the journal version

R. Cilibrasi and P.M.B. Vitányi. The Google similarity distance, *IEEE Transactions on Knowledge and Data Engineering*, To appear.

The supporting experimental data for the binary classification experimental comparison with WordNet can be found at
http://www.cwi.nl/ ~cilibrar/googlepaper/appendix.eps

This work was reported in, among others, "A search for meaning," *New Scientist*, 29 January 2005, p.21, by Duncan Graham-Rowe; on the Web in "Deriving semantics meaning from Google results," *Slashdot— News for nerds, Stuff that matters*, Discussion in the Science section, 29 January, 2005.

Chapter 8 is based on T. Roos, T. Heikkila, R. Cilibrasi, and P. Myllymäki. Compression-based stemmatology: A study of the legend of St. Henry of Finland, 2005. HIIT technical report, http://cosco.hiit.fi/Articles/hiit-2005-3.eps

Chapter 9 is based on unpublished work by R. Cilibrasi.

Chapter 10 describes the CompLearn system, a general software tool to apply the ideas in this Thesis, written by R. Cilibrasi, and explains the reasoning behind it; see http://complearn.org/ for more information.

Chapter 1

Introduction

> But certainly for the present age, which prefers the sign to the thing signified, the copy to the original, representation to reality, the appearance to the essence... illusion only is sacred, truth profane. Nay, sacredness is held to be enhanced in proportion as truth decreases and illusion increases, so that the highest degree of illusion comes to be the highest degree of sacredness. –Feuerbach, Preface to the second edition of *The Essence of Christianity*

1.1 Overview of this thesis

This thesis concerns a remarkable new scientific development that advances the state of the art in the field of data mining, or searching for previously unknown but meaningful patterns in fully or semi-automatic ways. A substantial amount of mathematical theory is presented as well as very many (though not yet enough) experiments. The results serve to test, verify, and demonstrate the power of this new technology. The core ideas of this thesis relate substantially to data compression programs. For more than 30 years, data compression software has been developed and significantly improved with better models for almost every type of file. Until recently, the main driving interests in such technology were to economize on disk storage or network data transmission costs. A new way of looking at data compressors and machine learning allows us to use compression programs for a wide variety of problems.

In this thesis a few themes are important. The first is the use of data compressors in new ways. The second is a new tree visualization technique. And the third is an information-theoretic connection of a web search engine to the data mining system. Let us examine each of these in turn.

1.1.1 Data Compression as Learning

The first theme concerns the statistical significance of compressed file sizes. Most computer users realize that there are freely available programs that can compress text files to about one quarter their original size. The less well known aspect of data compression is that combining

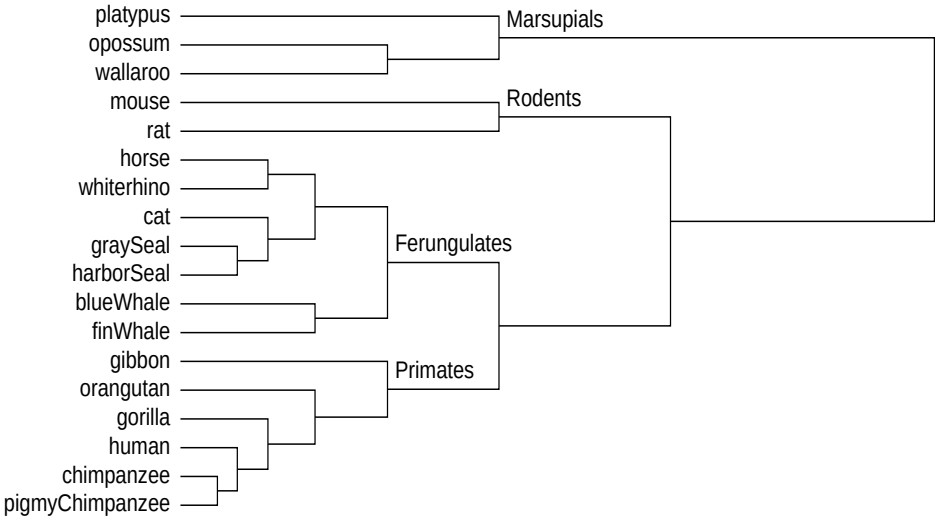

Figure 1.1: The evolutionary tree built from complete mammalian mtDNA sequences of 24 species, using the NCD matrix of Figure 4.14 on page 70 where it was used to illustrate a point of hierarchical clustering versus flat clustering. We have redrawn the tree from our output to agree better with the customary phylogeny tree format. The tree agrees exceptionally well with the NCD distance matrix: $S(T) = 0.996$.

two or more files together to create a larger single conglomerate *archive file* prior to compression often yields better compression in aggregate. This has been used to great advantage in widely popular programs like `tar` or `pkzip`, combining archival and compression functionality. Only in recent years have scientists begun to appreciate the fact that compression ratios signify a great deal of important statistical information. All of the experiments in this thesis make use of a group of compressible objects. In each case, the individual compressed sizes of each object are calculated. Then, some or all possible pairs of objects are combined and compressed to yield pairwise compressed sizes. It is the tiny variations in the pairwise compressed sizes that yields the surprisingly powerful results of the following experiments. The key concept to realize is that if two files are very similar in their contents, then they will compress much better when combined together prior to compression, as compared to the sum of the size of each separately compressed file. If two files have little or nothing in common, then combining them together would not yield any benefit over compressing each file separately.

Although the principle is intuitive to grasp, it has surprising breadth of applicability. By using even the simplest string-matching type compression made in the 1970's it is possible to construct evolutionary trees for animals fully automatically using files containing their mitochondrial gene sequence. One example is shown in Figure 4.12. We first construct a matrix of pairwise distances between objects (files) that indicate how similar they are. These distances are based on comparing compressed file sizes as described above. We can apply this to files of widely different types, such as music pieces or genetic codes as well as many other specialized domains. In Figure 4.12, we see a tree constructed from the similarity distance matrix based on the mitochondrial DNA of several species. The tree is constructed so that species with "similar" DNA are "close by" in the

tree. In this way we may lend support to certain evolutionary theories.

Although simple compressors work, it is also easy to use the most advanced modern compressors with the theory presented in this thesis; these results can often be more accurate than simpler compressors in a variety of particular circumstances or domains. The main advantage of this approach is its robustness in the face of strange or erroneous data. Another key advantage is the simplicity and ease of use. This comes from the generality of the method: it works in a variety of different application domains and when using general-purpose compressors it becomes a general-purpose inference engine. Throughout this thesis there is a focus on coding theory and data compression, both as a theoretical construct as well as practical approximations thereof through actual data compression programs in current use. There is a connection between a particular code and a probability distribution and this simple theoretical foundation allows one to use data compression programs of all types as statistical inference engines of remarkable robustness and generality. In Chapter 3, we describe the *Normalized Compression Distance* (NCD), which formalizes the ideas we have just described. We report on a plethora of experiments in Chapter 6 showing applications in a variety of interesting problems in data mining using gene sequences, music, text corpora, and other inputs.

1.1.2 Visualization

Custom open source software has been written to provide powerful new visualization capabilities. The *CompLearn* software system (Chapter 10) implements our theory and with it experiments of two types may be carried out: classification or clustering. Classification refers to the application of discrete labels to a set of objects based on a set of examples from a human expert. Clustering refers to arrangement of objects into groups without prior training or influence by a human expert. In this thesis we deal primarily with hierarchical or nested clustering in which a group of objects is arranged into a sort of binary tree. This clustering method is called the *quartet method* and will be discussed in detail later.

In a nutshell, the quartet method is a way to determine a best matching tree given some data that is to be understood in a hierarchical cluster. It is called the quartet method because it is based on the smallest unrooted binary tree, which happens to be two pairs of two nodes for a total of four nodes comprising the quartet. It adds up many such small binary trees together to evaluate a big tree and then adjusts the tree according to the results of the evaluation. After a time, a best fitting tree is declared and the interpretation of the experimental results is possible. The compression-based algorithms output a matrix of pairwise distances between objects. Because such a matrix is hard to interpret, we try to extract some of its essential features using the quartet method. This results in a tree optimized so that similar objects with small distances are placed nearby each other. The trees given in Figures 1.1, 1.2, and 1.3 (discussed below) have all been constructed using the quartet method.

The quartet tree search is non-deterministic. There are compelling theoretical reasons to suppose that the general quartet tree search problem is intractable to solve exactly for every case. But the method used here tries instead to approximate a solution in a reasonable amount of time, sacrificing accuracy for speed. It also makes extensive use of random numbers, and so there is sometimes variation in the results that the tree search produces. We describe the quartet tree

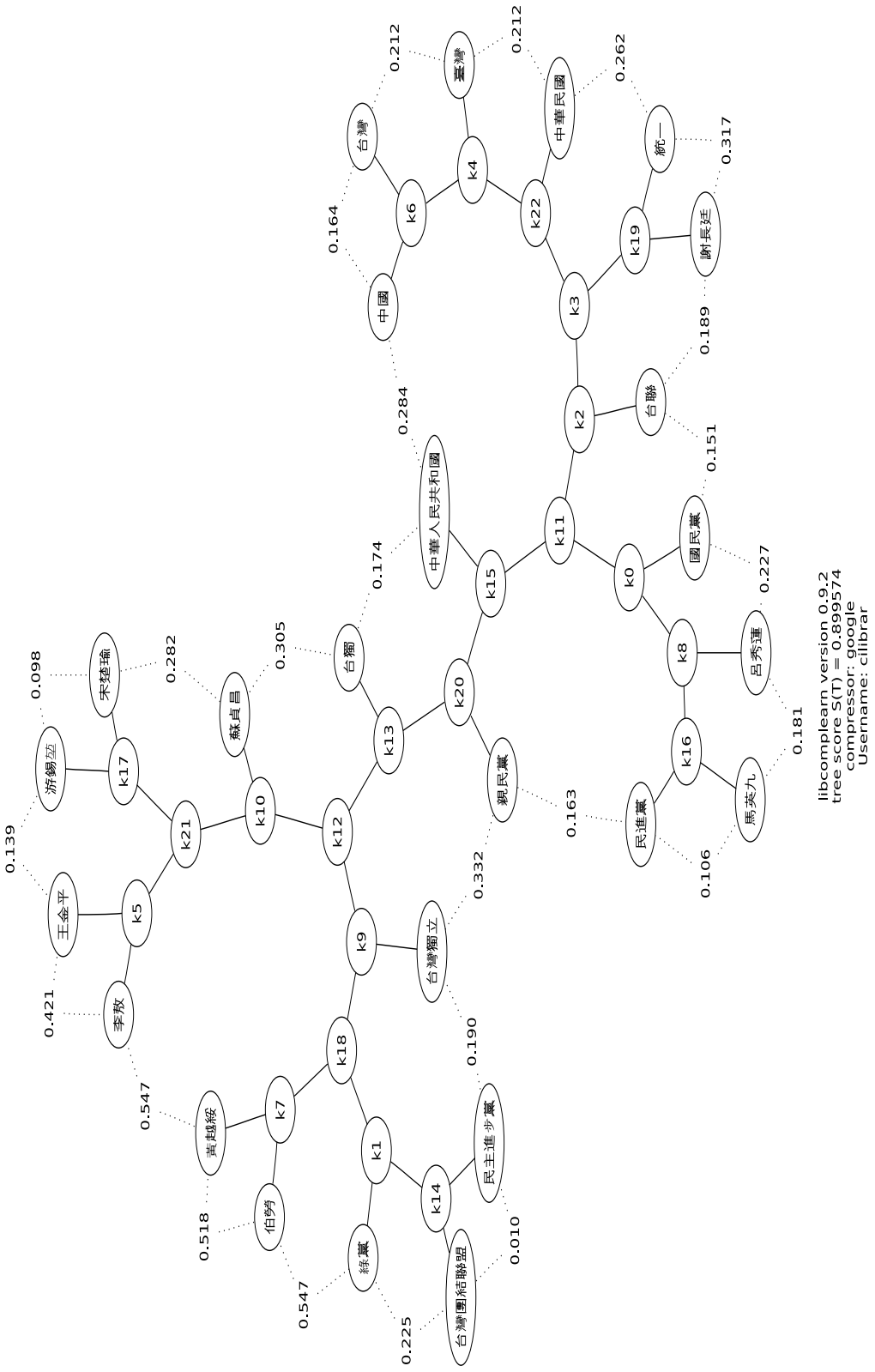

Figure 1.2: Several people's names, political parties, regions, and other Chinese names.

method in detail in Chapter 4. In Chapter 6 we show numerous trees based on applying the quartet method and the NCD to a broad spectrum of input files in a wide array of domains.

1.1.3 Learning From the Web

It is possible to use coding theory to connect the compression approach to the web with the help of a search engine index database. By using a simple formula based on logarithms we can find "compressed sizes" of search terms. This was used in the Chinese tree in Figure 1.2. The tree of Nobel prize winning authors in Figure 1.3 was also made this way. As in the last example, a distance matrix is made, but this time with Google providing page count statistics that are converted to codelengths for use in the distance matrix calculations. We can see English and American writers clearly separated in the tree, as well as many other defensible placements. Another example using prime numbers with Google is in Chapter 7, page 128.

Throughout this thesis the reader will find ample experiments demonstrating the machine learning technology. There are objective experiments based on pure statistics using true data compressors and subjective experiments using statistics from web pages as well. There are examples taken from genetics, linguistics, literature, radio astronomy, optical character recognition, music, and many more diverse areas. Most of the experiments can be found in Chapters 4, 6, and 7.

1.1.4 Clustering and Classification

The examples given above all dealt with clustering. It is also interesting to consider how we can use NCD to solve classification problems. Classification is the task of assigning labels to unknown test objects given a set of labeled training objects from a human expert. The goal is to try to learn the underlying patterns that the human expert is displaying in the choice of labellings shown in the training objects, and then to apply this understanding to the task of making predictions for unknown objects that are in some sense consistent with the given examples. Usually the problem is reduced to a combination of binary classification problems, where all target labels along a given dimension are either 0 or 1. In Chapter 5 we discuss this problem in greater detail, we give some information about a popular classification engine called the Support Vector Machine (SVM), and we connect the SVM to the NCD to create robust binary classifiers.

1.2 Gestalt Historical Context

Each of the three key ideas (compression as learning, quartet tree visualization, and learning from the web) have a common thread: all of them serve to increase the generality and practical robustness of the machine intelligence compared to more traditional alternatives. This goal is not new and has already been widely recognized as fundamental. In this section a brief and subjective overview of the recent history of artificial intelligence is given to provide a broader context for this thesis.

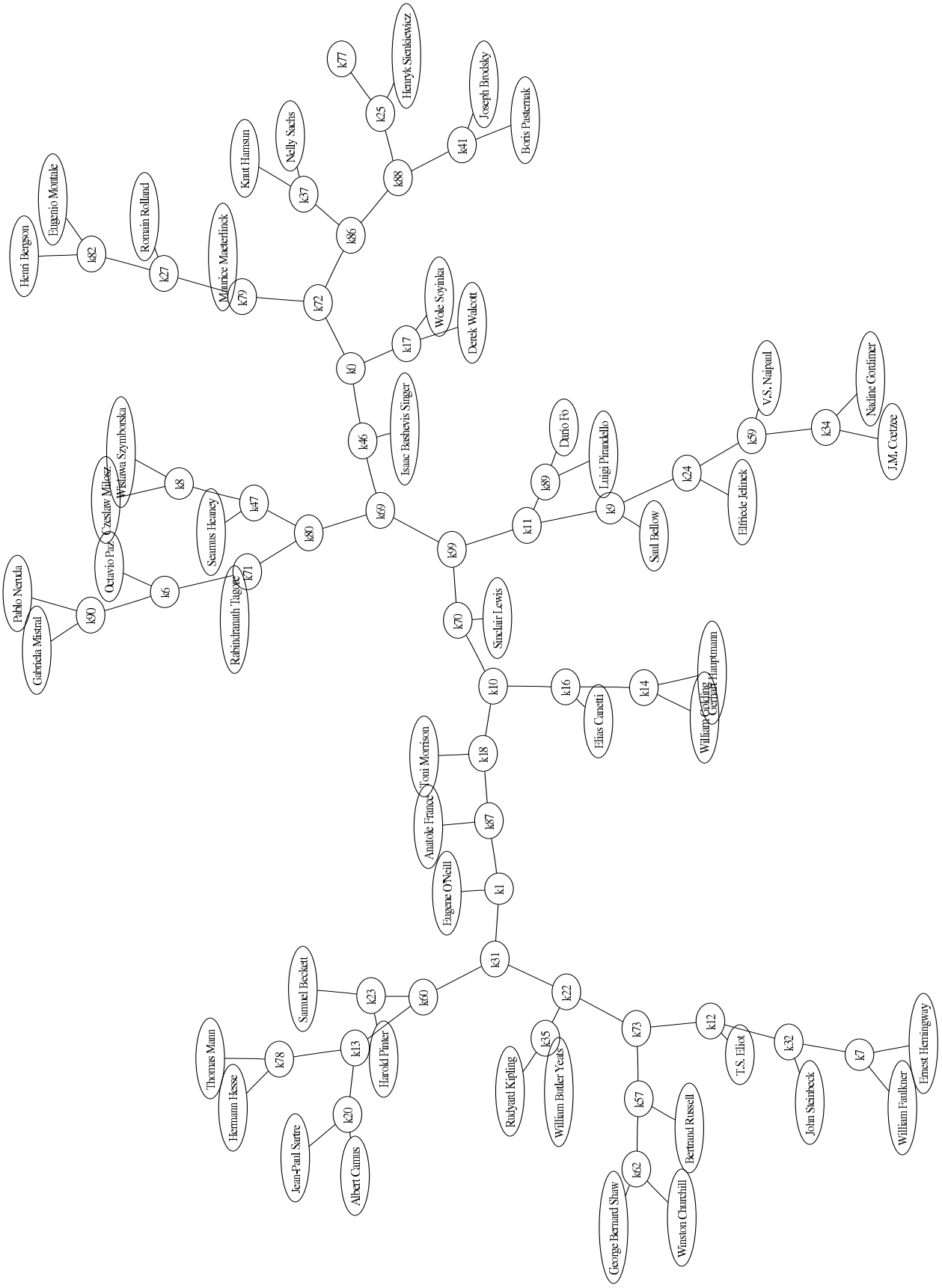

Figure 1.3: 102 Nobel prize winning writers using CompLearn and NGD; S(T)=0.905630 (part 3).

In the beginning, there was the idea of artificial intelligence. As circuit miniaturization took off in the 1970's, people's imaginations soared with ideas of a new sort of machine with virtually unlimited potential: a (usually humanoid) metal automaton with the capacity to perform intelligent work and yet ask not one question out of the ordinary. A sort of ultra-servant, made able to reason as well as man in most respects, yet somehow reasoning in a sort of rarefied form whereby the more unpredictable sides of human nature are factored out. One of the first big hurdles came as people tried to define just what intelligence was, or how one might codify knowledge in the most general sense into digital form. As Levesque and Brachman famously observed [73], reasoning and representation are hopelessly intertwined, and just what intelligence is depends very much on just who is doing the talking.

Immediately upon settling on the question of intelligence one almost automatically must grapple with the concept of language. Consciousness and intelligence is experienced only internally, yet the objects to which it applies are most often external to the self. Thus there is at once the question of communication and experience and this straight-away ends any hope of perfect answers. Most theories on language are not theories in the formal sense [14]. A notable early exception is Quine's famous observation that language translation is necessarily a dicey subject: for although you might collect very many pieces of evidence suggesting that a word means "X" or "Y", you can never collect a piece of evidence that ultimately confirms that your understanding of the word is "correct" in any absolute sense. In a logical sense, we can never be sure that the meaning of a word is as it was meant, for to explain any word we must use other words, and these words themselves have only other words to describe them, in an interminable web of ontological disarray. Kantian empiricism leads us to pragmatically admit we have only the basis of our own internal experience to ground our understanding at the most basic level, and the mysterious results of the reasoning mind, whatever that might be.

It is without a doubt the case that humans throughout the world develop verbal and usually written language quite naturally. Recent theories by Smale [38] have provided some theoretical support for empirical models of language evolution despite the formal impossibility of absolute certainty. Just the same it leaves us with a very difficult question: how do we make bits think?

Some twenty years later, progress has been bursty. We have managed to create some amazingly elegant search and optimization techniques including simplex optimization, tree search, curve-fitting, and modern variants such as neural networks or support vector machines. We have built computers that can beat any human in chess, but we cannot yet find a computer smart enough to walk to the grocery store to buy a loaf of bread. There is clearly a problem of overspecialization in the types of successes we have so far enjoyed in artificial intelligence. This thesis explores my experience in charting this new landscape of concepts via a combination of pragmatic and principled techniques. It is only with the recent explosion in internet use and internet writing that we can now begin to seriously tackle these problems so fundamental to the original dream of artificial intelligence.

In recent years, we have begun to make headway in defining and implementing universal prediction, arguably the most important part of artificial intelligence. Most notable is Solomonoff prediction [105], and the more practical analogs by Ryabko and Astola [98] using data compression.

In classical statistical settings, we typically make some observations of a natural (or at the

very least, measurable) phenomenon. Next, we use our intuition to "guess" which mathematical model might best apply. This process works well for those cases where the guesser has a good model for the phenomenon under consideration. This allows for at least two distinct modes of freedom: both in the choice of models, and also in the choice of criteria supporting "goodness".

In the past the uneasy compromise has been to focus attention firstly on those problems which are most amenable to exact solution, to advance the foundation of exact and fundamental science. The next stage of growth was the advent of machine-assisted exact sciences, such as the now-famous four-color proof that required input (by hand!) of 1476 different graphs for computer verification (by a complicated program) that all were colorable before deductive extension to the most general case in the plane [2]. After that came the beginning of modern machine learning, based on earlier ideas of curve fitting and least-squares regression. Neural networks, and later support vector machines, gave us convenient learning frameworks in the context of continuous functions. Given enough training examples, the theory assured us, the neural network would eventually find the right combination of weightings and multiplicative factors that would miraculously, and perhaps a bit circularly, reflect the underlying meaning that the examples were meant to teach. Just like spectral analysis that came before, each of these areas yielded a whole new broad class of solutions, but were essentially hit or miss in their effectiveness in each domain for reasons that remain poorly understood. The focus of my research has been on the use of data compression programs for generalized inference. It turns out that this modus operandi is surprisingly general in its useful application and yields oftentimes the most expedient results as compared to other more predetermined methods. It is often "one size fits all well enough" and this yields unexpected fruits. From the outset, it must be understood that the approach here is decidedly different than more classical ones, in that we avoid in most ways an exact statement of the problem at hand, instead deferring this until very near the end of the discussion, so that we might better appreciate what can be understood about all problems with a minimum of assumptions.

At this point a quote from Goldstein and Gigerenzer [43] is appropriate:

> What are heuristics? The Gestalt psychologists Karl Duncker and Wolfgang Koehler preserved the original Greek definition of "serving to find out or discover" when they used the term to describe strategies such as "looking around" and "inspecting the problem" (e.g., Duncker, 1935/1945).
>
> For Duncker, Koehler, and a handful of later thinkers, including Herbert Simon (e.g., 1955), heuristics are strategies that guide information search and modify problem representations to facilitate solutions. From its introduction into English in the early 1800s up until about 1970, the term heuristics has been used to refer to useful and indispensable cognitive processes for solving problems that cannot be handled by logic and probability theory (e.g., Polya, 1954; Groner, Groner, & Bischof, 1983). In the past 30 years, however, the definition of heuristics has changed almost to the point of inversion. In research on reasoning, judgment, and decision making, heuristics have come to denote strategies that prevent one from finding out or discovering correct answers to problems that are assumed to be in the domain of probability theory. In this view, heuristics are poor substitutes for computations that are too demanding for

ordinary minds to carry out. Heuristics have even become associated with inevitable cognitive illusions and irrationality.

This author sides with Goldstein and Gigerenzer in the view that sometimes "less is more"; the very fact that things are unknown to the naive observer can sometimes work to his advantage. The recognition heuristic is an important, reliable, and conservative general strategy for inductive inference. In a similar vein, the NCD based techniques shown in this thesis provide a general framework for inductive inference that is robust against a wide variety of circumstances.

1.3 Contents of this Thesis

In this chapter a summary is provided for the remainder of the thesis as well as some historical context. In Chapter 2, an introduction to the technical details and terminology surrounding the methods is given. In chapter 3 we introduce the Normalized Compression Distance (NCD), the core mathematical formula that makes all of these experiments possible, and we establish connections between NCD and other well-known mathematical formulas. In Chapter 4 a tree search system is explained based on groups of four objects at a time, the so-called *quartet method*. In Chapter 5 we combine NCD with other machine learning techniques such as Support Vector Machines. In Chapter 6, we provide a wealth of examples of this technology in action. All experiments in this thesis were done using the CompLearn Toolkit, an open-source general purpose data mining toolkit available for download from the **http://complearn.org/** website. In Chapter 7, we show how to connect the internet to NCD using the Google search engine, thus providing the advanced sort of subjective analysis as shown in Figure 1.2. In Chapter 8 we use these techniques and others to trace the evolution of the legend of Saint Henry. In Chapter 9 we compare CompLearn against another older tree search software system called PHYLIP. Chapter 10 gives a snapshot of the online documentation for the CompLearn system. After this, a Dutch language summary is provided as well as a bibliography, index, and list of papers by R. Cilibrasi.

Chapter 2

Technical Introduction

> The spectacle is the existing order's uninterrupted discourse about itself, its laudatory monologue. It is the self-portrait of power in the epoch of its totalitarian management of the conditions of existence. The fetishistic, purely objective appearance of spectacular relations conceals the fact that they are relations among men and classes: a second nature with its fatal laws seems to dominate our environment. But the spectacle is not the necessary product of technical development seen as a natural development. The society of the spectacle is on the contrary the form which chooses its own technical content. –Guy Debord, *Society of the Spectacle*

This chapter will give an informal introduction to relevant background material, familiarizing the reader with notation and basic concepts but omitting proofs. We discuss strings, languages, codes, Turing Machines and Kolmogorov complexity. This material will be extensively used in the chapters to come. For a more thorough and detailed treatment of all the material including a tremendous number of innovative proofs see [79]. It is assumed that the reader has a basic familiarity with algebra and probability theory as well as some rudimentary knowledge of classical information theory. We first introduce the notions of *finite*, *infinite* and *string of characters*. We go on to discuss basic coding theory. Next we introduce the idea of Turing Machines. Finally, in the last part of the chapter, we introduce Kolmogorov Complexity.

2.1 Finite and Infinite

In the domain of mathematical objects discussed in this thesis, there are two broad categories: finite and infinite. *Finite* objects are those whose extent is bounded. *Infinite* objects are those that are "larger" than any given precise bound. For example, if we perform 100 flips of a fair coin in sequence and retain the results in order, the full record will be easily written upon a single sheet of A4 size paper, or even a business card. Thus, the sequence is finite. But if we instead talk about the list of all prime numbers greater than 5, then the sequence written literally is infinite in extent. There are far too many to write on any given size of paper no matter how big. It is possible, however, to write a *computer program* that could, in principle, generate every prime

number, no matter how large, eventually, given unlimited time and memory. It is important to realize that some objects are infinite in their totality, but can be finite in a potential effective sense by the fact that every finite but a priori unbounded part of them can be obtained from a finite computer program. There will be more to say on these matters later in Section 2.5.

2.2 Strings and Languages

A *bit*, or binary digit, is just a single piece of information representing a choice between one of two alternatives, either 0 or 1.

A character is a symbol representing an atomic unit of written language that cannot be meaningfully subdivided into smaller parts. An alphabet is a set of symbols used in writing a given *language*. A language (in the formal sense) is a set of permissible *strings* made from a given alphabet. A *string* is an ordered list (normally written sequentially) of 0 or more symbols drawn from a common alphabet. For a given alphabet, different languages deem different strings permissible. In English, 26 letters are used, but also the space and some punctuation should be included for convenience, thus increasing the size of the alphabet. In computer files, the underlying base is 256 because there are 256 different states possible in each indivisible atomic unit of storage space, the *byte*. A byte is equivalent to 8 bits, so the 256-symbol alphabet is central to real computers. For theoretical purposes however, we can dispense with the complexities of large alphabets by realizing that we can encode large alphabets into small ones; indeed, this is how a byte can be encoded as 8 bits. A bit is a symbol from a 2-symbol, or binary, alphabet. In this thesis, there is not usually any need for an alphabet of more than two characters, so the notational convention is to restrict attention to the binary alphabet in the absence of countervailing remarks. Usually we encode numbers as a sequence of characters in a fixed radix format at the most basic level, and the space required to encode a number in this format can be calculated with the help of the logarithm function. The logarithm function is always used to determine a coding length for a given number to be encoded, given a probability or integer range. Similarly, it is safe for the reader to assume that all log's are taken base 2 so that we may interpret the results in bits.

We write Σ to represent the alphabet used. We usually work with the binary alphabet, so in that case $\Sigma = \{0,1\}$. We write Σ^* to represent the space of all possible strings including the empty string. This notation may be a bit unfamiliar at first, but is very convenient and is related to the well-known concept of *regular expressions*. Regular expressions are a concise way of representing formal languages as sets of strings over an alphabet. The curly braces represent a *set* (to be used as the alphabet in this case) and the $*$ symbol refers to the *closure* of the set; By *closure* we mean that the symbol may be repeated 0, 1, 2, 3, 5, or any number of times. By definition, $\{0,1\}^* = \bigcup_{n\geq 0}\{0,1\}^n$. It is important to realize that successive symbols need not be the same, but could be. Here we can see that the number of possible binary strings is infinite, yet any individual string in this class must itself be finite. For a string x, we write $|x|$ to represent the length, measured in symbols, of that string.

2.3 The Many Facets of Strings

Earlier we said that a string is a sequence of symbols from an alphabet. It is assumed that the symbols in Σ have a natural or at least conventional ordering. From this we may inductively create a rule that allows us to impose an ordering on all strings that are possible in Σ^* in the conventional way: use length first to bring the shorter strings as early as possible in the ordering, and then use the leftmost different character in any two strings to determine their relative ordering. This is just a generalized restatement of the familiar alphabetical or lexicographic ordering. It is included here because it allows us to associate a positive integer ordering number with each possible string. The empty string, ε, is the first string in this list. The next is the string 0, and the next 1. After that comes 00, then 01, then 10, then 11, then 000, and so on *ad nauseaum*. Next to each of these strings we might list their lengths as well as their ordering-number position in this list as follows:

x	$\lvert x\rvert$	$\text{ORD}(x)$	
ε	0	1	
0	1	2	
1	1	3	
00	2	4	
01	2	5	
10	2	6	... and so on forever ...
11	2	7	
000	3	8	
001	3	9	
010	3	10	
011	3	11	
100	3	12	

Here there are a few things to notice. First is that the second column, the length of x written $|x|$, is related to the $\text{ORD}(x)$ by the following relationship:

$$\log(\text{ORD}(x)) \leq |x| \leq \log(\text{ORD}(x)) + 1. \tag{2.3.1}$$

Thus we can see that the variable x can be interpreted polymorphically: as either a literal string of characters having a particular sequence and length or instead as an integer in one of two ways: either by referring to its length using the $|\cdot|$ symbol, or by referring to its ordinal number using $\text{ORD}(x)$. All of the mathematical functions used in this thesis are monomorphic in their argument types: each argument can be either a number (typically an integer) or a string, but not both. Thus without too much ambiguity we will sometimes leave out the ORD symbol and just write x and rely on the reader to pick out the types by their context and usage. Please notice that x can either stand for the string x or the number $\text{ORD}(x)$, but never for the length of x, which we always explicitly denote as $|x|$.

2.4 Prefix Codes

A binary string y is a *proper prefix* of a binary string x if we can write $x = yz$ for $z \neq \varepsilon$. A set $\{x, y, \ldots\} \subseteq \{0,1\}^*$ is *prefix-free* if no element is a proper prefix of any other. A prefix-free set can be used to define a *prefix code*. Formally, a prefix code is defined by a *decoding function* D, which is a function from a prefix free set to some arbitrary set $\mathcal{X}$. The elements of the prefix free set are called *code words*. The elements of $\mathcal{X}$ are called *source words*. If the inverse D^{-1} of D exists, we call it the *encoding function*. An example of a prefix code, that is used later, encodes a source word $x = x_1 x_2 \ldots x_n$ by the code word

$$\overline{x} = 1^n 0 x.$$

Here $\mathcal{X} = \{0,1\}^*$, $D^{-1}(x) = \overline{x} = 1^n 0 x$. This prefix-free code is called *self-delimiting*, because there is a fixed computer program associated with this code that can determine where the code word $\overline{x}$ ends by reading it from left to right without backing up. This way a composite code message can be parsed in its constituent code words in one pass by a computer program. [1]

In other words, a prefix code is a code in which no codeword is a prefix of another codeword. Prefix codes are very easy to decode because codeword boundaries are directly encoded along with each datum that is encoded. To introduce these, let us consider how we may combine any two strings together in a way that they could be later separated without recourse to guessing. In the case of arbitrary binary strings x, y, we cannot be assured of this prefix condition: x might be 0 while y was 00 and then there would be no way to tell the original contents of x or y given, say, just xy. Therefore let us concentrate on just the x alone and think about how we might augment the natural literal encoding to allow for prefix disambiguation. In real languages on computers, we are blessed with whitespace and commas, both of which are used liberally for the purpose of separating one number from the next in normal output formats. In a binary alphabet our options are somewhat more limited but still not too bad. The simplest solution would be to add in commas and spaces to the alphabet, thus increasing the alphabet size to 4 and the coding size to 2 bits, doubling the length of all encoded strings. This is a needlessly heavy price to pay for the privilege of prefix encoding, as we will soon see. But first let us reconsider another way to do it in a bit more than double space: suppose we preface x with a sequence of $|x|$ 0's, followed by a 1, followed by the literal string x. This then takes one bit more than twice the space for x and is even worse than the original scheme with commas and spaces added to the alphabet. This is just the scheme discussed in the beginning of the section. But this scheme has ample room for improvement: suppose now we adjust it so that instead of outputting all those 0's at first in unary, we instead just output a number of zeros equal to

$$\lceil \log(|x|) \rceil,$$

then a 1, then the binary number $|x|$ (which satisfies $|x| \leq \lceil \log x \rceil + 1$, see Eq. (2.3.1)), then x literally. Here, $\lceil \cdot \rceil$ indicates the ceiling operation that returns the smallest integer not less than

[1] This desirable property holds for every prefix-free encoding of a finite set of source words, but not for every prefix-free encoding of an infinite set of source words. For a single finite computer program to be able to parse a code message the encoding needs to have a certain uniformity property like the $\overline{x}$ code.

its argument. This, then, would take a number of bits about

$$2\lceil \log\log x\rceil + \lceil \log x\rceil + 1,$$

which exceeds $\lceil \log x\rceil$, the number of bits needed to encode x literally, only by a logarithmic amount. If this is still too many bits then the pattern can be repeated, encoding the first set of 0's one level higher using the system to get

$$2\lceil \log\log\log x\rceil + \lceil \log\log x\rceil + \lceil \log x\rceil + 1.$$

Indeed, we can "dial up" as many logarithms as are necessary to create a suitably slowly-growing composition of however many log's are deemed appropriate. This is sufficiently efficient for all purposes in this thesis and provides a general framework for converting arbitrary data into prefix-free data. It further allows us to compose any number of strings or numbers for any purpose without restraint, and allows us to make precise the difficult concept of $K(x,y)$, as we shall see in Section 2.6.4.

2.4.1 Prefix Codes and the Kraft Inequality

Let $\mathcal{X}$ be the set of natural numbers and consider the straightforward non-prefix binary representation with the ith binary string in the length-increasing lexicographical order corresponding to the number i. There are two elements of $\mathcal{X}$ with a description of length 1, four with a description of length 2 and so on. However, there are less binary prefix code words of each length: if x is a prefix code word then no $y = xz$ with $z \neq \varepsilon$ is a prefix code word. Asymptotically there are less prefix code words of length n than the 2^n source words of length n. Clearly this observation holds for arbitrary prefix codes. Quantification of this intuition for countable $\mathcal{X}$ and arbitrary prefix-codes leads to a precise constraint on the number of code-words of given lengths. This important relation is known as the *Kraft Inequality* and is due to L.G. Kraft [60].

2.4.1. LEMMA. *Let $l_1, l_2, \ldots$ be a finite or infinite sequence of natural numbers. There is a prefix code with this sequence as lengths of its binary code words iff*

$$\sum_n 2^{-l_n} \leq 1. \tag{2.4.1}$$

2.4.2 Uniquely Decodable Codes

We want to code elements of some set $\mathcal{X}$ in a way that they can be uniquely reconstructed from the encoding. Such codes are called *uniquely decodable*. Every prefix-code is a uniquely decodable code. On the other hand, not every uniquely decodable code satisfies the prefix condition. Prefix-codes are distinguished from other uniquely decodable codes by the property that the end of a code word is always recognizable as such. This means that decoding can be accomplished without the delay of observing subsequent code words, which is why prefix-codes are also called instantaneous codes. There is good reason for our emphasis on prefix-codes. Namely, it turns out that Lemma 2.4.1 stays valid if we replace "prefix-code" by "uniquely decodable code."

This important fact means that every uniquely decodable code can be replaced by a prefix-code without changing the set of code-word lengths. In this thesis, the only aspect of actual encodings that interests us is their length, because this reflects the underlying probabilities in an associated model. There is no loss of generality in restricting further discussion to prefix codes because of this property.

2.4.3 Probability Distributions and Complete Prefix Codes

A uniquely decodable code is *complete* if the addition of any new code word to its code word set results in a non-uniquely decodable code. It is easy to see that a code is complete iff equality holds in the associated Kraft Inequality. Let $l_1, l_2, \ldots$ be the code words of some complete uniquely decodable code. Let us define $q_x = 2^{-l_x}$. By definition of completeness, we have $\sum_x q_x = 1$. Thus, the q_x can be thought of as *probability mass functions* corresponding to some probability distribution Q for a random variable X. We say Q is the distribution *corresponding* to $l_1, l_2, \ldots$. In this way, each complete uniquely decodable code is mapped to a unique probability distribution. Of course, this is nothing more than a formal correspondence: we may choose to encode outcomes of X using a code corresponding to a distribution q, whereas the outcomes are actually distributed according to some $p \neq q$. But, as we argue below, if X is distributed according to p, then the code to which p corresponds is, in an average sense, the code that achieves optimal compression of X. In particular, every probability mass function p is related to a prefix code, the *Shannon-Fano code*, such that the expected number of bits per transmitted code word is as low as is possible for any prefix code, assuming that a random source X generates the source words x according to $P(X = x) = p(x)$. The Shannon-Fano prefix code encodes a source word x by a code word of length $l_x = \lceil \log 1/p(x) \rceil$, so that the expected transmitted code word length equals $\sum_x p(x) \log 1/p(x) = H(X)$, the entropy of the source X, up to one bit. This is optimal by Shannon's "noiseless coding" theorem [102]. This is further explained in Section 2.7.

2.5 Turing Machines

This section mainly serves as a preparation for the next section, in which we introduce the fundamental concept of *Kolmogorov complexity*. Roughly speaking, the Kolmogorov complexity of a string is the shortest computer program that computes the string, i.e. that prints it, and then halts. The definition depends on the specific computer programming language that is used. To make the definition more precise, we should base it on programs written for *universal Turing machines*, which are an abstract mathematical representation of a general-purpose computer equipped with a general-purpose or *universal* computer programming language.

Universal Computer Programming Languages: Most popular computer programming languages such as C, Lisp, Java and Ruby, are *universal*. Roughly speaking, this means that they must be powerful enough to emulate any other computer programming language: every universal computer programming language can be used to write a compiler for any other programming language, including any other universal programming language. Indeed, this has been done already

a thousand times over with the GNU (Gnu's Not Unix) C compiler, perhaps the most successful open-source computer program in the world. In this case, although there are many different assembly languages in use on different CPU architectures, all of them are able to run C programs. So we can always package any C program along with the GNU C compiler which itself is not more than 100 megabytes in order to run a C program anywhere.

Turing Machines: The *Turing machine* is an abstract mathematical representation of the idea of a computer. It generalizes and simplifies all the many specific types of deterministic computing machines into one regularized form. A Turing machine is defined by a set of rules which describe its behavior. It receives as its input a string of symbols, wich may be thought OF as a "program", and it outputs the result of running that program, which amounts to transforming the input using the given set of rules. Just as there are universal computer languages, there are also universal Turing machines. We say a Turing Machine is universal if it can simulate any other Turing Machine. When such a universal Turing machine receives as input a pair $\langle x, y \rangle$, where x is a formal specification of another Turing machine T_x, it outputs the same result as one would get if one would input the string y to the Turing machine T_x. Just as any universal programming language can be used to emulate any other one, any universal Turing machine can be used to emulate any other one. It may help intuition to imagine any familiar universal computer programming language as a definition of a universal Turing machine, and the runtime and hardware needed to execute it as a sort of real-world Turing machine itself. It is necessary to remove resource constraints (on memory size and input/output interface, for example) in order for these concepts to be thoroughly equivalent theoretically.

Turing machines, formally: A Turing machine consists of two parts: a finite control and a tape. The finite control is the memory (or current state) of the machine. It always contains a single symbol from a finite set Q of possible states. The tape initially contains the program which the Turing machine must execute. The tape contains symbols from the trinary alphabet $A = \{0, 1, B\}$. Initially, the entire tape contains the B (blank) symbol except for the place where the program is stored. The program is a finite sequence of bits. The finite control also is always positioned above a particular symbol on the tape and may move left or right one step. At first, the tape head is positioned at the first nonblank symbol on the tape. As part of the formal definition of a Turing machine, we must indicate which symbol from Q is to be the starting state of the machine. At every time step the Turing machine does a simple sort of calculation by consulting a list of *rules* that define its behavior. The rules may be understood to be a function taking two arguments (the current state and the symbol under the reading head) and returning a Cartesian pair: the action to execute this timestep and the next state to enter. This is to say that the two input arguments are a current state (symbol from Q) of the finite control and a letter from the alphabet A. The two outputs are a new state (also taken from Q) and an *action* symbol taken from S. The set of possible actions is $S = \{0, 1, B, L, R\}$. The 0, 1, and B symbols refer to writing that value below the tape head. The L and R symbols refer to moving left or right, respectively. This function defines the behavior of the Turing machine at each step, allowing it to perform simple actions and run a program on a tape just like a real computer but in a very mathematically simple

way. It turns out that we can choose a particular set of state-transition rules such that the Turing machine becomes *universal* in the sense described above. This simulation is plausible given a moment of reflection on how a Turing Machine is mechanically defined as a sequence of rules governing state transitions etc. The endpoint in this line of reasoning is that a universal Turing Machine can run a sort of Turing Machine simulation system and thereby compute identical results as any other Turing Machine.

Notation: We typically use the Greek letter Φ to represent a Turing machine T as a partially defined function. When the Turing machine T is not clear from the context, we write Φ_T. The function is supposed to take as input a program encoded as a finite binary string and outputs the results of running that program. Sometimes it is convenient to define the function as taking integers instead of strings; this is easy enough to do when we remember that each integer is identified with a given finite binary string given the natural lexicographic ordering of finite strings, as in Section 2.3 The function Φ need only be partially defined; for some input strings it is not defined because some programs do not produce a finite string as output, such as infinite looping programs. We say that Φ is defined only for those programs that halt and therefore produce a definite output. We introduce a special symbol ∞ that represents an abstract object outside the space of finite binary strings and unequal to any of them. For those programs that do not halt we say $\Phi(x) = \infty$ as a shorthand way of indicating this infinite loop: x is thus a non-halting program like the following:

```
x = while true ; do ; done
```

Here we can look a little deeper into the x program above and see that although its runtime is infinite, its definition is quite finite; it is less than 30 characters. Since this program is written in the ASCII codespace, we can multiply this figure by 8 to reach a size of 240 bits.

Prefix Turing Machines: In this thesis we look at Turing Machines whose set of halting programs is prefix free: that is to say that the set of such programs form a prefix code (Section 2.4), because no halting program is a prefix of another halting program. We can realize this by slightly changing the definition of a Turing machine, equipping it with a one-way input or 'data' tape, a separate working tape, and a one-way output tape. Such a Turing Machine is called a *prefix machine*. Just as there are universal "ordinary" Turing Machines, there are also universal prefix machines that have identical computational power.

2.6 Kolmogorov Complexity

Now is when things begin to become tricky. There is a very special function K called *Kolmogorov Complexity*. Intuitively, the Kolmogorov complexity of a finite string x is the shortest computer program that prints x and then halts. More precisely, K is usually defined as a unary function that maps strings to integers and is implicitly based (or *conditioned*) on a concrete reference Turing machine represented by function Φ. The complete way of writing it is $K_\Phi(x)$. In practice, we

want to use a Turing Machine that is as general as possible. It is convenient to require the prefix property. Therefore we take Φ to be a universal prefix Turing Machine.[2] Because all universal Turing Machines can emulate one another reasonably efficiently, it does not matter much which one we take. We will say more about this later. For our purposes, we can suppose a universal prefix Turing machine is equivalent to any formal (implemented, real) computer programming language extended with a potentially unlimited memory. Recall that Φ represents a particular Turing machine with particular rules, and remember Φ is a partial function that is defined for all programs that terminate. If Φ is the transformation that maps a program x to its output o, then $K_\Phi(z)$ represents the length of the minimum program size (in bits) $|x|$ over all valid programs x such that $\Phi(x) = z$.

We can think of K as representing the smallest quantity of information required to recreate an object by any reliable procedure. For example, let x be the first 1000000 digits of π. Then $K(x)$ is small, because there is a short program generating x, as explained further below. On the other hand, for a random sequence of digits, $K(x)$ will usually be large because the program will probably have to hardcode a long list of abitrary values.

2.6.1 Conditional Kolmogorov Complexity

There is another form of K which is a bit harder to understand but still important to our discussions called *conditional Kolmogorov Complexity* and written

$$K(z|y).$$

The notation is confusing to some because the function takes two arguments. Its definition requires a slight enhancement of the earlier model of a Turing machine. While a Turing machine has a single infinite tape, Kolmogorov complexity is defined with respect to prefix Turing machines, which have an infinite working tape, an output tape and a restricted input tape that supports only one operation called "read next symbol". This input tape is often referred to as a *data tape* and is very similar to an input data file or stream read from standard input in Unix. Thus instead of imagining a program as a single string we must imagine a total runtime environment consisting of two parts: an input program tape with read/write memory, and a data tape of extremely limited functionality, unable to seek backward with the same limitations as POSIX standard input: there is getchar but no fseek. In the context of this slightly more complicated machine, we can define $K(z|y)$ as the size, in bits, of the smallest program that outputs z given a prefix-free encoding of y, say $\bar{y}$, as an initial input on the data tape. The idea is that if y gives a lot of information about z then $K(z|y) << K(z)$, but if z and y are completely unrelated, then $K(z \mid y) \approx K(z)$. For example, if $z = y$, then y provides a maximal amount of information about z. If we know that $z = y$ then a suitable program might be the following:

```
while true ; do
  c = getchar()
```

[2]There exists a version of Kolmogorov complexity that is based on standard rather than prefix Turing machines, but we shall not go into it here.

```
  if (c == EOF) ; then \index{halt}halt
  else putchar(c)
done
```

Here, already, we can see that $K(x|x) < 1000$ given the program above and a suitable universal prefix Turing machine. Note that the number of bits used to encode the whole thing is less than 1000. The more interesting case is when the two arguments are not equal, but related. Then the program must provide the missing information through more-complicated translation, preprogrammed results, or some other device.

2.6.2 Kolmogorov Randomness and Compressibility

As it turns out, K provides a convenient means for characterizing random sequences. Contrary to popular belief, random sequences are not simply sequences with no discernible patterns. Rather, there are a great many statistical regularities that can be proven and observed, but the difficulty lies in simply expressing them. As mentioned earlier, we can very easily express the idea of randomness by first defining different degrees of randomness as follows: a string x is $k-random$ if and only if $K(x) > |x| - k$. This simple formula expresses the idea that random strings are incompressible. The vast majority of strings are 1-random in this sense. This definition improves greatly on earlier definitions of randomness because it provides a concrete way to show a given, particular string is non-random by means of a simple computer program.

At this point, an example is appropriate. Imagine the following sequence of digits:

1, 4, 1, 5, 9, 2, 6, 5, 3, ...

and so on. Some readers may recognize the aforementioned sequence as the first digits of the Greek letter π with the first digit (3) omitted. If we extend these digits forward to a million places and continue to follow the precise decimal approximation of π, we would have a sequence that might appear random to most people. But it would be a matter of some confusing debate to try to settle a bet upon whether or not the sequence were truly random, even with all million of the digits written out in several pages. However, a clever observer, having noticed the digits corresponded to π, could simply write a short computer program (perhaps gotten off the internet) of no more than 10 kilobytes that could calculate the digits and print them out. What a surprise it would be then, to see such a short program reproduce such a long and seemingly meaningless sequence perfectly. This reproduction using a much shorter (less than one percent of the literal size) program is itself direct evidence that the sequence is non-random and in fact implies a certain regularity to the data with a high degree of likelihood. Simple counting arguments show that there can be no more than a vanishingly small number of highly compressible programs; in particular, the proportion of programs that are compressible by even k bits is no more than 2^{-k}. This can be understood by remembering that there are just two 1-bit strings (0 and 1), four 2-bit strings, and 2^m m-bit strings. So if we consider encodings of length m for source strings of length n with $n > m$, then at most 2^m different strings out of the total of 2^n source strings can be encoded in m bits. Thus, the ratio of strings compressible by $n-m$ bits is at most a 2^{m-n} proportion of all strings.

2.6.3 Universality In K

We have remarked earlier how universal Turing machines may emulate one another using finite simulation programs. In talking about the asymptotic behavior of K, these finite simulation programs do not matter more than an additive constant. For example, if we take x^n to mean the first n digits of π, then $K(x^n) = O(\log n)$ no matter which universal Turing machine is in use. This is because it will be possible to calculate any number of digits of π using a fixed-size program that reads as input the number of digits to output. The length of this input cannot be encoded in shorter than $\log n$ bits by a counting argument as in the previous section.

This implies that all variations of K are in some sense equivalent, because any two different variants of K given two different reference universal Turing machines will never differ by more than a fixed-size constant that depends only on the particular Turing machines chosen and not on the sequence. It is this universal character that winds up lending credence to the idea that K can be used as an *absolute* measure of the information contained in a given object. This is quite different from standard Shannon Information Theory based on the idea of *average information* required to communicate an object over a large number of trials and given some sort of *generating source* [103]. The great benefit of the Kolmogorov Complexity approach is that we need not explicitly define the generating source nor run the many trials to see desired results; just one look at the object is enough. Section 2.7 provides an example that will serve to illustrate the point.

2.6.4 Sophisticated Forms of K

There is now one more form of the K function that should be addressed, though it is perhaps the most complicated of all. It is written as follows:

$$K(x,y).$$

This represents the size in bits of the minimum program that outputs x followed by y, provided the output is given by first outputting x in a self-delimitting way (as explained earlier) and then outputting y. Formally, we define $K(x,y)$ as $K(\langle x,y\rangle)$, where $\langle\cdot,\cdot\rangle$ is defined as the pairing operation that takes two numbers and returns a pair: $\overline{x}y$.

2.7 Classical Probability Compared to K

Suppose we flip a fair coin. The type of sequence generated by the series of N flips of a fair coin is unpredictable in nature *by assumption* in normal probability theory. To define precisely what this means presents a bewildering array of possibilities. In the simplest, we might say the sequence is generated by a Bernoulli process where X takes on value 0 or 1 with probability

$$P(X=0)_{fair} = \frac{1}{2} = P(X=1)_{fair}.$$

The notation $P(\cdot)$ represents the chance that the event inside occurs. It is expressed as a ratio between 0 and 1 with 0 meaning never, 1 meaning always, and every number inbetween representing the proportion of times the event will be true given a large enough number of independent

trials. In such a setting, we may use a single bit to represent either possibility efficiently, and can always store N coin flips in just N bits regardless of the outcomes.

What if, instead of a fair coin, we use a biased one? For instance, if

$$P(X=0)_{biased} = \frac{1}{8},$$

and therefore since our simplified coins always turn up 0 or 1,

$$P(X=1)_{biased} = \frac{7}{8}.$$

Then we may use the scheme above to reliably transmit N flips in N bits. Alternatively, we may decide to encode the 1's more efficiently by using the following simple rule. Assume that N is even. Divide the N flips into pairs, and encode the pairs so that a pair of 1's takes just a single 1 bit to encode. If both are not 1, then instead output a 0 and then two more bits to represent the actual outcomes in order. Then continue with the next pair of two. One can quickly calculate that "$\frac{49}{64}$ of the time" the efficient 1-bit codeword will be output in this scheme which will save a great deal of space. Some of this savings will be lost in the cases where the 3-bit codeword is emitted, $\frac{15}{64}$ of the time. The average number of bits needed per outcome transmitted is then the codelength c:

$$c = \frac{49}{128} + \frac{15 \cdot 3}{64} = \frac{94}{128}.$$

This can also be improved somewhat down to the *Shannon entropy* $H(X)$ [79] of the source X with longer blocks or smarter encoding such as arithmetic codes [92] over an alphabet Σ:

$$H(X) = \sum_{i \in \Sigma} -P(X=i) \log P(X=i),$$

$$c = -\frac{1}{8} \cdot \log(\frac{1}{8}) - \frac{7}{8} \cdot \log(\frac{7}{8}).$$

By Shannon's famous coding theorem, this is essentially the smallest average code length that can be obtained under the assumption that the coin is independently tossed according to P_{biased}. Here though, there is already a problem, as we now cannot say, *unconditionally*, at least, that this many bits will be needed for any actual sequence of bits; luck introduces some variation in the actual space needed, though it is usually near the average. We know that such a coin is highly unlikely to repeatedly emit 0's, yet we cannot actually rule out this possibility. More to the point, in abstract terms the probability, while exponentially decaying with the greatest haste, still never quite reaches zero. It is useful to think carefully about this problem. All the laws of classical probability theory cannot make claims about a particular sequence but instead only about ensembles of experiments and expected proportions. Trying to pin down uncertainty in this crude way often serves only to make it appear elsewhere instead. In the Kolmogorov Complexity approach, we turn things upside-down: we say that a string is random if it is uncompressible. A string is $c-random$ if $K(x) > |x| - c$. This then directly addresses the question of how random a given string is by introducing different grades of randomness and invoking the universal function

K to automatically rule out the possibility of any short programs predicting a random string defined in this way. Returning to the fair coin example, the entropy is 1 bit per outcome. But we cannot say with certainty that a sequence coming from such a coin cannot be substantially compressed. This is only true with high probability.

2.8 Uncomputability of Kolmogorov Complexity

Some have made the claim that Kolmogorov Complexity is objective. In theory, it is. But in practice it is difficult to say; one major drawback of *K* is that it is uncomputable. Trying to compute it leads one to try immediately the shortest programs first, and as shown above it does not take many characters in a reasonable language to produce an infinite loop. This problem is impossible to protect against in general, and any multi-threaded approach is doomed to failure for this reason as it bumps up against the Halting Problem. [79]

A more fruitful approach has been to apply Kolmogorov Complexity by approximating it with data compressors. We may consider the problem of efficiently encoding a known biased random source into a minimum number of bits in such a way that the original sequence, no matter what it was, can once again be reconstructed, but so that also for certain sequences a shorter code is output. This is the basic idea of a data compression program. The most commonly used data compression programs of the last decade include *gzip*, *bzip2*, and *PPM*.

gzip is an old and reliable Lempel-Ziv type compressor with a 32-kilobyte window [122]. It is the simplest and fastest of the three compressors.

bzip2 is a wonderful new compressor using the blocksort algorithm [17]. It provides good compression and an expanded window of 900 kilobytes allowing for longer-range patterns to be detected. It is also reasonably fast.

PPM stands for Prediction by Partial Matching [4]. It is part of a new generation of powerful compressors using a pleasing mix of statistical models arranged by trees, suffix trees or suffix arrays. It usually achieves the best performance of any real compressor yet is also usually the slowest and most memory intensive.

Although restricted to the research community, a new challenger to PPM has arisen called context mixing compression. It is often the best compression scheme for a variety of file types but is very slow; further, it currently uses a neural network to do the mixing of contexts. See the *paq* series of compressors on the internet for more information on this exciting development in compression technology.

We use these data compressors to approximate from above the Kolmogorov Complexity function *K*. It is worth mentioning that all of the real compressors listed above operate on a bytewide basis, and thus all will return a multiple of 8 bits in their results. This is unfortunate for analyzing small strings, because the granularity is too coarse to allow for fine resolution of subtle details. To overcome this problem, the CompLearn system – the piece of software using which almost all experiments in later chapters have been carried out – supports the idea of a *virtual compressor* (originally suggested by Steven de Rooij): a virtual compressor is one that does not actually output an encoded compressed form, but instead simply accumulates the number of bits necessary to encode the results using a hypothetical arithmetic (or entropy) encoder. This frees

us from the bytewide restriction and indeed eliminates the need for rounding to a whole number of bits. Instead we may just return a real or floating-point value. This becomes quite useful when analyzing very similar strings of less than 100 bytes.

2.9 Summary

We have introduced the notion of universal computation and the K function indicating Kolmogorov Complexity. We have introduced Turing Machines and prefix codes as well as prefix machines. We have discussed a definition of a random string using K. We use these concepts in the next few chapters to explain in great detail our theory and experimental results.

Chapter 3

Normalized Compression Distance (NCD)

> You may very appropriately want to ask me how we are going to resolve the ever acceleratingly dangerous impasse of world-opposed politicians and ideological dogmas. I answer, it will be resolved by the computer. Man has ever-increasing confidence in the computer; witness his unconcerned landings as airtransport passengers coming in for a landing in the combined invisibility of fog and night. While no politician or political system can ever afford to yield understandably and enthusiastically to their adversaries and opposers, all politicians can and will yield enthusiastically to the computers safe flight-controlling capabilities in bringing all of humanity in for a happy landing. –Buckminster Fuller in *Operating Manual for Spaceship Earth*

In this chapter the Normalized Compression Distance (NCD) and the related Normalized Information Distance (NID) are presented and investigated. NCD is a similarity measure based on a data compressor. NID is simply the instantiation of NCD using the theoretical (and uncomputable) Kolmogorov compressor. Below we first review the definition of a metric. In Section 3.3, we explain precisely what is meant by universality in the case of NID. We discuss compressor axioms in Section 3.2, and properties of NCD in Section 3.4. At the end of the chapter, we connect NCD with a classical statistical quantity called *Kullback-Leibler divergence*. In Section 3.6.1 we connect arithmetic compressors to entropy, and in Section 3.6.2 we relate them to KL-divergence.

3.1 Similarity Metric

In mathematics, different distances arise in all sorts of contexts, and one usually requires these to be a "metric". We give a precise formal meaning to the loose distance notion of "degree of similarity" used in the pattern recognition literature.

Metric: Let Ω be a nonempty set and $\mathcal{R}^+$ be the set of nonnegative real numbers. A *metric* on Ω is a function $D : \Omega \times \Omega \to \mathcal{R}^+$ satisfying the metric (in)equalities:

- $D(x,y) = 0$ iff $x = y$,

- $D(x,y) = D(y,x)$ (symmetry), and
- $D(x,y) \leq D(x,z) + D(z,y)$ (triangle inequality).

The value $D(x,y)$ is called the *distance* between $x,y \in \Omega$. A familiar example of a metric is the Euclidean metric, the everyday distance $e(a,b)$ between two objects a,b expressed in, say, meters. Clearly, this distance satisfies the properties $e(a,a) = 0$, $e(a,b) = e(b,a)$, and $e(a,b) \leq e(a,c) + e(c,b)$ (for instance, $a =$ Amsterdam, $b =$ Brussels, and $c =$ Chicago.) We are interested in "similarity metrics". For example, if the objects are classical music pieces then the function $D(a,b) = 0$ if a and b are by the same composer and $D(a,b) = 1$ otherwise, is a similarity metric. This metric captures only one similarity aspect (feature) of music pieces, presumably an important one because it subsumes a conglomerate of more elementary features.

Density: In defining a class of admissible distances (not necessarily metric distances) we want to exclude unrealistic ones like $f(x,y) = \frac{1}{2}$ for *every* pair $x \neq y$. We do this by restricting the number of objects within a given distance of an object. As in [9] we do this by only considering effective distances, as follows.

3.1.1. DEFINITION. Let $\Omega = \Sigma^*$, with Σ a finite nonempty alphabet and Σ^* the set of finite strings over that alphabet. Since every finite alphabet can be recoded in binary, we choose $\Sigma = \{0,1\}$. In particular, "files" in computer memory are finite binary strings. A function $D : \Omega \times \Omega \to \mathcal{R}^+$ is an *admissible distance* if for every pair of objects $x,y \in \Omega$ the distance $D(x,y)$ satisfies the *density* condition (a version of the Kraft Inequality (2.4.1)):

$$\sum_y 2^{-D(x,y)} \leq 1, \tag{3.1.1}$$

is *computable*, and is *symmetric*, $D(x,y) = D(y,x)$.

If D is an admissible distance, then for every x the set $\{D(x,y) : y \in \{0,1\}^*\}$ is the length set of a prefix code, since it satisfies (2.4.1), the Kraft inequality. Conversely, if a distance is the length set of a prefix code, then it satisfies (2.4.1), see [31].

3.1.2. EXAMPLE. In representing the Hamming distance d between two strings of equal length n differing in positions $i_1, \ldots, i_d$, we can use a simple prefix-free encoding of $(n, d, i_1, \ldots, i_d)$ in $2\log n + 4\log\log n + 2 + d\log n$ bits. We encode n and d prefix-free in $\log n + 2\log\log n + 1$ bits each, see e.g. [79], and then the literal indexes of the actual flipped-bit positions. Adding an $O(1)$-bit program to interpret these data, with the strings concerned being x and y, we have defined $H_n(x,y) = 2\log n + 4\log\log n + d\log n + O(1)$ as the length of a prefix code word (prefix program) to compute x from y and *vice versa*. Then, by the Kraft inequality (Chapter 2, Section 2.4.1), $\sum_y 2^{-H_n(x,y)} \leq 1$. It is easy to verify that H_n is a metric in the sense that it satisfies the metric (in)equalities up to $O(\log n)$ additive precision.

Normalization: Large objects (in the sense of long strings) that differ by a tiny part are intuitively closer than tiny objects that differ by the same amount. For example, two whole mitochondrial genomes of 18,000 bases that differ by 9,000 are very different, while two whole nuclear genomes of 3×10^9 bases that differ by only 9,000 bases are very similar. Thus, absolute difference between two objects does not govern similarity, but relative difference seems to.

3.1.3. DEFINITION. A *compressor* is a lossless encoder mapping Ω into $\{0,1\}^*$ such that the resulting code is a prefix code. "Lossless" means that there is a decompressor that reconstructs the source message from the code message. For convenience of notation we identify "compressor" with a "code word length function" $C : \Omega \to \mathcal{N}$, where $\mathcal{N}$ is the set of nonnegative integers. That is, the compressed version of a file x has length $C(x)$. We only consider compressors such that $C(x) \leq |x| + O(\log |x|)$. (The additive logarithmic term is due to our requirement that the compressed file be a prefix code word.) We fix a compressor C, and call the fixed compressor the *reference compressor*.

3.1.4. DEFINITION. Let D be an admissible distance. Then we may make the definition $D^+(x) = \max\{D(x,z) : C(z) \leq C(x)\}$, and $D^+(x,y)$ is $D^+(x,y) = \max\{D^+(x), D^+(y)\}$. Note that since $D(x,y) = D(y,x)$, also $D^+(x,y) = D^+(y,x)$.

3.1.5. DEFINITION. Let D be an admissible distance. The *normalized admissible distance*, also called a *similarity distance*, $d(x,y)$, based on D relative to a reference compressor C, is defined by

$$d(x,y) = \frac{D(x,y)}{D^+(x,y)}.$$

It follows from the definitions that a normalized admissible distance is a function $d : \Omega \times \Omega \to [0,1]$ that is symmetric: $d(x,y) = d(y,x)$.

3.1.6. LEMMA. *For every $x \in \Omega$, and constant $e \in [0,1]$, a normalized admissible distance satisfies the density constraint*

$$|\{y : d(x,y) \leq e,\ C(y) \leq C(x)\}| < 2^{eD^+(x)+1}. \tag{3.1.2}$$

PROOF. Assume to the contrary that d does not satisfy (3.1.2). Then, there is an $e \in [0,1]$ and an $x \in \Omega$, such that (3.1.2) is false. We first note that, since $D(x,y)$ is an admissible distance that satisfies (3.1.1), $d(x,y)$ satisfies a "normalized" version of the Kraft inequality:

$$\sum_{y:C(y)\leq C(x)} 2^{-d(x,y)D^+(x)} \leq \sum_y 2^{-d(x,y)D^+(x,y)} \leq 1. \tag{3.1.3}$$

Starting from (3.1.3) we obtain the required contradiction:

$$\begin{aligned} 1 &\geq \sum_{y:C(y)\leq C(x)} 2^{-d(x,y)D^+(x)} \\ &\geq \sum_{y:d(x,y)\leq e,\ C(y)\leq C(x)} 2^{-eD^+(x)} \\ &\geq 2^{eD^+(x)+1} 2^{-eD^+(x)} > 1. \end{aligned}$$

□

If $d(x,y)$ is the normalized version of an admissible distance $D(x,y)$ then (3.1.3) is equivalent to (3.1.1). We call a normalized distance a "similarity" distance, because it gives a relative similarity according to the distance (with distance 0 when objects are maximally similar and distance 1 when they are maximally dissimilar) and, conversely, for every well-defined computable notion of similarity we can express it as a metric distance according to our definition. In the literature a distance that expresses lack of similarity (like ours) is often called a "dissimilarity" distance or a "disparity" distance.

3.1.7. REMARK. As far as this author knows, the idea of normalized metric is, surprisingly, not well-studied. An exception is [121], which investigates normalized metrics to account for relative distances rather than absolute ones, and it does so for much the same reasons as in the present work. An example there is the normalized Euclidean metric $|x-y|/(|x|+|y|)$, where $x,y \in \mathcal{R}^n$ ($\mathcal{R}$ denotes the real numbers) and $|\cdot|$ is the Euclidean metric—the L_2 norm. Another example is a normalized symmetric-set-difference metric. But these normalized metrics are not necessarily effective in that the distance between two objects gives the length of an effective description to go from either object to the other one.

3.1.8. REMARK. Our definition of normalized admissible distance is more direct than in [77], and the density constraints (3.1.2) and (3.1.3) follow from the definition. In [77] we put a stricter density condition in the definition of "admissible" normalized distance, which is, however, harder to satisfy and maybe too strict to be realistic. The purpose of this stricter density condition was to obtain a stronger "universality" property than the present Theorem 3.5.3, namely one with $\alpha = 1$ and $\varepsilon = O(1/\max\{C(x),C(y)\})$. Nonetheless, both definitions coincide if we set the length of the compressed version $C(x)$ of x to the ultimate compressed length $K(x)$, the Kolmogorov complexity of x.

3.1.9. EXAMPLE. To obtain a normalized version of the Hamming distance of Example 3.1.2, we define $h_n(x,y) = H_n(x,y)/H_n^+(x,y)$. We can set $H_n^+(x,y) = H_n^+(x) = (n+2)\lceil \log n \rceil + 4\lceil \log\log n \rceil + O(1)$ since every contemplated compressor C will satisfy $C(x) = C(\overline{x})$, where $\overline{x}$ is x with all bits flipped (so $H_n^+(x,y) \geq H_n^+(z,\overline{z})$ for either $z = x$ or $z = y$). By (3.1.2), for every x, the number of y with $C(y) \leq C(x)$ in the Hamming ball $h_n(x,y) \leq e$ is less than $2^{eH_n^+(x)+1}$. This upper bound is an obvious overestimate for $e \geq 1/\log n$. For lower values of e, the upper bound is correct by the observation that the number of y's equals $\sum_{i=0}^{en} \binom{n}{en} \leq 2^{nH(e)}$, where $H(e) = e\log e + (1-e)\log(1-e)$, Shannon's entropy function. Then, $eH_n^+(x) > en\log n > enH(e)$ since $e\log n > H(e)$.

3.2 Normal Compressor

We give axioms determining a large family of compressors that both include most (if not all) real-world compressors and ensure the desired properties of the NCD to be defined later.

3.2.1. DEFINITION. 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 of Ω involved in the (in)equality concerned, the following:

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*: $C(xy) + C(z) \leq C(xz) + C(yz)$.

Idempotency: A reasonable compressor will see exact repetitions and obey idempotency up to the required precision. It will also compress the empty string to the empty string.

Monotonicity: A real compressor must have the monotonicity property, at least up to the required precision. The property is evident for stream-based compressors, and only slightly less evident for block-coding compressors.

Symmetry: Stream-based compressors of the Lempel-Ziv family, like gzip and pkzip, and the predictive PPM family, like PPMZ, are possibly not precisely symmetric. This is related to the stream-based property: the initial file x may have regularities to which the compressor adapts; after crossing the border to y it must unlearn those regularities and adapt to the ones of y. This process may cause some imprecision in symmetry that vanishes asymptotically with the length of x, y. A compressor must be poor indeed (and will certainly not be used to any extent) if it doesn't satisfy symmetry up to the required precision. Apart from stream-based, the other major family of compressors is block-coding based, like bzip2. They essentially analyze the full input block by considering all rotations in obtaining the compressed version. It is to a great extent symmetrical, and real experiments show no departure from symmetry.

Distributivity: The distributivity property is not immediately intuitive. In Kolmogorov complexity theory the stronger distributivity property

$$C(xyz) + C(z) \leq C(xz) + C(yz) \tag{3.2.1}$$

holds (with $K = C$). However, to prove the desired properties of NCD below, only the weaker distributivity property

$$C(xy) + C(z) \leq C(xz) + C(yz) \tag{3.2.2}$$

above is required, also for the boundary case were $C = K$. In practice, real-world compressors appear to satisfy this weaker distributivity property up to the required precision.

3.2.2. DEFINITION. Define

$$C(y|x) = C(xy) - C(x). \tag{3.2.3}$$

This number $C(y|x)$ of bits of information in y, relative to x, can be viewed as the excess number of bits in the compressed version of xy compared to the compressed version of x, and is called the amount of *conditional compressed information*.

In the definition of compressor the decompression algorithm is not included (unlike the case of Kolmorogov complexity, where the decompressing algorithm is given by definition), but it is easy to construct one: Given the compressed version of x in $C(x)$ bits, we can run the compressor on all candidate strings z—for example, in length-increasing lexicographical order, until we find the

compressed string $z_0 = x$. Since this string decompresses to x we have found $x = z_0$. Given the compressed version of xy in $C(xy)$ bits, we repeat this process using strings xz until we find the string xz_1 of which the compressed version equals the compressed version of xy. Since the former compressed version decompresses to xy, we have found $y = z_1$. By the unique decompression property we find that $C(y|x)$ is the extra number of bits we require to describe y apart from describing x. It is intuitively acceptable that the conditional compressed information $C(x|y)$ satisfies the triangle inequality

$$C(x|y) \leq C(x|z) + C(z|y). \tag{3.2.4}$$

3.2.3. LEMMA. *Both* (3.2.1) *and* (3.2.4) *imply* (3.2.2).

PROOF. ((3.2.1) implies (3.2.2):) By monotonicity.

((3.2.4) implies (3.2.2):) Rewrite the terms in (3.2.4) according to (3.2.3), cancel $C(y)$ in the left- and right-hand sides, use symmetry, and rearrange. □

3.2.4. LEMMA. *A normal compressor satisfies additionally* subadditivity*:* $C(xy) \leq C(x) + C(y)$.

PROOF. Consider the special case of distributivity with z the empty word so that $xz = x$, $yz = y$, and $C(z) = 0$. □

Subadditivity: The subadditivity property is clearly also required for every viable compressor, since a compressor may use information acquired from x to compress y. Minor imprecision may arise from the unlearning effect of crossing the border between x and y, mentioned in relation to symmetry, but again this must vanish asymptotically with increasing length of x, y.

3.3 Background in Kolmogorov complexity

Technically, the *Kolmogorov complexity* of x given y is the length of the shortest binary program, for the reference universal prefix Turing machine, that on input y outputs x; it is denoted as $K(x|y)$. For precise definitions, theory and applications, see [79]. The Kolmogorov complexity of x is the length of the shortest binary program with no input that outputs x; it is denoted as $K(x) = K(x|\lambda)$ where λ denotes the empty input. Essentially, the Kolmogorov complexity of a file is the length of the ultimate compressed version of the file. In [9] the *information distance* $E(x,y)$ was introduced, defined as the length of the shortest binary program for the reference universal prefix Turing machine that, with input x computes y, and with input y computes x. It was shown there that, up to an additive logarithmic term, $E(x,y) = \max\{K(x|y), K(y|x)\}$. It was shown also that $E(x,y)$ is a metric, up to negligible violations of the metric inequalties. Moreover, it is universal in the sense that for every admissible distance $D(x,y)$ as in Definition 3.1.1, $E(x,y) \leq D(x,y)$ up to an additive constant depending on D but not on x and y. In [77], the normalized version of $E(x,y)$, called the *normalized information distance*, is defined as

$$\mathrm{NID}(x,y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}. \tag{3.3.1}$$

It too is a metric, and it is universal in the sense that this single metric minorizes up to an negligible additive error term all normalized admissible distances in the class considered in [77]. Thus, if two files (of whatever type) are similar (that is, close) according to the particular feature described by a particular normalized admissible distance (not necessarily metric), then they are also similar (that is, close) in the sense of the normalized information metric. This justifies calling the latter *the* similarity metric. We stress once more that different pairs of objects may have different dominating features. Yet every such dominant similarity is detected by the NID . However, this metric is based on the notion of Kolmogorov complexity. Unfortunately, the Kolmogorov complexity is non-computable in the Turing sense. Approximation of the denominator of (3.3.1) by a given compressor C is straightforward: it is $\max\{C(x),C(y)\}$. The numerator is more tricky. It can be rewritten as

$$\max\{K(x,y)-K(x),K(x,y)-K(y)\}, \tag{3.3.2}$$

within logarithmic additive precision, by the additive property of Kolmogorov complexity [79]. The term $K(x,y)$ represents the length of the shortest program for the pair (x,y). In compression practice it is easier to deal with the concatenation xy or yx. Again, within logarithmic precision $K(x,y) = K(xy) = K(yx)$. Following a suggestion by Steven de Rooij, one can approximate (3.3.2) best by $\min\{C(xy),C(yx)\} - \min\{C(x),C(y)\}$. Here, and in the later experiments using the CompLearn Toolkit, we simply use $C(xy)$ rather than $\min\{C(xy),C(yx)\}$. This is justified by the observation that block-coding based compressors are symmetric almost by definition, and experiments with various stream-based compressors (gzip, PPMZ) show only small deviations from symmetry.

The result of approximating the NID using a real compressor C is called the normalized compression distance (NCD), formally defined in (3.5.1). The theory as developed for the Kolmogorov-complexity based NID in [77], may not hold for the (possibly poorly) approximating NCD . It is nonetheless the case that experiments show that the NCD apparently has (some) properties that make the NID so appealing. To fill this gap between theory and practice, we develop the theory of NCD from first principles, based on the axiomatics of Section 3.2. We show that the NCD is a quasi-universal similarity metric relative to a normal reference compressor C. The theory developed in [77] is the boundary case $C = K$, where the "quasi-universality" below has become full "universality".

3.4 Compression Distance

We define a compression distance based on a normal compressor and show it is an admissible distance. In applying the approach, we have to make do with an approximation based on a far less powerful real-world reference compressor C. A compressor C approximates the information distance $E(x,y)$, based on Kolmogorov complexity, by the compression distance $E_C(x,y)$ defined as

$$E_C(x,y) = C(xy) - \min\{C(x),C(y)\}. \tag{3.4.1}$$

Here, $C(xy)$ denotes the compressed size of the concatenation of x and y, $C(x)$ denotes the compressed size of x, and $C(y)$ denotes the compressed size of y.

3.4.1. LEMMA. *If C is a normal compressor, then $E_C(x,y)+O(1)$ is an admissible distance.*

PROOF. **Case 1:** Assume $C(x) \leq C(y)$. Then $E_C(x,y) = C(xy) - C(x)$. Then, given x and a prefix-program of length $E_C(x,y)$ consisting of the suffix of the C-compressed version of xy, and the compressor C in $O(1)$ bits, we can run the compressor C on all xz's, the candidate strings z in length-increasing lexicographical order. When we find a z so that the suffix of the compressed version of xz matches the given suffix, then $z = y$ by the unique decompression property.

Case 2: Assume $C(y) \geq C(x)$. By symmetry $C(xy) = C(yx)$. Now follow the proof of Case 1. □

3.4.2. LEMMA. *If C is a normal compressor, then $E_C(x,y)$ satisfies the metric (in)equalities up to logarithmic additive precision.*

PROOF. Only the triangular inequality is non-obvious. By (3.2.2) $C(xy)+C(z) \leq C(xz)+C(yz)$ up to logarithmic additive precision. There are six possibilities, and we verify the correctness of the triangular inequality in turn for each of them. Assume $C(x) \leq C(y) \leq C(z)$: Then $C(xy) - C(x) \leq C(xz) - C(x) + C(yz) - C(y)$. Assume $C(y) \leq C(x) \leq C(z)$: Then $C(xy) - C(y) \leq C(xz) - C(y) + C(yz) - C(x)$. Assume $C(x) \leq C(z) \leq C(y)$: Then $C(xy) - C(x) \leq C(xz) - C(x) + C(yz) - C(z)$. Assume $C(y) \leq C(z) \leq C(x)$: Then $C(xy) - C(y) \leq C(xz) - C(z) + C(yz) - C(y)$. Assume $C(z) \leq C(x) \leq C(y)$: Then $C(xy) - C(x) \leq C(xz) - C(z) + C(yz) - C(z)$. Assume $C(z) \leq C(y) \leq C(x)$: Then $C(xy) - C(y) \leq C(xz) - C(z) + C(yz) - C(z)$. □

3.4.3. LEMMA. *If C is a normal compressor, then $E_C^+(x,y) = \max\{C(x), C(y)\}$.*

PROOF. Consider a pair (x,y). The $\max\{C(xz) - C(z) : C(z) \leq C(y)\}$ is $C(x)$ which is achieved for $z = \lambda$, the empty word, with $C(\lambda) = 0$. Similarly, the $\max\{C(yz) - C(z) : C(z) \leq C(x)\}$ is $C(y)$. Hence the lemma. □

3.5 Normalized Compression Distance

The normalized version of the admissible distance $E_C(x,y)$, the compressor C based approximation of the normalized information distance (3.3.1), is called the *normalized compression distance* or NCD:

$$\mathrm{NCD}(x,y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}. \tag{3.5.1}$$

This NCD is the main concept of this work. It is the real-world version of the ideal notion of normalized information distance NID in (3.3.1).

3.5.1. REMARK. In practice, the NCD is a non-negative number $0 \leq r \leq 1+\varepsilon$ representing how different the two files are. Smaller numbers represent more similar files. The ε in the upper bound is due to imperfections in our compression techniques, but for most standard compression algorithms one is unlikely to see an ε above 0.1 (in our experiments gzip and bzip2 achieved NCD 's above 1, but PPMZ always had NCD at most 1).

There is a natural interpretation to $\mathrm{NCD}(x,y)$: If, say, $C(y) \geq C(x)$ then we can rewrite

$$\mathrm{NCD}(x,y) = \frac{C(xy) - C(x)}{C(y)}.$$

That is, the distance $\mathrm{NCD}(x,y)$ between x and y is the improvement due to compressing y using x as previously compressed "data base," and compressing y from scratch, expressed as the ratio between the bit-wise length of the two compressed versions. Relative to the reference compressor we can define the information in x about y as $C(y) - C(y|x)$. Then, using (3.2.3),

$$\mathrm{NCD}(x,y) = 1 - \frac{C(y) - C(y|x)}{C(y)}.$$

That is, the NCD between x and y is 1 minus the ratio of the information x about y and the information in y.

3.5.2. THEOREM. *If the compressor is normal, then the* NCD *is a normalized admissible distance satsifying the metric (in)equalities, that is, a similarity metric.*

PROOF. If the compressor is normal, then by Lemma 3.4.1 and Lemma 3.4.3, the NCD is a normalized admissible distance. It remains to show it satisfies the three metric (in)equalities.

1. By idempotency we have $\mathrm{NCD}(x,x) = 0$. By monotonicity we have $\mathrm{NCD}(x,y) \geq 0$ for every x,y, with inequality for $y \neq x$.

2. $\mathrm{NCD}(x,y) = \mathrm{NCD}(y,x)$. The NCD is unchanged by interchanging x and y in (3.5.1).

3. The difficult property is the triangle inequality. Without loss of generality we assume $C(x) \leq C(y) \leq C(z)$. Since the NCD is symmetrical, there are only three triangle inequalities that can be expressed by $\mathrm{NCD}(x,y), \mathrm{NCD}(x,z), \mathrm{NCD}(y,z)$. We verify them in turn:

   (a) $\mathrm{NCD}(x,y) \leq \mathrm{NCD}(x,z) + \mathrm{NCD}(z,y)$: By distributivity, the compressor itself satisfies $C(xy) + C(z) \leq C(xz) + C(zy)$. Subtracting $C(x)$ from both sides and rewriting, $C(xy) - C(x) \leq C(xz) - C(x) + C(zy) - C(z)$. Dividing by $C(y)$ on both sides we find

   $$\frac{C(xy) - C(x)}{C(y)} \leq \frac{C(xz) - C(x) + C(zy) - C(z)}{C(y)}.$$

   The left-hand side is ≤ 1.

      i. Assume the right-hand side is ≤ 1. Setting $C(z) = C(y) + \Delta$, and adding Δ to both the numerator and denominator of the right-hand side, it can only increase and draw closer to 1. Therefore,

      $$\begin{aligned} &\frac{C(xy) - C(x)}{C(y)} \\ &\leq \frac{C(xz) - C(x) + C(zy) - C(z) + \Delta}{C(y) + \Delta} \\ &= \frac{C(zx) - C(x)}{C(z)} + \frac{C(zy) - C(y)}{C(z)}, \end{aligned}$$

      which was what we had to prove.

ii. Assume the right-hand side is > 1. We proceed like in the previous case, and add Δ to both numerator and denominator. Although now the right-hand side decreases, it must still be greater than 1, and therefore the right-hand side remains at least as large as the left-hand side.

(b) $\mathrm{NCD}(x,z) \leq \mathrm{NCD}(x,y) + \mathrm{NCD}(y,z)$: By distributivity we have $C(xz) + C(y) \leq C(xy) + C(yz)$. Subtracting $C(x)$ from both sides, rearranging, and dividing both sides by $C(z)$ we obtain

$$\frac{C(xz) - C(x)}{C(z)} \leq \frac{C(xy) - C(x)}{C(z)} + \frac{C(yz) - C(y)}{C(z)}.$$

The right-hand side doesn't decrease when we substitute $C(y)$ for the denominator $C(z)$ of the first term, since $C(y) \leq C(z)$. Therefore, the inequality stays valid under this substitution, which was what we had to prove.

(c) $\mathrm{NCD}(y,z) \leq \mathrm{NCD}(y,x) + \mathrm{NCD}(x,z)$: By distributivity we have $C(yz) + C(x) \leq C(yx) + C(xz)$. Subtracting $C(y)$ from both sides, using symmetry, rearranging, and dividing both sides by $C(z)$ we obtain

$$\frac{C(yz) - C(y)}{C(z)} \leq \frac{C(xy) - C(x)}{C(z)} + \frac{C(yz) - C(y)}{C(z)}.$$

The right-hand side doesn't decrease when we substitute $C(y)$ for the denominator $C(z)$ of the first term, since $C(y) \leq C(z)$. Therefore, the inequality stays valid under this substitution, which was what we had to prove.

□

Quasi-Universality: We now digress to the theory developed in [77], which formed the motivation for developing the NCD . If, instead of the result of some real compressor, we substitute the Kolmogorov complexity for the lengths of the compressed files in the NCD formula, the result is the NID as in (3.3.1). It is universal in the following sense: Every admissible distance expressing similarity according to some feature, that can be computed from the objects concerned, is comprised (in the sense of minorized) by the NID . Note that every feature of the data gives rise to a similarity, and, conversely, every similarity can be thought of as expressing some feature: being similar in that sense. Our actual practice in using the NCD falls short of this ideal theory in at least three respects:

(i) The claimed universality of the NID holds only for indefinitely long sequences x, y. Once we consider strings x, y of definite length n, it is only universal with respect to "simple" computable normalized admissible distances, where "simple" means that they are computable by programs of length, say, logarithmic in n. This reflects the fact that, technically speaking, the universality is achieved by summing the weighted contribution of all similarity distances in the class considered with respect to the objects considered. Only similarity distances of which the complexity is small (which means that the weight is large), with respect to the size of the data concerned, kick in.

(ii) The Kolmogorov complexity is not computable, and it is in principle impossible to compute how far off the NCD is from the NID . So we cannot in general know how well we are doing using the NCD .

(iii) To approximate the NCD we use standard compression programs like gzip, PPMZ, and bzip2. While better compression of a string will always approximate the Kolmogorov complexity better, this may not be true for the NCD . Due to its arithmetic form, subtraction and division, it is theoretically possible that while all items in the formula get better compressed, the improvement is not the same for all items, and the NCD value moves away from the NID value. In our experiments we have not observed this behavior in a noticable fashion. Formally, we can state the following:

3.5.3. THEOREM. *Let d be a computable normalized admissible distance and C be a normal compressor. Then,* $\mathrm{NCD}(x,y) \leq \alpha d(x,y) + \varepsilon$, *where for $C(x) \geq C(y)$, we have $\alpha = D^+(x)/C(x)$ and $\varepsilon = (C(x|y) - K(x|y))/C(x)$, with $C(x|y)$ according to* (3.2.3).

PROOF. Fix d, C, x, y in the statement of the theorem. Since the NCD is symmetrical, we can, without loss of generality, let $C(x) \geq C(y)$. By (3.2.3) and the symmetry property $C(xy) = C(yx)$ we have $C(x|y) \geq C(y|x)$. Therefore, $\mathrm{NCD}(x,y) = C(x|y)/C(x)$. Let $d(x,y)$ be the normalized version of the admissible distance $D(x,y)$; that is, $d(x,y) = D(x,y)/D^+(x,y)$. Let $d(x,y) = e$. By (3.1.2), there are $< 2^{eD^+(x)+1}$ many (x,v) pairs, such that $d(x,v) \leq e$ and $C(y) \leq C(x)$. Since d is computable, we can compute and enumerate all these pairs. The initially fixed pair (x,y) is an element in the list and its index takes $\leq eD^+(x) + 1$ bits. Therefore, given x, the y can be described by at most $eD^+(x) + O(1)$ bits—its index in the list and an $O(1)$ term accounting for the lengths of the programs involved in reconstructing y given its index in the list, and algorithms to compute functions d and C. Since the Kolmogorov complexity gives the length of the shortest effective description, we have $K(y|x) \leq eD^+(x) + O(1)$. Substitution, rewriting, and using $K(x|y) \leq E(x,y) \leq D(x,y)$ up to ignorable additive terms (Section 3.3), yields $\mathrm{NCD}(x,y) = C(x|y)/C(x) \leq \alpha e + \varepsilon$, which was what we had to prove.

□

3.5.4. REMARK. Clustering according to NCD will group sequences together that are similar according to features that are not explicitly known to us. Analysis of what the compressor actually does, still may not tell us which features that make sense to us can be expressed by conglomerates of features analyzed by the compressor. This can be exploited to track down unknown features implicitly in classification: forming automatically clusters of data and see in which cluster (if any) a new candidate is placed.

Another aspect that can be exploited is exploratory: Given that the NCD is small for a pair x, y of specific sequences, what does this really say about the sense in which these two sequences are similar? The above analysis suggests that close similarity will be due to a dominating feature (that perhaps expresses a conglomerate of subfeatures). Looking into these deeper causes may give feedback about the appropriateness of the realized NCD distances and may help extract more intrinsic information about the objects, than the oblivious division into clusters, by looking for the common features in the data clusters.

3.6 Kullback-Leibler divergence and NCD

NCD is sometimes considered a mysterious and obscure measure of information distance. In fact, as we explain in this section, in some cases it can be thought of as a generalization and extension of older and well-established methods. The Normalized Information Distance is a purely theoretical concept and cannot be exactly computed for even the simplest files due to the inherent incomputability of Kolmogorov Complexity. The Normalized Compression Distance, however, replaces the uncomputable K with an approximation based on a particular data compressor. Different data compression algorithms lead to different varieties of NCD. Modern data compression programs use highly evolved and complicated schemes that involve stochastic and adaptive modelling of the data at many levels simultaneously. These are all but impossible to analyze from a precise mathematical viewpoint, and thus many consider modern data compression as much an art as a science. If, however, we look instead at the compressors popular in UNIX in the 1970's, we can begin to understand how NCD achieves its results. As we show in this section, it turns out that with such simple compressors, the NCD calculates the *total KL-divergence to the mean*. Below we first (Section 3.6.1) connect such compressors to entropy, and then (Section 3.6.2) relate them to KL-divergence.

3.6.1 Static Encoders and Entropy

The UNIX System V *pack* command uses a static (non-adaptive) Huffman coding scheme to compress files. The method works in two passes. First, the input file is considered as a sequence of 8-bit bytes, and a histogram is constructed to represent the frequency of each byte. Next, an optimal Huffman code is constructed according to this histogram, and is represented in memory as a Huffman tree. This Huffman tree is written out as a variable-length header in the compressed file. Finally, the algorithm makes a second pass through the file and encodes it using the Huffman code it has constructed. Notice that the coding scheme does not change throughout the duration of the file. It is this failure to adapt that makes this compressor amenable to mathematical analysis. In the following example, we analyze a hypothetical static arithmetic coder which yields simpler codelength equations. The simpler Huffman pack encoder will perform similarly but must round upwards the codelengths of each symbol to a whole number of bits and thus can lose at most 1 bit per symbol as compared to the arithmetic coder described below.

Consider therefore the particular case of a simple static arithmetic coder S. Let $S(D)$ represent the function mapping a file, D, to the number of bits needed to encode D with S. A static arithmetic encoder really models its input file as an i.i.d. (independently, identically distributed) Bernoulli process. For distributions of this type, the code length that is achieved very closely approximates the empirical Shannon entropy of the file [92, 48] multiplied by the file size N_D. Thus, if data are indeed distributed according to a Bernoulli process, then this encoder almost achieves the theoretically ideal Shannon limit. Let us explain this in more detail. Let D be a file over an alphabet Σ. Let $n(D,i)$ denote the number of occurrences of symbol i in file D, and let N_D denote the total number of symbols in file D. Then

$$N_D = \sum_{i \in \Sigma} n(D,i). \tag{3.6.1}$$

The *empirical distribution* corresponding to file D is defined as the probability distribution that assigns a probability to each symbol in the alphabet given by

$$P_D(i) = \frac{n(D,i)}{N_D}. \tag{3.6.2}$$

The empirical distribution P_D is just the histogram of relative frequencies of the symbols of Σ occurring in D.

It turns out that, when provided the empirical distribution P_D, the theoretical arithmetic coder S requires just this many bits:

$$Z(D) = N_D H(P_D), \tag{3.6.3}$$

with H representing *Shannon entropy*:

$$H(P_D) = \sum_{i\in\Sigma} -P_D(i)\log P_D(i). \tag{3.6.4}$$

For a real static arithmetic coding implementation S, there is a need to transmit a small fixed sized header as a preamble to the arithmetic coding. This header provides an encoding of the histogram P_D corresponding to the file D to be encoded. This quantity is termed $\|hdr\|$. So:

$$S(D) = N_D H(P_D) + \|hdr\|. \tag{3.6.5}$$

To be fully precise, in a real implementation, the number of bits needed is always an integer, so (3.6.5) really needs to be rounded up; but the effects of this change is negligible, so we will ignore it.

3.6.1. REMARK. Let us give a little bit more explanation of (3.6.3). From the Kraft inequality (Section 2.4.1), we know that for any distribution P on strings $D \in \Sigma^n$ of length N, there exists a compressor Z such that for all $D \in \Sigma^N$, $Z(D) = -\log P(D)$, where again we ignore rounding issues. Now let us model D according to a Bernoulli process, where each element of D is distributed independently according to the empirical distribution P_D. Under this distribution, setting $D = x_1 \ldots x_n$,

$$\begin{aligned}
Z(D) &= -\log P(D) = -\log \prod_{j=1}^{N} P_D(x_j) \\
&= -\log \prod_{i\in\Sigma} P_D(i)^{n(D,i)} = -N \sum_{i\in\Sigma} \frac{n(D,i)}{N} \log P_D(i) && (3.6.6)\\
&= -N \sum_{i\in\Sigma} P_D(i)\log P_D(i) = -N E_{P_D}[-\log P_D(X)] && (3.6.7)\\
&= N H(P_D). && (3.6.8)
\end{aligned}$$

Such a compressor Z makes use of the empirical distribution P_D, so the encoding of D with length $Z(D)$ can only be decoded by a decoder who already knows P_D. Thus, to turn Z into a compressor S that can be used on all sequences (and not only those with a given, fixed P_D), it suffices to first encode P_D using some previously agreed-upon code, which takes $\|hdr\|$ bits, and then encode D using $Z(D)$ bits. By (3.6.8) this is equal to (3.6.5).

3.6.2 NCD and KL-divergence

We now connect the NCD based on a static arithmetic encoder with the KL-divergence [31]. Consider two files F and G with empirical distributions P_F and P_G:

$$P_F(i) = \frac{n(F,i)}{N_F} \;\; ; \;\; P_G(i) = \frac{n(G,i)}{N_G}. \tag{3.6.9}$$

There is a file B that is the concatenation of F followed by G and has empirical distribution

$$P_B(i) = \frac{n(F,i)+n(G,i)}{N_F+N_G}. \tag{3.6.10}$$

The size for S run on B is just the size of the histogram $\|hdr\|$ and the entropy of P_B times the number of symbols:

$$\begin{aligned} S(B) &= \|hdr\| + (N_F+N_G)H(P_B) \\ &= \|hdr\| + (N_F+N_G)\sum_{i\in\Sigma} -P_B(i)\log P_B(i) \\ &= \|hdr\| - (N_F+N_G)\sum_{i\in\Sigma} \frac{n(F,i)+n(G,i)}{N_F+N_G}\log\frac{n(F,i)+n(G,i)}{N_F+N_G} \\ &= \|hdr\| - \sum_{i\in\Sigma}(n(F,i)+n(G,i))\log\frac{n(F,i)+n(G,i)}{N_F+N_G}. \end{aligned} \tag{3.6.11}$$

Recall that Kullback-Leibler divergence [31] is defined upon two distributions P,Q as

$$KL(P \parallel Q) = \sum_{i\in\Sigma} P(i)\log\frac{P(i)}{Q(i)}, \tag{3.6.12}$$

so that

$$S(B) = \|hdr\| + N_F H(P_F) + N_G H(P_G) + N_F KL(P_F \parallel P_B) + N_G KL(P_G \parallel P_B). \tag{3.6.13}$$

At this point we determine a formula for NCD_S. Recall that the NCD is defined in (3.5.1) as a function to determine an information distance between two input files:

$$NCD_C(x,y) = \frac{C(xy) - \min\{C(x),C(y)\}}{\max\{C(x),C(y)\}}.$$

Here, $C(xy)$ denotes the compressed size of the concatenation of x and y, $C(x)$ denotes the compressed size of x, and $C(y)$ denotes the compressed size of y. C is a function returning a length, usually measured in bits, and realized by a particular data compression program. Different algorithms will yield different behaviors for this quotient. For our simple compressor S, we get

$$NCD_S(F,G) = \frac{S(B) - \min\{S(F),S(G)\}}{\max\{S(F),S(G)\}}. \tag{3.6.14}$$

Assume without loss of generality that $S(F) \leq S(G)$, then $NCD_S(F,G) =$

$$\begin{aligned} &= \frac{\|hdr\| + N_F H(P_F) + N_G H(P_G) + N_F KL(P_F \parallel P_B) + N_G KL(P_G \parallel P_B) - \|hdr\| - N_F H(P_F)}{\|hdr\| + N_G H(P_G)} \\ &= \frac{N_G H(P_G) + N_F KL(P_F \parallel P_B) + N_G KL(P_G \parallel P_B)}{\|hdr\| + N_G H(P_G)}. \end{aligned} \tag{3.6.15}$$

In the limit, as $N_{F,G} \to \infty$,

$$\begin{aligned} \lim_{N_F, N_G \to \infty} NCD_S(F,G) &= 1 + \frac{N_F KL(P_F \parallel P_B) + N_G KL(P_G \parallel P_B)}{N_G H(P_G)} \\ &= 1 + \frac{\frac{N_F}{N_G} KL(P_F \parallel P_B) + KL(P_G \parallel P_B)}{H(P_G)}. \end{aligned} \tag{3.6.16}$$

Notice that $0 \leq \frac{N_F}{N_G} < 1$. It is apparent that P_B represents an aggregate distribution formed by combining P_F and P_G. When $N_F = N_G$,

$$\lim_{N_F, N_G \to \infty} NCD_S(F,G) = 1 + \frac{KL(P_F \parallel P_B) + KL(P_G \parallel P_B)}{\max\{H(P_F), H(P_G)\}}, \tag{3.6.17}$$

or using $A(P_F, P_G)$ to represent *information radius* [52] or *total KL-divergence to the mean* [32], then

$$\lim_{N_F, N_G \to \infty} NCD_S(F,G) = 1 + \frac{A(P_F, P_G)}{\max\{H(P_F), H(P_G)\}}. \tag{3.6.18}$$

We may interpret NCD_S to behave as a ratio of information radius to maximum individual entropy. The static arithmetic coder S severely violates the *subadditivity* assumption of a *normal compressor* (Section 3.2) and causes a positive offset bias of $+1$. In general, NCD_S behaves locally quadratically when at least one of the two files involved has high entropy. This fact can be demonstrated by use of Taylor series to approximate the logarithms in NCD_S about P_B (we omit the details). When both $H(P_F)$ and $H(P_G)$ are small, NCD_S can grow hyperbolically without bound.

Let us now turn to a new compressor, T. T is a first-order static arithmetic coder. It maintains a table of $|\Sigma|$ separate contexts, and corresponds to modelling data as a first-order Markov chain. In fact, it can be shown that the correspondence between *NCD*, *KL*, and H continues for any finite-order Markov chain (we omit the details).

We have done an experiment to verify these relations. In this experiment, we create files of exact empirical distributions. The alphabet is $\Sigma = \{0,1\}$. The "fixed" file F is set to a $\theta = 0.55$ Bernoulli binary distribution, i.e. F consists of 55% 1s. The other file G is allowed to vary from $0.30 \leq \theta \leq 0.80$. We have used Michael Schindler's Range encoder, R, as a fast and simple arithmetic coder. The results are in Figure 3.1. The graph's horizontal axis represents empirical bias in file G. The $\|hdr\|$ addend is necessary to allow for the empirical discrete finite probability distribution for the input file to be encoded so that during decompression the arithmetic decoder statistics can accurately represent those of the original encoding. There is good agreement between the NCD_R and the prediction based on information radius and maximum entropy. In these

experiments, 200000 symbol files were used for F and G. The deviation between the NCD_R and the information radius (both shown in the graph, with nearly overlapping curves) is on the order of 0.001 bit, and this can be attributed to imperfections in compression, header length, etc.

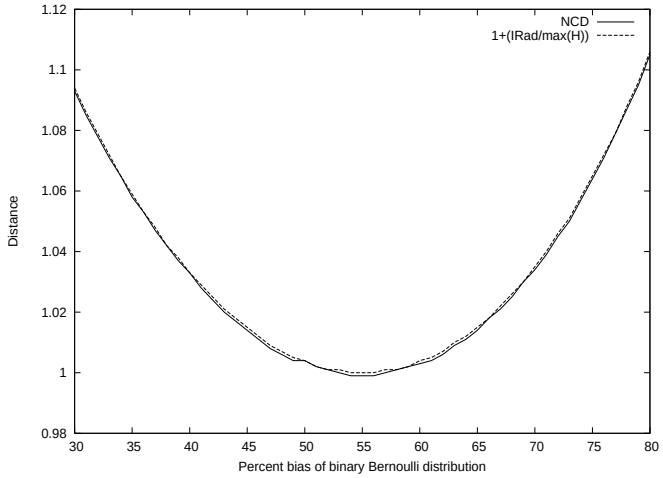

Figure 3.1: A comparison of predicted and observed values for NCD_R.

3.6.2. REMARK. It seems clear that many simple compressors yield simple closed-form formulas for specific variants of NCD. It is not clear whether such a close correspondence between the NCD and KL-divergence (or other simple analytic quantities) still holds in realistic situations, where a sophisticated compressor (such as gzip or ppm, or paq) is used on real-world data. The Shannon entropy and KL-divergence are *expected* codelengths, i.e. theoretical averages taken with respect to some hypothesized distribution. The NCD is based on *actual*, individual sequence codelengths, obtained with some compressor Z. By the Kraft inequality (Chapter 2), Z must correspond to some distribution P such that for all data sequences D of given length,

$$Z(D) = -\log P(D).$$

In case of the static arithmetic encoder, it turned out that this expression could be rewritten as

$$Z(D) = \|hdr\| - \log P_D(D) = \|hdr\| + N_D E_{P_D}[-\log P_D(D)],$$

where the latter equality follows from (3.6.6) and (3.6.7). These two crucial steps, which replace a log-probability of an actually realized sequence by its expectation, allow us to connect NCD with Shannon entropy, and then, KL-divergence. It can readily be seen that a similar replacement can still be done if a fixed-order arithmetic coder is used (corresponding, to, say, k-th order Markov chains). However, the larger k, the larger the size of the header will be, and thus the more data are needed before the size of the header becomes negligible. With real data, not generated by any finite order chain, and modern compressors (which are not fixed-order), it is therefore not clear whether an analogue of (3.6.18) still holds. This would be an interesting topic for future research.

3.7 Conclusion

In this chapter we have introduced the idea of a mathematical distance function and discussed the notion of a similarity metric. We defined NCD and the related NID function, and talked about some properties of each. A strategy for calculating these functions using real compressors was outlined, and a mathematical connection was made between a particular case of NCD and the familiar statistical quantity called KL-divergence.

Chapter 4

A New Quartet Tree Heuristic For Hierarchical Clustering

This chapter is about the quartet method for hierarchical clustering. We introduce the notion of hierarchical clustering in Section 4.3, and then proceed to explain the quartet method in Section 4.4. We address computational complexity issues in Section 4.5.1. Our line of reasoning leads naturally to a simple but effective non-deterministic algorithm to monotonically approximate a best-fitting solution to a given input quartet cost list. We describe the algorithm in Section 4.6.1, with performance analysis in Section 4.6.2. In the remainder of the chapter we present a series of experiments demonstrating the tree building system.

4.1 Summary

We consider the problem of constructing an optimal-weight tree from the $3\binom{n}{4}$ weighted quartet topologies on n objects, where optimality means that the summed weight of the embedded quartet topologies is optimal (so it can be the case that the optimal tree embeds all quartets as non-optimal topologies). We present a heuristic for reconstructing the optimal-weight tree, and a canonical manner to derive the quartet-topology weights from a given distance matrix. The method repeatedly transforms a bifurcating tree, with all objects involved as leaves, achieving a monotonic approximation to the exact single globally optimal tree. This contrasts to other heuristic search methods from biological phylogeny, like DNAML or quartet puzzling, which, repeatedly, incrementally construct a solution from a random order of objects, and subsequently add agreement values. We do not assume that there exists a true bifurcating supertree that embeds each quartet in the optimal topology, or represents the distance matrix faithfully—not even under the assumption that the weights or distances are corrupted by a measuring process. Our aim is to hierarchically cluster the input data as faithfully as possible, both phylogenetic data and data of completely different types. In our experiments with natural data, like genomic data, texts or music, the global optimum appears to be reached. Our method is capable of handling over 100 objects, possibly up to 1000 objects, while no existing quartet heuristic can computationally approximate the exact optimal solution of a quartet tree of more than about 20–30 objects without

running for years. The method is implemented and AVAILABle as public software.

4.2 Introduction

We present a method of hierarchical clustering based on a novel fast randomized hill-climbing heuristic of a new global optimization criterion. Given a the weights of all quartet topologies, or a matrix of the pairwise distances between the objects, we obtain an output tree with the objects as leaves, and we score how well the tree represents the information in the distance matrix on a scale of 0 to 1. As proof of principle, we experiment on three data sets, where we know what the final answer should be: (i) reconstruct a tree from a distance matrix obtained from a randomly generated tree; (ii) reconstruct a tree from files containing artificial similarities; and (iii) reconstruct a tree from natural files of heterogeneous data of vastly different types. We give examples in whole-genome phylogeny using the whole mitochondrial DNA of the species concerned, in SARS virus localization among other virri, and in analyzing the spreading of the bird-flu H5N1 virus mutations. We compare the hierarchical clustering of our method with a more standard method of two-dimensional clustering (to show that our dendrogram method of depicting the clusters is more informative). The new method was developed as an auxiliary tool for [25, 26, 22], since the available quartet tree methods were too slow when they were exact, and too inaccurate or uncertain when they were statistical incremental. Our new quartet tree heuristic runs orders of magnitudes faster than any other exact quartet tree method, and gives consistently good results in practice.

Relation with Previous Work: The Minimum Quartet Tree Cost (MQTC) problem below for which we give a new computational heuristic is related to the Quartet Puzzling problem, [109]. There, the quartet topologies are provided with a probability value, and for each quartet the topology with the highest probability is selected (randomly, if there are more than one) as the maximum-likelihood optimal topology. The goal is to find a bifurcating tree that embeds these optimal quartet topologies. In the biological setting it is assumed that the observed genomic data are the result of an evolution in time, and hence can be represented as the leaves of an evolutionary tree. Once we obtain a proper probabilistic evolutionary model to quantify the evolutionary relations between the data we can search for the true tree. In a quartet method one determines the most likely quartet topology under the given assumptions, and then searches for a tree that represents as many of such topologies as is possible. If the theory and data were perfect then there was a tree that represented precisely all most likely quartet topologies. Unfortunately, in real life the theory is not perfect, the data are corrupted, and the observation pollutes and makes errors. Thus, one has to settle for embedding as many most likely quartet topologies as possible, do error correction on the quartet topologies, and so on. For n objects, there are $(2n-5)!! \equiv (2n-5) \times (2n-3) \times \cdots \times 3$ unrooted bifurcating trees. For n large, exhaustive search for the optimal tree is impossible, and turns out to be NP-hard, and hence infeasible in general. There are two main avenues that have been taken:

(i) Incrementally grow the tree in random order by stepwise addition of objects in the current optimal way, repeat this for different object orders, and add agreement values on the branches, like DNAML [39], or quartet puzzling [109].

(ii) Approximate the global optimum monotonically or compute it, using geometric algorithm or dynamic programming [6], and linear programming [119].

These methods, other methods, as well as methods related to the MQT problem, cannot handle more than 15–30 objects [119, 81, 89, 12] directly, even while using farms of desktops. To handle more objects one needs to construct a supertree from the constituent quartet trees for subsets of the original data sets, [95], as in [81, 89].

In 2003 in [25, 26, 22] we considered a new approach, like [119], and possibly predating it. Our goal was to use a quartet method to obtain high-quality hierarchical clustering of data from arbitrary (possibly heterogeneous) domains, not necessarily phylogeny data. We thus do not assume that there exists a true evolutionary tree, and our aim is not to just embed as many optimal quartet topologies as is possible. Instead, for n objects we consider all $3\binom{n}{4}$ possible quartet topologies, each with a given weight, and our goal is to find the tree such that the summed weights of the embedded quartet topologies is optimal. We develop an heuristic that monotonically approximates this optimum, a figure of merit that quantifies the quality of the best current candidate tree. We show that the problem is NP-hard, but we give evidence that the natural data sets we consider have qualities of smoothness so that the monotonic heuristic obtains the global optimum in a feasible number of steps.

Materials and Methods: Some of the experiments reported are taken from [25, 26, 22] where many more can be found. The data samples we used were obtained from standard data bases accessible on the world-wide web, generated by ourselves, or obtained from research groups in the field of investigation. We supply the details with each experiment. The clustering heuristic generates a tree with an optimality quantification, called standardized benefit score or $S(T)$ value in the sequel. Contrary to other phylogeny methods, we do not have agreement or confidence values on the branches: we generate the best tree possible, globally balancing all requirements. Generating trees from the same distance matrix many times resulted in the same tree in case of high $S(T)$ value, or a similar tree in case of moderately high $S(T)$ value, for all distance matrices we used, even though the heuristic is randomized. That is, there is only one way to be right, but increasingly many ways to be increasingly wrong which can all be realized by different runs of the randomized algorithm. The quality of the results depends on how well the hierarchical tree represents the information in the matrix. That quality is measured by the $S(T)$ value, and is given with each experiment. In certain natural data sets, such as H5N1 genomic sequences, consistently high $S(T)$ values are returned even for large sets of objects of 100 or more nodes. In other discordant natural data sets however, the $S(T)$ value deteriorates more and more with increasing number of elements being put in the same tree. The reason is that with increasing size of a discordant natural data set the projection of the information in the distance matrix into a ternary tree gets necessarily increasingly distorted because the underlying structure in the data is incommensurate with any tree shape whatsoever. In this way, larger structures may induce additional "stress" in the mapping that is visible as lower and lower $S(T)$ scores.

Figures: We use two styles to display the hierarchical clusters. In the case of genomics of Eutherian orders, it is convenient to follow the dendrograms that are customary in that area (suggesting temporal evolution) for easy comparison with the literature. In the other experiments (even the genomic SARS experiment) it is more informative to display an unrooted ternary tree (or binary tree if we think about incoming and outgoing edges) with explicit internal nodes. This

facilitates identification of clusters in terms of subtrees rooted at internal nodes or contiguous sets of subtrees rooted at branches of internal nodes.

4.3 Hierarchical Clustering

Given a set of objects as points in a space provided with a (not necessarily metric) distance measure, the associated *distance matrix* has as entries the pairwise distances between the objects. Regardless of the original space and distance measure, it is always possible to configure n objects in n-dimensional Euclidean space in such a way that the associated distances are identical to the original ones, resulting in an identical distance matrix. This distance matrix contains the pairwise distance relations according to the chosen measure in raw form. But in this format that information is not easily usable, since for $n > 3$ our cognitive capabilities rapidly fail. Just as the distance matrix is a reduced form of information representing the original data set, we now need to reduce the information even further in order to achieve a cognitively acceptable format like data clusters. To extract a hierarchy of clusters from the distance matrix, we determine a dendrogram (ternary tree) that agrees with the distance matrix according to a cost measure. This allows us to extract more information from the data than just flat clustering (determining disjoint clusters in dimensional representation).

Clusters are groups of objects that are similar according to our metric. There are various ways to cluster. Our aim is to analyze data sets for which the number of clusters is not known a priori, and the data are not labeled. As stated in [36], conceptually simple, hierarchical clustering is among the best known unsupervised methods in this setting, and the most natural way is to represent the relations in the form of a dendrogram, which is customarily a directed binary tree or undirected ternary tree. With increasing number of data items, the projection of the distance matrix information into the tree representation format may get distorted. Not all natural data sets exhibit this phenomenon; but for some, the tree gets increasingly distorted as more objects are added. A similar situation sometimes arises in using alignment cost in genomic comparisons. Experience shows that in both cases the hierarchical clustering methods seem to work best for small sets of data, up to 25 items, and to deteriorate for some (but not all) larger sets, say 40 items or more. This deterioration is directly observable in the $S(T)$ score and degrades solutions in two common forms: tree instability when different or very different solutions are returned on successive runs or tree "overlinearization" when some data sets produce caterpillar-like structures only or predominantly. In case a large set of objects, say 100 objects, clusters with high $S(T)$ value this is evidence that the data are of themselves tree-like, and the quartet-topology weights, or underlying distances, truly represent to similarity relationships between the data.

4.4 The Quartet Method

Given a set N of n objects, we consider every set of four elements from our set of n elements; there are $\binom{n}{4}$ such sets. From each set $\{u, v, w, x\}$ we construct a tree of arity 3, which implies that the tree consists of two subtrees of two leaves each. Let us call such a tree a *quartet topology*. The

set of $3\binom{n}{4}$ quartet topologies induced by N is denoted by Q. We denote a partition $\{u,v\},\{w,x\}$ of $\{u,v,w,x\}$ by $uv|wx$. There are three possibilities to partition $\{u,v,w,x\}$ into two subsets of two elements each: (i) $uv|wx$, (ii) $uw|vx$, and (iii) $ux|vw$. In terms of the tree topologies: a vertical bar divides the two pairs of leaf nodes into two disjoint subtrees (Figure 4.1).

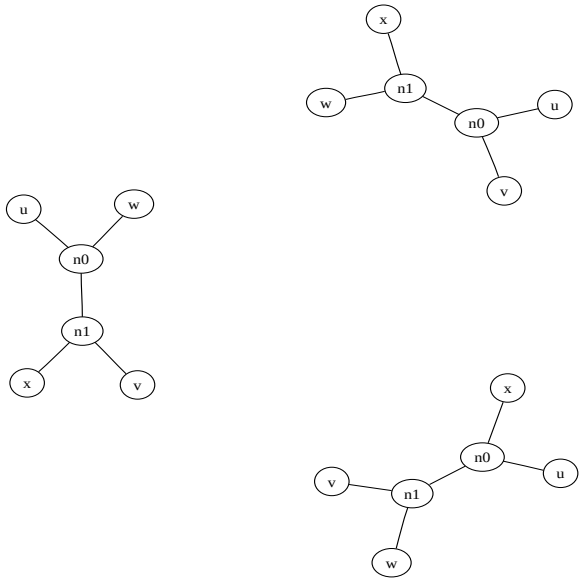

Figure 4.1: The three possible quartet topologies for the set of leaf labels *u,v,w,x*.

4.4.1. DEFINITION. Define a *binary dendrogram* as an element from the class $\mathcal{T}$ of undirected trees of arity 3 with $n \geq 4$ leaves, labeled with the elements of N.

Such trees have n leaves and $n-2$ internal nodes. For any given tree T from this class, and any set of four leaf labels $u,v,w,x \in N$, we say T is *consistent* with $uv|wx$ if and only if the path from u to v does not cross the path from w to x. It is easy to see that precisely one of the three possible quartet topologies for any set of 4 labels is consistent for a given tree from the above class, and therefore a tree from $\mathcal{T}$ contains precisely $\binom{n}{4}$ different quartet topologies. We may think of a large tree having many smaller quartet topologies embedded within its structure. Commonly the goal in the quartet method is to find (or approximate as closely as possible) the tree that embeds the maximal number of consistent (possibly weighted) quartet topologies from a given set $P \subseteq Q$ of quartet topologies [53] (Figure 4.2). A *weight function* $W : P \rightarrow \mathcal{R}$, with $\mathcal{R}$ the set of real numbers determines the weights. The unweighted case is when $W(uv|wx) = 1$ for all $uv|wx \in P$.

4.4.2. DEFINITION. The (weighted) *Maximum Quartet Consistency (MQC)* is defined as follows:

GIVEN: N, P, and W.

QUESTION: Find $T_0 = \max_T \sum\{W(uv|wx) : uv|wx \in P \text{ and } uv|wx \text{ is consistent with } T\}$.

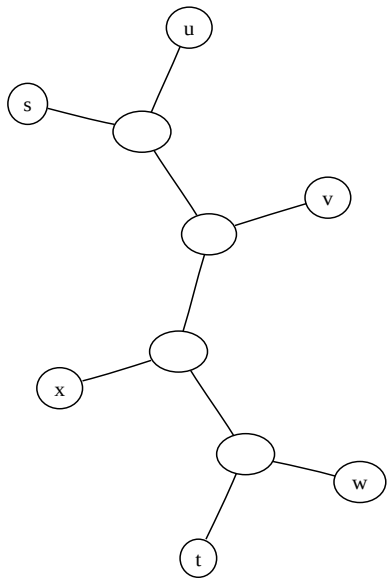

Figure 4.2: An example tree consistent with quartet topology $uv|wx$.

4.5 Minimum Quartet Tree Cost

The rationale for the MQC optimization problem is the assumption that there is exists a tree T_0 as desired in the class $\mathcal{T}$ under consideration, and our only problem is to find it. This assumption reflects the genesis of the method in the phylogeny community. Under the assumption that biological species developed by evolution in time, and N is a subset of the now existing species, there is a phylogeny P (tree in $\mathcal{T}$) that represents that evolution. The set of quartet topologies consistent with this tree, has one quartet topology per quartet which is the true one. The quartet topologies in P are the ones which we assume to be among the true quartet topologies, and weights are used to express our relative certainty about this assumption concerning the individual quartet topologies in P.

However, the data may be corrupted so that this assumption is no longer true. In the general case of hierarchical clustering we do not even have a priori knowledge that certain quartet topologies are objectively true and must be embedded. Rather, we are in the position that we can somehow assign a relative importance to the different quartet topologies. Our task is then to balance the importance of embedding different quartet topologies against one another, leading to a tree that represents the concerns as well as possible. We start from a cost-assignment to the quartet topologies; the method by which we assign costs to the $3\binom{n}{4}$ quartet topologies is for now immaterial to our problem. Given a set N of n objects, let Q be the set of quartet topologies, and let $C : Q \to \mathcal{R}$ be a *cost function* assigning a real valued cost $C_{uv|wx}$ to each quartet $uv|wx \in Q$.

4.5.1. DEFINITION. The *cost* C_T of a tree T with a set N of leaves (external nodes of degree 1) is defined by $C_T = \sum_{\{u,v,w,x\}\subseteq N}\{C_{uv|wx} : T \text{ is consistent with } uv|wx\}$—the sum of the costs of all its consistent quartet topologies.

4.5.2. DEFINITION. Given N and C, the *Minimum Quartet Tree Cost (MQTC)* is $\min_T\{C_T : T$ is a tree with the set N labeling its leaves$\}$.

We normalize the problem of finding the MQTC as follows: Consider the list of all possible quartet topologies for all four-tuples of labels under consideration. For each group of three possible quartet topologies for a given set of four labels u,v,w,x, calculate a best (minimal) cost

$$m(u,v,w,x) = \min\{C_{uv|wx}, C_{uw|vx}, C_{ux|vw}\}$$

and a worst (maximal) cost $M(u,v,w,x) = \max\{C_{uv|wx}, C_{uw|vx}, C_{ux|vw}\}$. Summing all best quartet topologies yields the best (minimal) cost $m = \sum_{\{u,v,w,x\}\subseteq N} m(u,v,w,x)$. Conversely, summing all worst quartet topologies yields the worst (maximal) cost $M = \sum_{\{u,v,w,x\}\subseteq N} M(u,v,w,x)$. For some distance matrices, these minimal and maximal values can not be attained by actual trees; however, the score C_T of every tree T will lie between these two values. In order to be able to compare the scores of quartet trees for different numbers of objects in a uniform way, we now rescale the score linearly such that the worst score maps to 0, and the best score maps to 1:

4.5.3. DEFINITION. The *normalized tree benefit score* $S(T)$ is defined by

$$S(T) = (M - C_T)/(M - m).$$

Our goal is to find a full tree with a maximum value of $S(T)$, which is to say, the lowest total cost. Now we can rephrase the MQTC problem in such a way that solutions of instances of different sizes can be uniformly compared in terms of relative quality:

4.5.4. DEFINITION. Definition of the *MQTC problem*:
GIVEN: N and C.
QUESTION: Find a tree T_0 with $S(T_0) = \max\{S(T) : T$ is a tree with the set N labeling its leaves$\}$.

4.5.1 Computational Hardness

The hardness of Quartet Puzzling is informally mentioned in the literature [119, 81, 89], but we provide explicit proofs. To express the notion of computational difficulty one uses the notion of "nondeterministic polynomial time (NP)". If a problem concerning n objects is NP-hard this means that the best known algorithm for this (and a wide class of significant problems) requires computation time exponential in n. That is, it is infeasible in practice. The *MQC decision problem* is the following: Given a set N of n objects, let T be a tree of which the n leaves are labeled by the objects, and let Q be the set of quartet topologies and Q_T be the set of quartet topologies embedded in T. Given a set of quartet topologies $P \subseteq Q$, and an integer k, the problem is to decide whether there is a binary tree T such that $P \bigcap Q_T > k$. In [108] it is shown that the MQC decision problem is NP-hard. We have formulated the NP-hardness of the so-called *incomplete* MQC decision problem, the less general *complete MQC decision problem* requires P to contain precisely one quartet topology per quartet out of N, and is proven to be NP-hard as well in [12].

4.5.5. THEOREM. *The MQTC decision problem is NP-hard.*

PROOF. By reduction from the MQC decision problem. For every MQC decision problem one can define a corresponding MQTC decision problem that has the same solution: give the quartet topologies in P cost 0 and the ones in $Q-P$ cost 1. Consider the MQTC decision problem: is there a tree T with the set N labeling its leaves such that $C_T < \binom{n}{4} - k$? An alternative equivalent formulation is: is there a tree T with the set N labeling its leaves such that

$$S(T) > \frac{M - \binom{n}{4} + k}{M - m}?$$

Note that every tree T with the set N labeling its leaves has precisely one out of the three quartet topologies of every of the $\binom{n}{4}$ quartets embedded in it. Therefore, the cost $C_T = \binom{n}{4} - |P \bigcap Q_T|$. If the answer to the above question is affirmative, then the number of quartet topologies in P that are embedded in the tree exceeds k; if it is not then there is no tree such that the number of quartet topologies in P embedded in it exceeds k. This way the MQC decision problem can be reduced to the MQTC decision problem, which shows also the latter to be NP-hard. □

Is it possible that the best $S(T)$ value is always one, that is, there always exists a tree that embeds all quartets at minimum cost quartet topologies? Consider the case $n = |N| = 4$. Since there is only one quartet, we can set T_0 equal to the minimum cost quartet topology, and have $S(T_0) = 1$. A priori we cannot exclude the possibility that for every N and C there always is a tree T_0 with $S(T_0) = 1$. In that case, the MQTC Problem reduces to finding that T_0. However, the situation turns out to be more complex. Note first that the set of quartet topologies uniquely determines a tree in $\mathcal{T}$, [15].

4.5.6. LEMMA. *Let T, T' be different labeled trees in $\mathcal{T}$ and let $Q_T, Q_{T'}$ be the sets of embedded quartet topologies, respectively. Then, $Q_T \neq Q_{T'}$.*

A *complete set* of quartet topologies on N is a set containing precisely one quartet topology per quartet. There are $3^{\binom{n}{4}}$ such combinations, but only $2^{\binom{n}{2}}$ labeled undirected graphs on n nodes (and therefore $|\mathcal{T}| \leq 2^{\binom{n}{2}}$). Hence, not every complete set of quartet topologies corresponds to a tree in $\mathcal{T}$. This already suggests that we can weight the quartet topologies in such a way that the full combination of all quartet topologies at minimal costs does not correspond to a tree in $\mathcal{T}$, and hence $S(T_0) < 1$ for $T_0 \in \mathcal{T}$ realizing the MQTC optimum. For an explicit example of this, we use that a complete set corresponding to a tree in $\mathcal{T}$ must satisfy certain transitivity properties, [29, 28]:

4.5.7. LEMMA. *Let T be a tree in the considered class with leaves N, Q the set of quartet topologies and $Q_0 \subseteq Q$. Then Q_0 uniquely determines T if*

(i) Q_0 contains precisely one quartet topology for every quartet, and

(ii) For all $\{a,b,c,d,e\} \subseteq N$, if $ab|bc, ab|de \in Q$ then $ab|ce \in Q$, as well as if $ab|cd, bc|de \in Q$ then $ab|de \in Q$.

4.5.8. THEOREM. *There are N (with $n = |N| = 5$) and a cost function C such that, for every $T \in \mathcal{T}$, $S(T)$ does not exceed $4/5$.*

PROOF. Consider $N = \{u, v, w, x, y\}$ and $C(uv|wx) = 1 - \varepsilon(\varepsilon > 0), C(uw|xv) = C(ux|vw) = 0$, $C(xy|uv) = C(wy|uv) = C(uy|wx) = C(vy|wx) = 0$, and $C(ab|cd) = 1$ for all remaining quartet topologies $ab|cd \in Q$. We see that $M = 5 - \varepsilon$, $m = 0$. The tree $T_0 = (y, ((u, v), (w, x)))$ has cost $C_{T_0} = 1 - \varepsilon$, since it embeds quartet topologies $uw|xv, xy|uv, wy|uv, uy|wx, vy|wx$. We show that T_0 achieves the MQTC optimum. *Case 1:* If a tree $T \neq T_0$ embeds $uv|wx$, then it must by Item (i) of Lemma 4.5.7 also embed a quartet topology containing y that has cost 1.

Case 2: If a tree $T \neq T_0$ embeds $uw|xv$ and $xy|uv$, then it must by Item (ii) of the Lemma 4.5.7 also embed $uw|xy$, and hence have cost $C_T \geq 1$. Similarly, all other remaining cases of embedding a combination of a quartet topology not containing y of 0 cost with a quartet topology containing y of 0 cost in T, imply an embedded quartet topology of cost 1 in T. □

Altogether, the MQTC optimization problem is infeasible in practice, and natural data can have an optimal $S(T) < 1$. In fact, it follows from the above analysis that to determine $S(T)$ in general is NP-hard. In [12] a polynomial time approximation scheme for complete MQC is exhibited, a theoretical approximation scheme allowing the approximation of the optimal solution up to arbitrary precision, with running time polynomial in the inverse of that precision. We say "theoretical" since that algorithm would run in something like n^{19}. For incomplete MQC it is shown that even such a theoretical algorithm does not exist, unless P=NP. Hence, computation of the MQTC optimum, and even its approximation with given precision, requires superpolynomial time unless P=NP. Therefore, any practical approach to obtain or approximate the MQTC optimum requires heuristics.

4.6 New Heuristic

Our algorithm is essentially randomized hill-climbing, using parallellized Genetic Programming, where undirected trees evolve in a random walk driven by a prescribed fitness function. We are given a set N of n objects and a weighting function W.

4.6.1. DEFINITION. We define a *simple mutation* on a labeled undirected ternary tree as one of three possible transformations:

1. A *leaf swap*, which consists of randomly choosing two leaf nodes and swapping them.
2. A *subtree swap*, which consists of randomly choosing two internal nodes and swapping the subtrees rooted at those nodes.
3. A *subtree transfer*, whereby a randomly chosen subtree (possibly a leaf) is detached and reattached in another place, maintaining arity invariants.

Each of these simple mutations keeps the number of leaf nodes and internal nodes in the tree invariant; only the structure and placements change.

4.6.2. DEFINITION. A *k-mutation* is a sequence of k simple mutations. Thus, a simple mutation is a 1-mutation.

4.6.1 Algorithm

Step 1: First, a random tree $T \in \mathcal{T}$ with $2n-2$ nodes is created, consisting of n leaf nodes (with 1 connecting edge) labeled with the names of the data items, and $n-2$ non-leaf or *internal* nodes labeled with the lowercase letter "k" followed by a unique integer identifier. Each internal node has exactly three connecting edges.

Step 2: For this tree T, we calculate the total cost of all embedded quartet topologies, compute $S(T)$.

Comment: A tree is consistent with precisely $\frac{1}{3}$ of all quartet topologies, one for every quartet. A random tree is likely to be consistent with about $\frac{1}{3}$ of the best quartet topologies—but this is necessarily imprecise because of dependencies.

Step 3: The *currently best known tree* variable T_0 is set to T: $T_0 \leftarrow T$.

Comment: This T_0 is used as the basis for further searching.

Step 4: Pick a number k with probability $p(k) = c/(k(\log k)^2)$ where $1/c = \sum_{k=1}^{\infty} 1/(k(\log k)^2)$.

Comment: This number k is the number of simple mutations that we will perform in the next k-mutation. The probability distribution $p(k)$ is easily generated by running a random fair bit generator and set k to the length of the first self-delimiting sequence generated. That is, if $x = x_1 \ldots x_k \in \{0,1\}^k$ ($|x| = k \geq 1$), then $\bar{x} = 1^{k-1}0x$, $x' = \overline{|x|}x$, and $x'' = \overline{|x'|}x'$. Thus, the length $|x''| = k + \log k + 2\log\log k$. The probability of generating x'' corresponding to a given x of length k by fair coin flips is $2^{-|x''|} = 2^{-k-\log k-2\log\log k} = 2^{-k}/(k(\log k)^2)$. The probability of generating x'' corresponding to *some* x of length k is 2^k times as large, that is, $1/(k(\log k)^2)$. In practice, we used a "shifted" fat tail distribution $1/((k+2)(\log k+2)^2)$

Step 5: Compose a k-mutation by, for each such simple mutation, choosing one of the three types listed above with equal probability. For each of these simple mutations, we uniformly at random select leaves or internal nodes, as appropriate.

Comment: Notice that trees which are close to the original tree (in terms of number of simple mutation steps in between) are examined often, while trees that are far away from the original tree will eventually be examined, but not very frequently.

Step 6: In order to search for a better tree, we simply apply the k-mutation constructed in **Step 5** on T_0 to obtain T', and then calculate $S(T')$. If $S(T') \geq S(T_0)$, then replace the current candidate in T_0 by T (as the new best tree): $T_0 \leftarrow T$.

Step 7: If $S(T_0) = 1$ or a **termination condition** to be discussed below holds, then output the tree in T_0 as the best tree and halt. Otherwise, go to **Step 4**.

4.6.3. REMARK. We have chosen $p(k)$ to be a "fat-tail" distribution, with the fattest tail possible, so that we may concentrate maximal probability also on the larger values of k. That way, the likelihood of getting trapped in local minima is minimized. In contrast, if one would choose an exponential scheme, like $q(k) = ce^{-k}$, then the larger values of k would arise so scarcely that practically speaking the distinction between being absolutely trapped in a local optimum, and the very low escape probability, would be insignificant. Considering positive-valued probability mass functions $q : \mathcal{N} \to (0,1]$, with $\mathcal{N}$ the natural numbers, as we do here, we note that such a function (i) $\lim_{k\to\infty} q(k) = 0$, and (ii) $\sum_{k=1}^{\infty} q(k) = 1$. Thus, every function of the natural numbers that has strictly positive values and converges can be normalized to such a probability mass

function. For smooth analytic functions that can be expressed a series of fractional powers and logarithms, the borderline between converging and diverging is as follows: $\sum 1/k, \sum 1/(k\log k)$, $\sum 1/(k\log k\log\log k)$ and so on diverge, while $\sum 1/k^2, \sum 1/(k(\log k)^2), \sum 1/(k\log k(\log\log k)^2)$ and so on converge. Therefore, the maximal fat tail of a "smooth" function $f(x)$ with $\sum f(x) < \infty$ arises for functions at the edge of the convergence family. The distribution $p(k) = c/(k(\log k)^2)$ is as close to the edge as is reasonable, and because the used coding $x \to x''$ is a prefix code we have $\sum 1/(k(\log k)^2) \leq 1$ by the Kraft Inequality (see for example [79]) and therefore $c \geq 1$. Let us see what this means for our algorithm using the chosen distribution $p(k)$. For $N = 64$, say, we can change any tree in $\mathcal{T}$ to any other tree in $\mathcal{T}$ with a 64-mutation. The probability of such a complex mutation occurring is quite large with such a fat tail: $1/(64 \cdot 6^2) = 1/2304$, that is, more than 40 times in 100,000 generations. If we can get out of a local minimum with already a 32-mutation, then this occurs with probability at least $1/800$, so 125 times, and with a 16-mutation with probability at least $1/196$, so 510 times.

4.6.2 Performance

The main problem with hill-climbing algorithms is that they can get stuck in a local optimum. However, by randomly selecting a sequence of simple mutations, longer sequences with decreasing probability, we essentially run a Metropolis Monte Carlo algorithm [83], reminiscent of simulated annealing [56] at random temperatures. Since there is a nonzero probability for every tree in $\mathcal{T}$ being transformed into every other tree in $\mathcal{T}$, there is zero probability that we get trapped forever in a local optimum that is not a global optimum. That is, trivially:

4.6.4. LEMMA. *(i) The algorithm approximates the MQTC optimal solution monotonically in each run.*

(ii) The algorithm without termination condition solves the MQTC optimization problem eventually with probability 1 (but we do not in general know when the optimum has been reached in a particular run).

The main question therefore is the convergence speed of the algorithm on natural data, and a termination criterion to terminate the algorithm when we have an acceptable approximation. From the impossibility result in [12] we know that there is no polynomial approximation scheme for MQTC optimization, and whether our scheme is expected polynomial time seems to require proving that the involved Metropolis chain is rapidly mixing [116], a notoriously hard and generally unsolved problem. In practice, in our experiments there is unanimous evidence that for the natural data and the weighting function we have used, convergence is always fast. We have to determine the cost of $\binom{n}{4}$ quartets to determine each $S(T)$ value. Hence the algorithm runs in time at least that much. In experiments we found that for the same data set different runs consistently showed the same behavior, for example Figure 4.3 for a 60-object computation. There the $S(T)$ value leveled off at about 70,000 examined trees, and the termination condition was "no improvement in 5,000 trees." Different random runs of the algorithm nearly always gave the same behavior, returning a tree with the same $S(T)$ value, albeit a different tree in most cases with here $S(T) \approx 0.865$, a relatively low value. That is, since there are many ways to find a tree of optimal $S(T)$ value, and apparently the algorithm never got trapped in a lower local optimum.

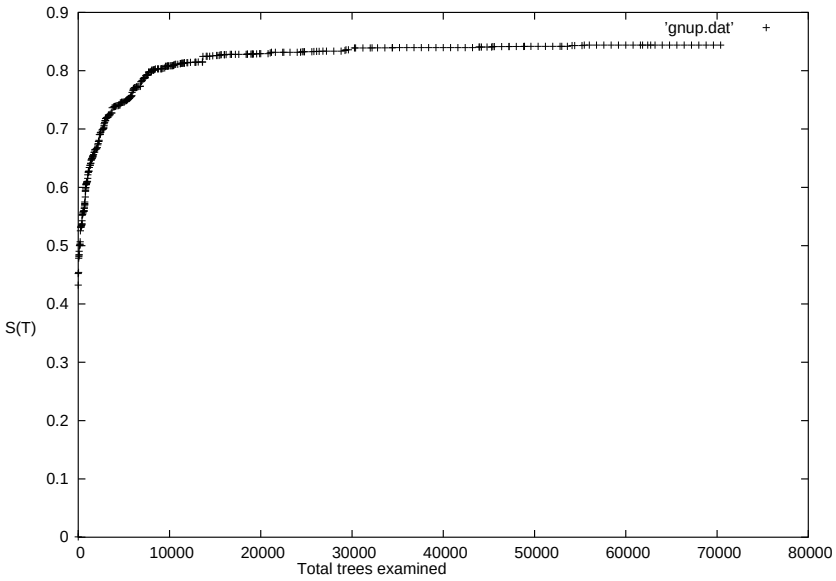

Figure 4.3: Progress of a 60-item data set experiment over time.

For problems with high $S(T)$ value, as we see later, the algorithm consistently returned the same tree. This situation is perhaps similar to the behavior of the Simplex method in linear programming, that can be shown to run in exponential time on a badly chosen problem instance, but in practice on natural problems consistently runs in linear time.

Note that if a tree is ever found such that $S(T) = 1$, then we can stop because we can be certain that this tree is optimal, as no tree could have a lower cost. In fact, this perfect tree result is achieved in our artificial tree reconstruction experiment (Section 4.6.5) reliably in a few minutes. For real-world data, $S(T)$ reaches a maximum somewhat less than 1, presumably reflecting distortion of the information in the distance matrix data by the best possible tree representation, as noted above, or indicating getting stuck in a local optimum or a search space too large to find the global optimum. On many typical problems of up to 40 objects this tree-search gives a tree with $S(T) \geq 0.9$ within half an hour. For large numbers of objects, tree scoring itself can be slow: as this takes order n^4 computation steps. Current single computers can score a tree of this size in about a minute. Additionally, the space of trees is large, so the algorithm may slow down substantially. For larger experiments, we used the C program called partree (part of the CompLearn package [21]) with MPI (Message Passing Interface, a common standard used on massively parallel computers) on a cluster of workstations in parallel to find trees more rapidly. We can consider the graph mapping the achieved $S(T)$ score as a function of the number of trees examined. Progress occurs typically in a sigmoidal fashion towards a maximal value ≤ 1, Figure 4.3.

4.6.3 Termination Condition

The *termination condition* is of two types and which type is used determines the number of objects we can handle.

Simple termination condition: We simply run the algorithm until it seems no better trees are being found in a reasonable amount of time. Here we typically terminate if no improvement in $S(T)$ value is achieved within 100,000 examined trees. This criterion is simple enough to enable us to hierarchically cluster data sets up to 80 objects in a few hours. This is way above the 15–30 objects in the previous exact (non-incremental) methods (see Introduction).

Agreement termination condition: In this more sophisticated method we select a number $2 \leq r \leq 6$ of runs, and we run r invocations of the algorithm in parallel. Each time an $S(T)$ value in run $i = 1, \ldots, r$ is increased in this process it is compared with the $S(T)$ values in all the other runs. If they are all equal, then the candidate trees of the runs are compared. This can be done by simply comparing the ordered lists of embedded quartet topologies, in some standard order, since the set of embedded quartet topologies uniquely determines the quartet tree by [15]. If the r candidate trees are identical, then terminate with this quartet tree as output, otherwise continue the algorithm.

This termination condition takes (for the same number of steps per run) about r times as long as the simple termination condition. But the termination condition is much more rigorous, provided we choose r appropriate to the number n of objects being clustered. Since all the runs are randomized independently at startup, it seems very unlikely that with natural data all of them get stuck in the same local optimum with the same quartet tree instance, provided the number n of objects being clustered is not too small. For $n = 5$ and the number of invocations $r = 2$, there is a reasonable probability that the two different runs by chance hit the same tree in the same step. This phenomenon leads us to require more than two successive runs with exact agreement before we may reach a final answer for small n. In the case of $4 \leq n \leq 5$, we require 6 dovetailed runs to agree precisely before termination. For $6 \leq n \leq 9$, $r = 5$. For $10 \leq n \leq 15$, $r = 4$. For $16 \leq n \leq 17$, $r = 3$. For all other $n \geq 18$, $r = 2$. This yields a reasonable tradeoff between speed and accuracy.

It is clear that there is only one tree with $S(T) = 1$ (if that is possible for the data), and random trees (the majority of all possible quartet trees) have $S(T) \approx 1/3$ (above). This gives evidence that the number of quartet trees with large $S(T)$ values is much smaller than the number of trees with small $S(T)$ values. It is furthermore evident that the precise relation depends on the data set involved, and hence cannot be expressed by a general formula without further assumptions on the data. However, we can safely state that small data sets, of say ≤ 15 objects, that in our experience often lead to $S(T)$ values close to 1 have very few quartet trees realizing the optimal $S(T)$ value. On the other hand, those large sets of 60 or more objects that contain some inconsistency and thus lead to a low final $S(T)$ value also tend to exhibit more variation as one might expect. This suggests that in the agreement termination method each run will get stuck in a different quartet tree of a similar $S(T)$ value, so termination with the same tree is not possible. Experiments show that with the rigorous agreement termination we can handle sets of up to 40 objects, and with the simple termination up to at least 80 objects on a single computer or 100-200 objects using a cluster of computers in parallel.

4.6.4 Tree Building Statistics

We used the CompLearn package, (further described in Chapter 10) [21], to analyze a "10-mammals" example with *zlib* compression yielding a 10×10 distance matrix, similar to the examples in Section 4.10. The algorithm starts with four randomly initialized trees. It tries to improve each one randomly and finishes when they match. Thus, every run produces an output tree, a maximum score associated with this tree, and has examined some total number of trees, T, before it finished. Figure 4.4 shows a graph displaying a histogram of T over one thousand runs of the distance matrix. The *x*-axis represents a number of trees examined in a single run of the program, measured in thousands of trees and binned in 1000-wide histogram bars. The maximum number is about 12000 trees examined. The graph suggests a Poisson distribution.

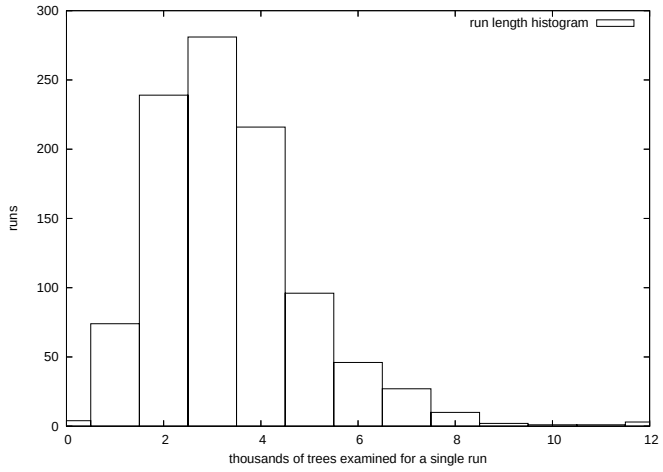

Figure 4.4: Histogram of run-time number of trees examined before termination.

About $2/3$rd of the trials take less than 4000 trees. In the thousand trials above, 994 ended with the optimal $S(T) = 0.999514$. The remaining six runs returned 5 cases of the second-highest score, $S(T) = 0.995198$ and one case of $S(T) = 0.992222$. It is important to realize that outcome stability is dependent on input matrix particulars.

Another interesting distribution is the mutation stepsize. Recall that the mutation length is drawn from a shifted fat-tail distribution. But if we restrict our attention to just the mutations that improve the $S(T)$ value, then we may examine these statistics to look for evidence of a modification to this distribution due to, for example, the presence of very many isolated areas that have only long-distance ways to escape. Figure 4.5 shows the histogram of successful mutation lengths (that is, number of simple mutations composing a single complex mutation) and rejected lengths (both normalized) which shows that this is not the case. Here the *x*-axis is the number of mutation steps and the *y*-axis is the normalized proportion of times that step size occurred. This gives good empirical evidence that in this case, at least, we have a relatively easy search space, without large gaps.

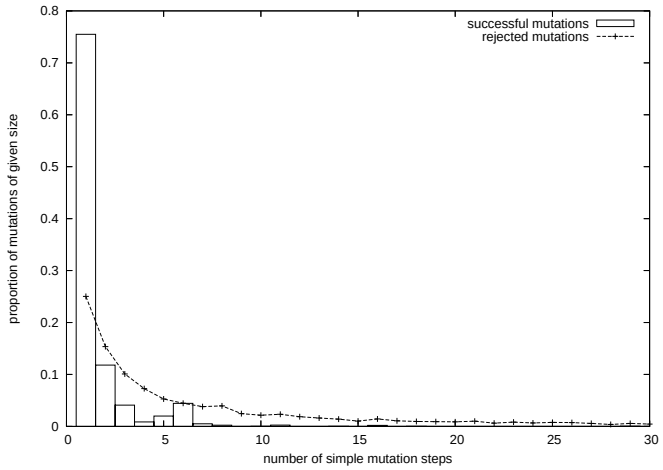

Figure 4.5: Histogram comparing distributions of k-mutations per run.

4.6.5 Controlled Experiments

With natural data sets, say music data, one may have the preconception (or prejudice) that music by Bach should be clustered together, music by Chopin should be clustered together, and so should music by rock stars. However, the preprocessed music files of a piece by Bach and a piece by Chopin, or the Beatles, may resemble one another more than two different pieces by Bach—by accident or indeed by design and copying. Thus, natural data sets may have ambiguous, conflicting, or counterintuitive outcomes. In other words, the experiments on natural data sets have the drawback of not having an objective clear "correct" answer that can function as a benchmark for assessing our experimental outcomes, but only intuitive or traditional preconceptions. We discuss three experiments that show that our program indeed does what it is supposed to do—at least in artificial situations where we know in advance what the correct answer is.

4.7 Quartet Topology Costs Based On Distance Matrix

Given a distance matrix, with entries giving the pairwise distances between the objects, we want to form a hierarchical cluster by representing the objects as leaves of a ternary tree representing the distances in the matrix as faithfully as possible. It is important that we do not assume that there is a true tree; rather, we want to model the data as well as possible. The cost of a quartet topology is defined as the sum of the distances between each pair of neighbors; that is, $C_{uv|wx} = d(u,v) + d(w,x)$. This seems most natural given a distance matrix. In the next section, we review in brief the most common inputs to the quartet tree algorithm as used in this thesis. This information is more thoroughly covered in other chapters.

4.7.1 Distance Measure Used

Recall that the problem of clustering data consists of two parts: (i) extracting a distance matrix from the data, and (ii) constructing a tree from the distance matrix using our novel quartet based heuristic. To check the new quartet tree method in action we use a new compression based distance, called NCD. The theoretical foundation and notional antecedent for NCD was developed by Li and Vitanyi *et al.* [75, 77] as a normalized version of the "information metric" of [79, 9]. Roughly speaking, two objects are deemed close if we can significantly "compress" one given the information in the other, the idea being that if two pieces are more similar, then we can more succinctly describe one given the other. The mathematics used is based on Kolmogorov complexity theory [79]. In [77] we defined a new class of (possibly non-metric) distances, taking values in $[0,1]$ and appropriate for measuring effective similarity relations between sequences, say one type of similarity per distance, and *vice versa*. It was shown that an appropriately "normalized" information metric minorizes every distance in the class. It discovers all effective similarities in the sense that if two objects are close according to some effective similarity, then they are also close according to the normalized information distance. Put differently, the normalized information distance represents similarity according to the dominating shared feature between the two objects being compared. In comparisons of more than two objects, different pairs may have different dominating features. The normalized information distance is a metric and takes values in $[0,1]$; hence it may be called *"the" similarity metric*. To apply this ideal precise mathematical theory in real life, we have to replace the use of the uncomputable Kolmogorov complexity by an approximation using a standard real-world compressor, resulting in the NCD, see [22]. This has been used in the first completely automatic construction of the phylogeny tree based on whole mitochondrial genomes, [75, 80, 77], a completely automatic construction of a language tree for over 50 Euro-Asian languages [77], detects plagiarism in student programming assignments [74], gives phylogeny of chain letters [10], and clusters music [26, 25], Analyzing network traffic and worms using compression [118], and many more topics [22]. The method turns out to be robust under change of the underlying compressor-types: statistical (PPMZ), Lempel-Ziv based dictionary (gzip), block based (bzip2), or special purpose (Gencompress).

4.7.2 CompLearn Toolkit

Oblivious to the problem area concerned, simply using the distances according to the NCD above, the method described in this thesis fully automatically classifies the objects concerned. The method has been released in the public domain as open-source software: The CompLearn Toolkit [21] is a suite of simple utilities that one can use to apply compression techniques to the process of discovering and learning patterns in completely different domains, and hierarchically cluster them using the new quartet method described in this thesis. In fact, this method is so general that it requires no background knowledge about any particular subject area. There are no domain-specific parameters to set, and only a handful of general settings.

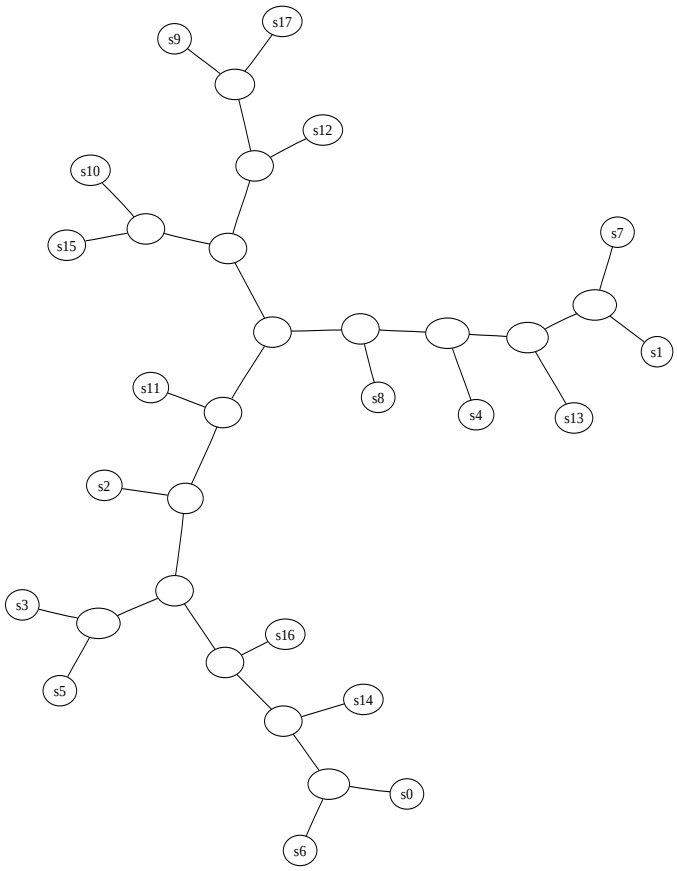

Figure 4.6: The randomly generated tree that our algorithm reconstructed. $S(T) = 1$.

4.7.3 Testing The Quartet-Based Tree Construction

We first test whether the quartet-based tree construction heuristic is trustworthy: We generated a ternary tree T with 18 leaves, using the pseudo-random number generator "rand" of the Ruby programming language, and derived a metric from it by defining the distance between two nodes as follows: Given the length of the path from a to b, in an integer number of edges, as $L(a,b)$, let

$$d(a,b) = \frac{L(a,b)+1}{18},$$

except when $a = b$, in which case $d(a,b) = 0$. It is easy to verify that this simple formula always gives a number between 0 and 1, and is monotonic with path length. Given only the 18×18 matrix of these normalized distances, our quartet method exactly reconstructed the original tree T represented in Figure 4.6, with $S(T) = 1$.

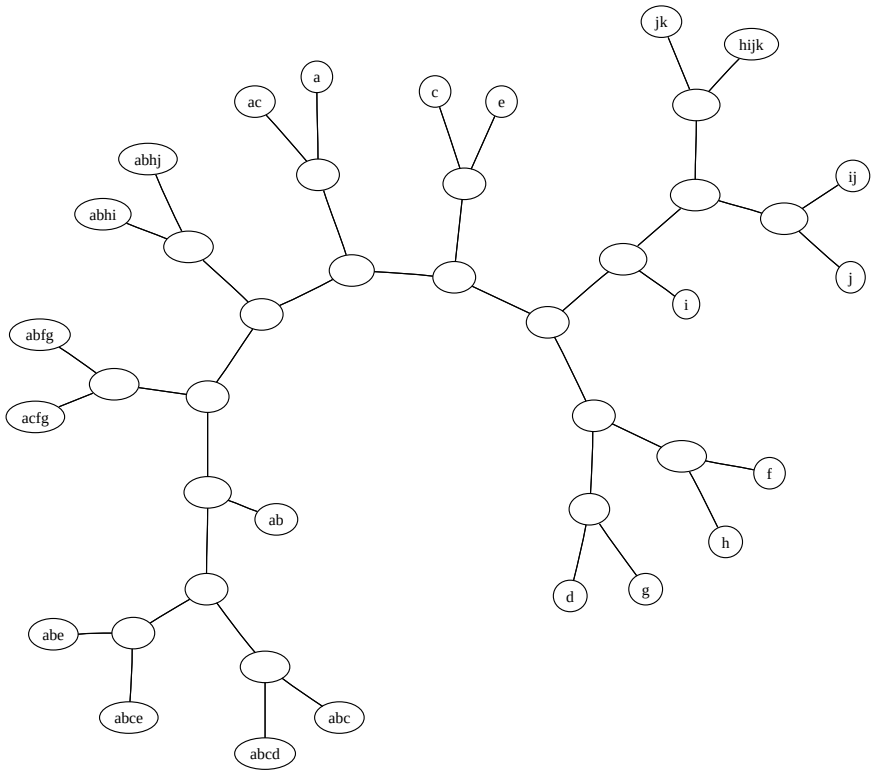

Figure 4.7: Classification of artificial files with repeated 1-kilobyte tags. Not all possibilities are included; for example, file "*b*" is missing. $S(T) = 0.905$.

4.8 Testing On Artificial Data

Given that the tree reconstruction method is accurate on clean consistent data, we tried whether the full procedure works in an acceptable manner when we know what the outcome should be like. We used the "rand" pseudo-random number generator from the C programming language standard library under Linux. We randomly generated 11 separate 1-kilobyte blocks of data where each byte was equally probable and called these *tags*. Each tag was associated with a different lowercase letter of the alphabet. Next, we generated 22 files of 80 kilobyte each, by starting with a block of purely random bytes and applying one, two, three, or four different tags on it. Applying a tag consists of ten repetitions of picking a random location in the 80-kilobyte file, and overwriting that location with the globally consistent tag that is indicated. So, for instance, to create the file referred to in the diagram by "a," we start with 80 kilobytes of random data, then pick ten places to copy over this random data with the arbitrary 1-kilobyte sequence identified as tag *a*. Similarly, to create file "ab," we start with 80 kilobytes of random data, then pick ten places to put copies of tag *a*, then pick ten more places to put copies of tag *b* (perhaps overwriting some of the *a* tags). Because we never use more than four different tags, and therefore never place more than 40 copies of tags, we can expect that at least half of the data in each file is random and uncorrelated with the rest of the files. The rest of the file is correlated

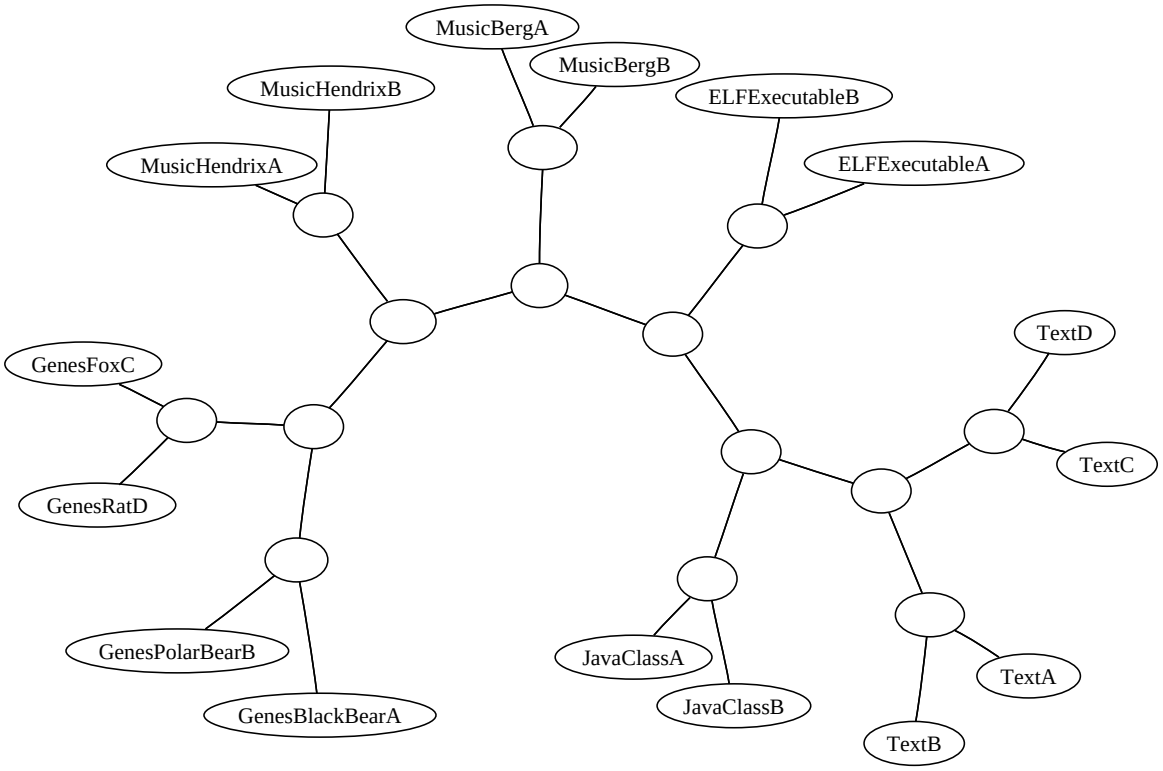

Figure 4.8: Classification of different file types. Tree agrees exceptionally well with NCD distance matrix: $S(T) = 0.984$.

with other files that also contain tags in common; the more tags in common, the more related the files are. The compressor used to compute the NCD matrix was bzip2. The resulting tree is given in Figure 4.7; it can be seen that the clustering has occurred exactly as we would expect. The $S(T)$ score is 0.905.

We will provide more examples of natural data later in this thesis.

4.9 Testing On Heterogeneous Natural Data

We test gross classification of files based on heterogeneous data of markedly different file types: (i) Four mitochondrial gene sequences, from a black bear, polar bear, fox, and rat obtained from the GenBank Database on the world-wide web; (ii) Four excerpts from the novel *The Zeppelin's Passenger* by E. Phillips Oppenheim, obtained from the Project Gutenberg Edition on the World-Wide web; (iii) Four MIDI files without further processing; two from Jimi Hendrix and two movements from Debussy's Suite Bergamasque, downloaded from various repositories on the world-wide web; (iv) Two Linux x86 ELF executables (the *cp* and *rm* commands), copied directly from the RedHat 9.0 Linux distribution; and (v) Two compiled Java class files, generated by ourselves. The compressor used to compute the NCD matrix was bzip2. As expected, the

program correctly classifies each of the different types of files together with like near like. The result is reported in Figure 4.8 with $S(T)$ equal to the very high confidence value 0.984. This experiment shows the power and universality of the method: no features of any specific domain of application are used. We believe that there is no other method known that can cluster data that is so heterogeneous this reliably. This is borne out by the massive experiments with the method in [54].

4.10 Testing on Natural Data

Like most hierarchical clustering methods for natural data, the quartet tree method has been developed in the biological setting to determine phylogeny trees from genomic data. In that setting, the data are (parts of) genomes of currently existing species, and the purpose is to reconstruct the evolutionary tree that led to those species. Thus, the species are labels of the leaves, and the tree is traditionally binary branching with each branching representing a split in lineages. The internal nodes and the root of the tree correspond with extinct species (possibly a still existing species in a leaf directly connected to the internal node). The case is roughly similar for the language tree reconstruction mentioned in the Introduction. The root of the tree is commonly determined by adding an object that is known to be less related to all other objects than the original objects are with respect to each other. Where the unrelated object joins the tree is where we put the root. In these settings, the direction from the root to the leaves represents an evolution in time, and the assumption is that there is a true tree we have to discover. However, we can also use the method for hierarchical clustering, resulting an unrooted ternary tree, and the assumption is not that there is a true tree we must discover. To the contrary, there is no true tree, but all we want is to model the similarity relations between the objects as well as possible, given the distance matrix. The interpretation is that objects in a given subtree are pairwise closer (more similar) to each other than any of those objects is with respect to any object in a disjoint subtree.

4.10.1 Analyzing the SARS and H5N1 Virus Genomes

As an application of our methods we clustered the SARS virus after its sequenced genome was made publicly available, in relation to potential similar virii. The 15 virus genomes were downloaded from The Universal Virus Database of the International Committee on Taxonomy of Viruses, available on the world-wide web. The SARS virus was downloaded from Canada's Michael Smith Genome Sciences Centre which had the first public SARS Coronavirus draft whole genome assembly available for download (SARS TOR2 draft genome assembly 120403). The NCD distance matrix was computed using the compressor bzip2. The relations in Figure 4.9 are very similar to the definitive tree based on medical-macrobio-genomics analysis, appearing later in the New England Journal of Medicine, [63]. We depicted the figure in the ternary tree style, rather than the genomics-dendrogram style, since the former is more precise for visual inspection of proximity relations.

More recently, we downloaded 100 different H5N1 sample genomes from the NCBI/NIH database online. We simply concatenated all data together directly, ignoring problems of data

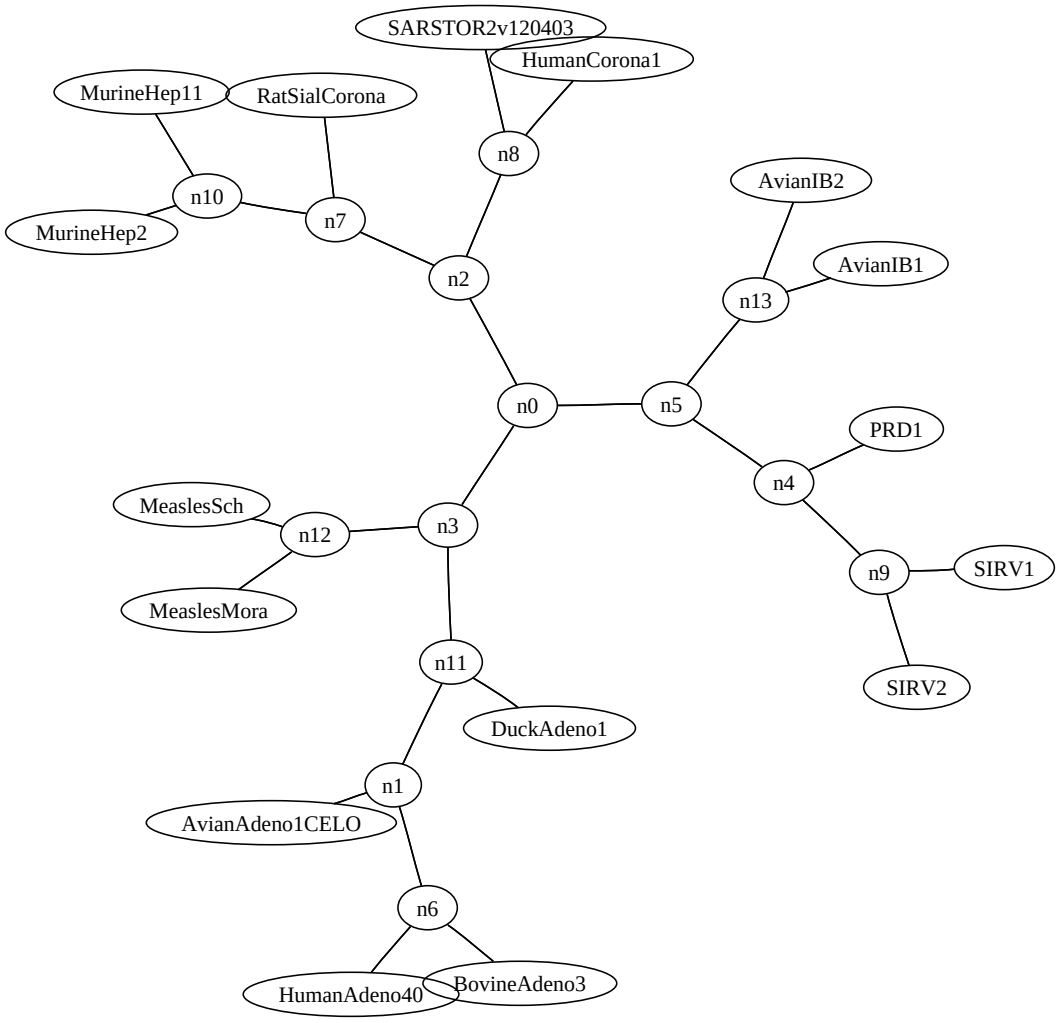

Figure 4.9: SARS virus among other virii. Legend: AvianAdeno1CELO.inp: Fowl adenovirus 1; AvianIB1.inp: Avian infectious bronchitis virus (strain Beaudette US); AvianIB2.inp: Avian infectious bronchitis virus (strain Beaudette CK); BovineAdeno3.inp: Bovine adenovirus 3; DuckAdeno1.inp: Duck adenovirus 1; HumanAdeno40.inp: Human adenovirus type 40; HumanCorona1.inp: Human coronavirus 229E; MeaslesMora.inp: Measles virus Moraten; MeaslesSch.inp: Measles virus strain Schwarz; MurineHep11.inp: Murine hepatitis virus strain ML-11; MurineHep2.inp: Murine hepatitis virus strain 2; PRD1.inp: Enterobacteria phage PRD1; RatSialCorona.inp: Rat sialodacryoadenitis coronavirus; SARS.inp: SARS TOR2v120403; SIRV1.inp: Sulfolobus SIRV-1; SIRV2.inp: Sulfolobus virus SIRV-2. $S(T) = 0.988$.

cleanup and duplication. We were warned in advance that certain coding regions in the viral genome were sometimes listed twice and also many sequences are incomplete or missing certain proteins. In this case we sought to test the robustness at the high end and at the same time verify, contextualize, and expand on the many claims of genetic similarity and diversity in the virology community. We used the CompLearn package, [21], with the *ppmd* compressor for this experiment and performed no alignment step whatsoever. We used order 15 with 250 megabytes memory maximum.

We have abbreviated Ck for Chicken and Dk for duck. Samples are named with species, location, sequence number, followed by the year double digits at the end. Naming is not 100% consistent. We can see the following features in Figure 4.10, that are possibly significant:

First, there is near-perfect temporal separation by branch and year, going all the way back to HongKong and GuangDong in 1997. Next, there is near-perfect regional separation with clear delineation of Japan and the crucial Qinghai, Astrakhan, Mongolia, and Novosibirsk, as well as near-perfect separation of Vietnam and Thailand. The placement CkVietnamC5804 and Vietnam306204 is interesting in that they are both near Thailand branches and suggest that they may be for example the migratory bird links that have been hypothesized or some other genetic intermediate. There is also throughout the tree substantial agreement with (and independent verification of) independent experts like Dr. Henry L. Niman [86] on every specific point regarding genetic similarity. The technique provides here an easy verification procedure without much work.

4.10.2 Music

The amount of digitized music available on the internet has grown dramatically in recent years, both in the public domain and on commercial sites. Napster and its clones are prime examples. Websites offering musical content in some form or other (MP3, MIDI, ...) need a way to organize their wealth of material; they need to somehow classify their files according to musical genres and subgenres, putting similar pieces together. The purpose of such organization is to enable users to navigate to pieces of music they already know and like, but also to give them advice and recommendations ("If you like this, you might also like..."). Currently, such organization is mostly done manually by humans, but some recent research has been looking into the possibilities of automating music classification. In [26, 25] we cluster music using the CompLearn Toolkit [21]. One example is a small set of classical piano sonatas, consisting of the 4 movements from Debussy's "Suite Bergamasque," 4 movements of book 2 of Bach's "Wohltemperierte Klavier," and 4 preludes from Chopin's "Opus 28." As one can see in Figure 4.11, our program does a pretty good job at clustering these pieces. The $S(T)$ score is also high: 0.968. The 4 Debussy movements form one cluster, as do the 4 Bach pieces. The only imperfection in the tree, judged by what one would intuitively expect, is that Chopin's Prélude no. 15 lies a bit closer to Bach than to the other 3 Chopin pieces. This Prélude no 15, in fact, consistently forms an odd-one-out in our other experiments as well. This is an example of pure data mining, since there is some musical truth to this, as no. 15 is perceived as by far the most eccentric among the 24 Préludes of Chopin's opus 28.

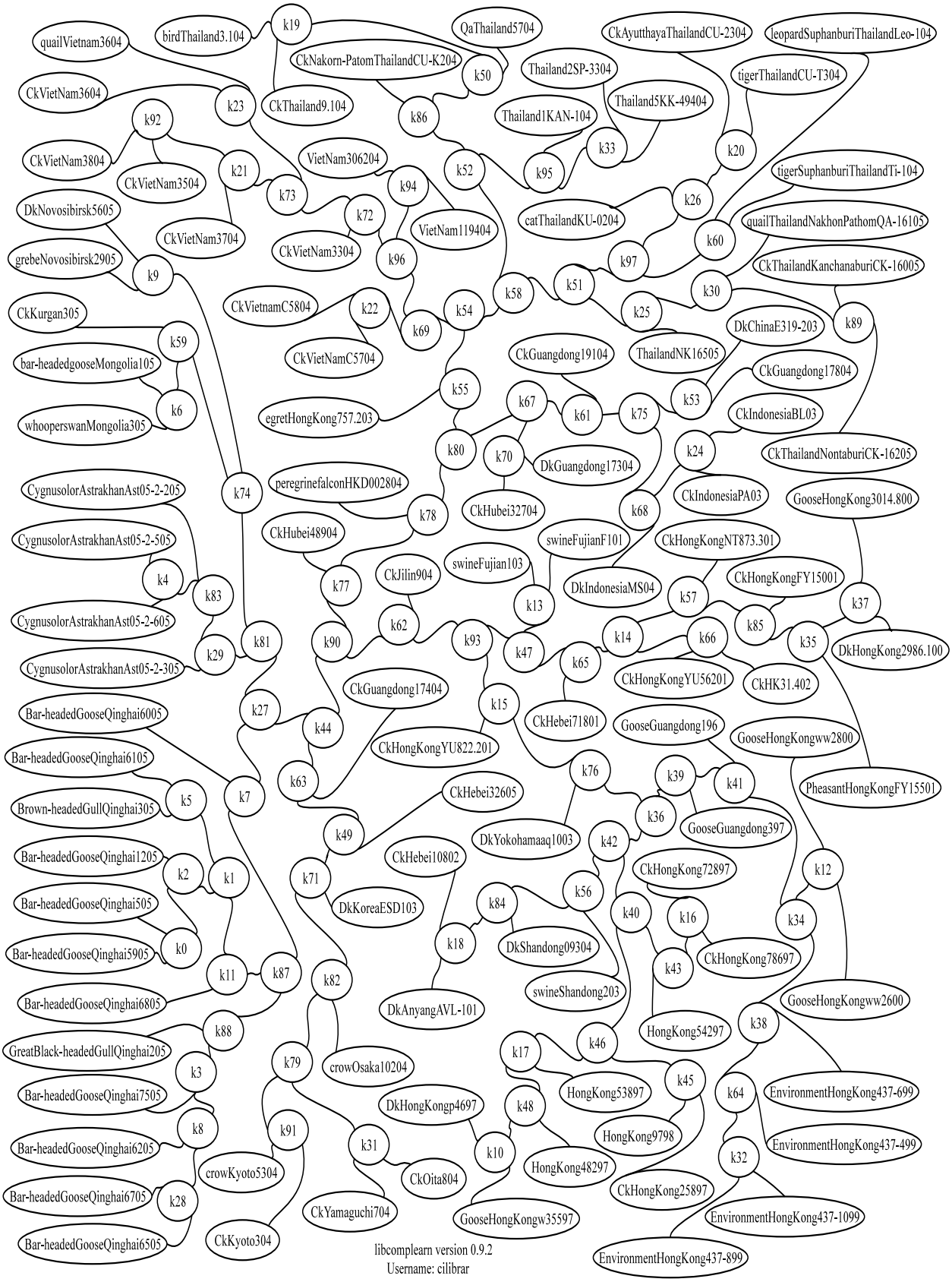

Figure 4.10: One hundred H5N1 (bird flu) sample genomes, S(T) = 0.980221.

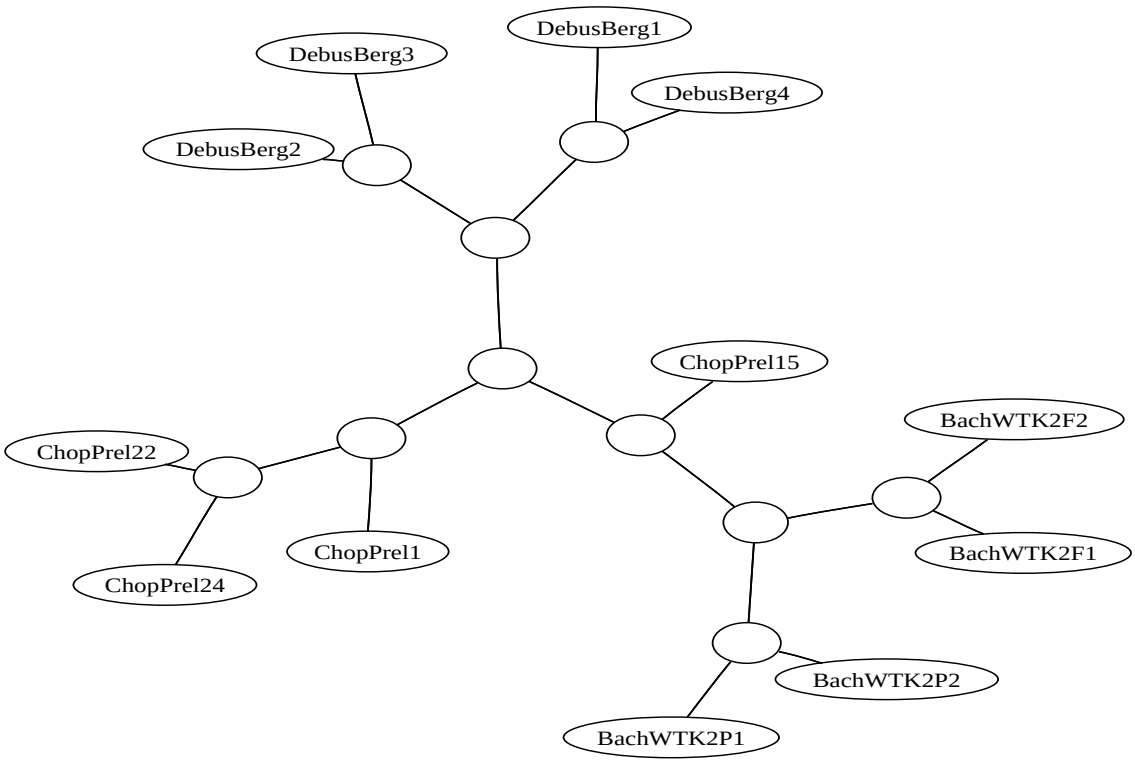

Figure 4.11: Output for the 12-piece set.

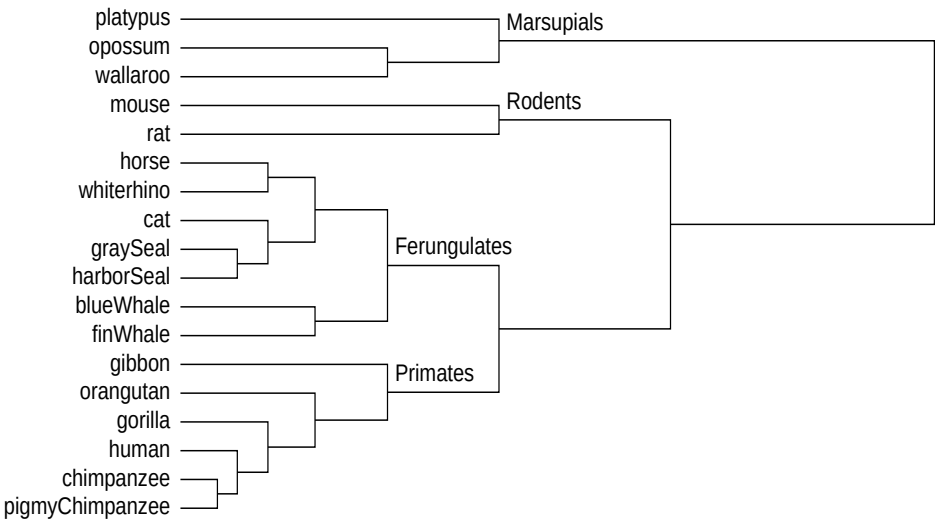

Figure 4.12: The evolutionary tree built from complete mammalian mtDNA sequences of 24 species, using the NCD matrix of Figure 4.14 on page 70 where it was used to illustrate a point of hierarchical clustering versus flat clustering. We have redrawn the tree from our output to agree better with the customary phylogeny tree format. The tree agrees exceptionally well with the NCD distance matrix: $S(T) = 0.996$.

4.10.3 Mammalian Evolution

As the complete genomes of various species become available, it has become possible to do whole genome phylogeny (this overcomes the problem that using different targeted parts of the genome, or proteins, may give different trees [94]). Traditional phylogenetic methods on individual genes depended on multiple alignment of the related proteins and on the model of evolution of individual amino acids. Neither of these is practically applicable to the genome level. In absence of such models, a method which can compute the shared information between two sequences is useful because biological sequences encode information, and the occurrence of evolutionary events (such as insertions, deletions, point mutations, rearrangements, and inversions) separating two sequences sharing a common ancestor will result in the loss of their shared information. Our method (in the form of the CompLearn Toolkit) is a fully automated software tool based on such a distance to compare two genomes. In evolutionary biology the timing and origin of the major extant placental clades (groups of organisms that have evolved from a common ancestor) continues to fuel debate and research.

The full experiment on mammalian evolution is discussed in Section 4.10.3. Here we just want to point out issues relevant for hierarchical clustering versus nonhierarchical clustering, and to our quartet tree method. We demonstrate that a whole mitochondrial genome phylogeny of the Eutherians (placental mammals) can be reconstructed automatically from a set of *unaligned* complete mitochondrial genomes by use of our compression method.

The whole mitochondrial genomes of the total of 24 species we used were downloaded from the GenBank Database on the world-wide web. Each is around 17,000 bases. The NCD dis-

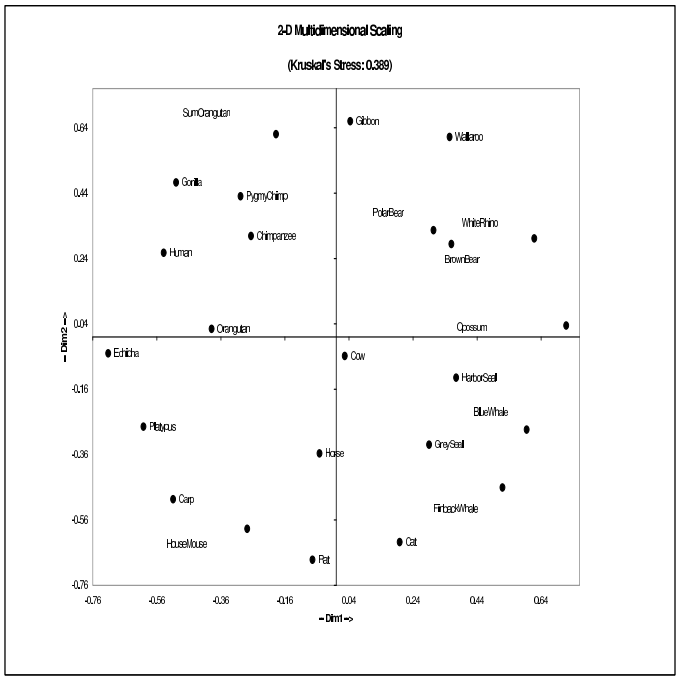

Figure 4.13: Multidimensional clustering of same NCD matrix (Figure 4.14) as used for Figure 6.7. Kruskall's stress-1 = 0.389.

tance matrix was computed using the compressor PPMZ. The resulting phylogeny, with an almost maximal $S(T)$ score of 0.996 supports anew the currently accepted grouping (Rodents, (Primates, Ferungulates)) of the Eutherian orders, and additionally the Marsupionta hypothesis ((Prototheria, Metatheria), Eutheria), see Figure 4.12. The NCD distance matrix is given in Figure 4.14, so the reader can get a feeling on what distances the quartet tree is based. For more explanation and details see Section 4.10.3.

4.11 Hierarchical versus Flat Clustering

This is a good place to contrast the informativeness of hierarchical clustering with multidimensional clustering using the same NCD matrix, exhibited in Figure 4.14. The entries give a good example of typical NCD values; we truncated the number of decimals from 15 to 3 significant digits to save space. Note that the majority of distances bunches in the range $[0.9, 1]$. This is due to the regularities the compressor can perceive. The diagonal elements give the self-distance, which, for PPMZ, is not actually 0, but is off from 0 only in the third decimal. In Figure 4.13 we clustered the 24 animals using the NCD matrix by multidimenional scaling as points in 2-dimensional Euclidean space. In this method, the NCD matrix of 24 animals can be viewed as a set of distances between points in n-dimensional Euclidean space ($n \leq 24$), which we want to project into a 2-dimensional Euclidean space, trying to distort the distances between the pairs as little as possible. This is akin to the problem of projecting the surface of the earth globe on a two-dimensional map with minimal distance distortion. The main feature is the choice of the

measure of distortion to be minimized, [36]. Let the original set of distances be $d_1,\ldots,d_k$ and the projected distances be $d'_1,\ldots,d'_k$. In Figure 4.13 we used the distortion measure *Kruskall's stress-1*, [62], which minimizes $\sqrt{(\sum_{i\leq k}(d_i-d'_i)^2)/\sum_{i\leq k}d_i^2}$. Kruskall's stress-1 equal 0 means no distortion, and the worst value is at most 1 (unless you have a really bad projection). In the projection of the NCD matrix according to our quartet method one minimizes the more subtle distortion $S(T)$ measure, where 1 means perfect representation of the relative relations between every 4-tuple, and 0 means minimal representation. Therefore, we should compare distortion Kruskall stress-1 with $1-S(T)$. Figure 4.12 has a very good $1-S(T)=0.04$ and Figure 4.13 has a poor Kruskall stress 0.389. Assuming that the comparison is significant for small values (close to perfect projection), we find that the multidimensional scaling of this experiment's NCD matrix is formally inferior to that of the quartet tree. This conclusion formally justifies the impression conveyed by the figures on visual inspection.

	BlueWhale	BrownBear	Carp	Cat	Chimpanzee	Cow	Echidna	FinWhale	Gibbon	Gorilla	GreySeal	HarborSeal	Horse	HouseMouse	Human	Opossum	Orangutan	Platypus	PolarBear	PygmyChimp	Rat	SumOrang	Wallaroo	WhiteRhino
BlueWhale	0.005	0.906	0.943	0.897	0.925	0.883	0.936	0.616	0.928	0.931	0.901	0.898	0.896	0.926	0.920	0.936	0.928	0.929	0.907	0.930	0.927	0.929	0.925	0.902
BrownBear	0.906	0.002	0.943	0.887	0.935	0.906	0.944	0.915	0.939	0.940	0.875	0.872	0.910	0.934	0.930	0.936	0.938	0.937	0.269	0.940	0.935	0.936	0.923	0.915
Carp	0.943	0.943	0.006	0.946	0.954	0.947	0.955	0.952	0.951	0.957	0.949	0.950	0.952	0.956	0.946	0.956	0.953	0.954	0.945	0.960	0.950	0.953	0.942	0.960
Cat	0.897	0.887	0.946	0.003	0.926	0.897	0.942	0.905	0.928	0.931	0.870	0.872	0.885	0.919	0.922	0.933	0.932	0.931	0.885	0.929	0.920	0.934	0.919	0.897
Chimpanzee	0.925	0.935	0.954	0.926	0.006	0.926	0.948	0.926	0.849	0.731	0.925	0.922	0.921	0.943	0.667	0.943	0.841	0.946	0.931	0.441	0.933	0.835	0.934	0.930
Cow	0.883	0.906	0.947	0.897	0.926	0.006	0.936	0.885	0.931	0.927	0.890	0.888	0.893	0.925	0.920	0.931	0.930	0.929	0.905	0.931	0.921	0.930	0.923	0.899
Echidna	0.936	0.944	0.955	0.942	0.948	0.936	0.005	0.936	0.947	0.947	0.940	0.937	0.942	0.941	0.939	0.936	0.947	0.855	0.935	0.949	0.941	0.947	0.929	0.948
FinbackWhale	0.616	0.915	0.952	0.905	0.926	0.885	0.936	0.005	0.930	0.931	0.911	0.908	0.901	0.933	0.922	0.936	0.933	0.934	0.910	0.932	0.928	0.932	0.927	0.902
Gibbon	0.928	0.939	0.951	0.928	0.849	0.931	0.947	0.930	0.005	0.859	0.932	0.930	0.927	0.948	0.844	0.951	0.872	0.952	0.936	0.854	0.939	0.868	0.933	0.929
Gorilla	0.931	0.940	0.957	0.931	0.731	0.927	0.947	0.931	0.859	0.006	0.927	0.929	0.924	0.944	0.737	0.944	0.835	0.943	0.928	0.732	0.938	0.836	0.934	0.929
GreySeal	0.901	0.875	0.949	0.870	0.925	0.890	0.940	0.911	0.932	0.927	0.003	0.399	0.888	0.924	0.922	0.933	0.931	0.936	0.863	0.929	0.922	0.930	0.920	0.898
HarborSeal	0.898	0.872	0.950	0.872	0.922	0.888	0.937	0.908	0.930	0.929	0.399	0.004	0.888	0.922	0.922	0.933	0.932	0.937	0.860	0.930	0.922	0.928	0.919	0.900
Horse	0.896	0.910	0.952	0.885	0.921	0.893	0.942	0.901	0.927	0.924	0.888	0.888	0.003	0.928	0.913	0.937	0.923	0.936	0.903	0.923	0.912	0.924	0.924	0.848
HouseMouse	0.926	0.934	0.956	0.919	0.943	0.925	0.941	0.933	0.948	0.944	0.924	0.922	0.928	0.006	0.932	0.923	0.944	0.930	0.924	0.942	0.860	0.945	0.921	0.928
Human	0.920	0.930	0.946	0.922	0.667	0.920	0.939	0.922	0.844	0.737	0.922	0.922	0.913	0.932	0.005	0.949	0.834	0.949	0.931	0.681	0.938	0.826	0.934	0.929
Opossum	0.936	0.936	0.956	0.933	0.943	0.931	0.936	0.936	0.951	0.944	0.933	0.933	0.937	0.923	0.949	0.006	0.960	0.938	0.939	0.954	0.941	0.960	0.891	0.952
Orangutan	0.928	0.938	0.953	0.932	0.841	0.930	0.947	0.933	0.872	0.835	0.931	0.932	0.923	0.944	0.834	0.960	0.006	0.954	0.933	0.843	0.943	0.585	0.945	0.934
Platypus	0.929	0.937	0.954	0.931	0.946	0.929	0.855	0.934	0.952	0.943	0.936	0.937	0.936	0.930	0.949	0.938	0.954	0.003	0.932	0.948	0.937	0.949	0.920	0.948
PolarBear	0.907	0.269	0.945	0.885	0.931	0.905	0.935	0.910	0.936	0.928	0.863	0.860	0.903	0.924	0.931	0.939	0.933	0.932	0.002	0.942	0.940	0.936	0.927	0.917
PygmyChimp	0.930	0.940	0.960	0.929	0.441	0.931	0.949	0.932	0.854	0.732	0.929	0.930	0.923	0.942	0.681	0.954	0.843	0.948	0.942	0.007	0.935	0.838	0.931	0.929
Rat	0.927	0.935	0.950	0.920	0.933	0.921	0.941	0.928	0.939	0.938	0.922	0.922	0.912	0.860	0.938	0.941	0.943	0.937	0.940	0.935	0.006	0.939	0.922	0.922
SumOrangutan	0.929	0.936	0.953	0.934	0.835	0.930	0.947	0.932	0.868	0.836	0.930	0.928	0.924	0.945	0.826	0.960	0.585	0.949	0.936	0.838	0.939	0.007	0.942	0.937
Wallaroo	0.925	0.923	0.942	0.919	0.934	0.923	0.929	0.927	0.933	0.934	0.920	0.919	0.924	0.921	0.934	0.891	0.945	0.920	0.927	0.931	0.922	0.942	0.005	0.935
WhiteRhino	0.902	0.915	0.960	0.897	0.930	0.899	0.948	0.902	0.929	0.929	0.898	0.900	0.848	0.928	0.929	0.952	0.934	0.948	0.917	0.929	0.922	0.937	0.935	0.002

Figure 4.14: Distance matrix of pairwise NCD. For display purpose, we have truncated the original entries from 15 decimals to 3 decimals precision.

Chapter 5

Classification systems using NCD

This chapter explores the concept of *classification*. In rough terms, classification refers to the placement of unknown *test objects* into one of several categories based on a training set of objects. It is different from the hierarchical clustering problem that has been the primary focus in this thesis up to this point, yet it is no less fundamental. Combining NCD with a trainable machine learning module yields wonderfully effective and often surprising results, showing that in certain situations, we may in essence *learn by example* with the help of human experts. In Section 5.1 below, classification is first addressed from a general perspective. In Section 5.2, it is shown how to combine NCD in combination with trainable classifiers based on so-called *anchors*. Section 5.3 discusses two options for such trainable classifiers: neural networks and support vector machines.

5.1 Basic Classification

The classification problem setting as considered in this thesis is given as follows. A human expert has prepared n training examples. Each training example consists of a d-dimensional input vector $\mathbf{x}$ and a target training label y. y must be chosen from a discrete set L of labels. A human expert supplies this information; the accuracy of the trained system would of course be limited by the accuracy of the labels in the training set. A *training session* computes a model M from the input or *training* data $(\mathbf{x}_i, y_i)$ for $1 \leq i \leq n$. The goal of a classification algorithm is to make good predictions based on the input training data. After M has been calculated by some *classifier learning algorithm*, henceforth called *trainable classifier system*, it is used to classify unknown test objects $\mathbf{x}_{\text{test}}$, also of dimension d. It will output an element from L for each such object. Note that M can be thought of as a *function* that maps test input vectors $\mathbf{x}$ to labels in the set L. We refer to such a function as a *classifier*. Learning good classifiers is considered one of the most important problems in machine learning and pattern recognition in current times. One of the most important early successes has been in the field of optical character recognition, or OCR. This is the problem of taking a rasterized bitmap image and converting it to a sequence of ASCII characters according to some symbol recognition algorithm. Typical OCR software can work on as little as ten classes for applications such as numerical digit recognition,

or many dozens of characters for scanning full text. They can operate in a highly specific way, for instance recognizing only one text font of a particular size, or they can be more generic, for instance learning handwritten letters and digits. In this situation, a typical setup would be to first use some preprocessing to try to split the large image into smaller images each containing just one symbol (or glyph). Then pad these boxes so that they are all squares of the same size and the glyph is roughly centered. Next each pixel may be read off in order and converted to a successive dimension in $\mathbf{x} = (x_1, \ldots, x_d)$: pixel i corresponds to dimension x_i. For each pixel, a background color would be represented by 0 and a foreground color by 1. Thus each $\mathbf{x}$ input would be a binary vector with dimension equal to the number of pixels in the boxes surrounding each symbol. The output from such a classification system would be a single character from the range of possible characters under consideration. In this context, a learning algorithm would be given as input a training sequence $((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, and then output a "learned" classifier M. This learned classifier M could then be used, for any new example $\mathbf{x}$ (pixel vector representing a character), to make a prediction of the corresponding class y (the actual character). The situation just described corresponds to a typical setup, however in the following experiments we take a somewhat different approach using NCD.

5.1.1 Binary and Multiclass Classifiers

The simplest kind of classifier is called the binary classifier. This type of classifier can only produce two labels as output for each test case. The labels are usually written $+1$ and -1 or 0 and 1 depending on the problem. A common example of binary classification is the spam-filtering problem. This problem is to determine if a given email message is an unwanted commercial advertisement (spam) or not automatically before being brought to the attention of an email user. Another important kind of classifier is the multiclass classifier. This type of classifier applies more than two different types of labels. This is useful in cases like handwritten digit recognition, where there are at least ten different labels for handwritten digits that must sometimes be output.

It is usually simpler mathematically to consider only the binary classification case. This is justified by the fact that there are two well known ways to create a multiclass classifier using several binary classifiers acting cooperatively together. The two traditional ways of doing this are called *one-of-k* style combination and *pairwise* combination. The one-of-k style is simple to explain. A classifier is trained, one per class, for each of the k classes in the multiclass classification problem. Each classifier is trained to distinguish only members of its own class and reject (return 0) members of any other class.

In contrast, pairwise combination trains a separate binary classifier to distinguish each unique unordered pair of classes under consideration. Then a voting scheme is used to determine the winning class by running through all classifiers for each input sample. This results in $O(k^2)$ classifiers for k classes.

The one-of-k style combination yields the worst accuracy typically but is the simplest and fastest. The pairwise combination is usually the most accurate.

5.1.2 Naive NCD Classification

The simplest (and undoubtedly popular) way to use NCD to classify is to choose, for each of k classes, a single *prototype object* of the class that somehow captures the essence of the category. So, for example, if the task were to distinguish English and Chinese, we might consider using an English dictionary as the prototype object for the English class, and a Chinese dictionary for the Chinese class. In each case, we simplify a class down to a single example. Then the classification is done by calculating the NCD of the test object with each of the k prototype objects, and selecting the class corresponding to the object with the minimum NCD value. This approach seems intuitively obvious and is usually the first method people new to NCD invent. In some domains it works well, but in many more subtle classification problems it suffers a problem of uncorrectable bias. This relates to the fact that the different classes of most problems (such as the character classes in OCR) do not usually balance well under any particular available compressors. For example, the pixelated character class of the numeral "1" is sufficiently bland as to have a high NCD when compared to most other scribblings, even other members of the "1" class. This contrasts with the numeral "8" which has a rich combination of shapes that tends to compress well with most other outline images due to the wide variety of possible matches. This type of bias leads to a large constant error margin that cannot readily be improved within this framework as there are no natural adjustments available. In the next section, we explore a solution to this problem.

5.2 NCD With Trainable Classifiers

The simplest solution to the problem of how to use NCD for high accuracy classification is to combine it with a trainable classifier system by using NCD as a feature extraction technique. A trainable classifier system tries to pick up functional relationships in the input and output quantities. While trainable classifier systems output functions with a discrete range (the set of classes), some of the most successful ones, such as neural network– and SVM– based algorithms, are built on top of continuous learning algorithms. The continuous learners are a broad and important class of algorithms in machine learning. Given a training sequence $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, they learn/output a continuous function M mapping d-dimensional input vectors to the one-dimensional reals. Such learners can be transformed into learning algorithms for binary classification, by classifying test vector $\mathbf{x}$ as 1 if $M(\mathbf{x}) > 0$, and $\mathbf{x}$ as -1 if $M(\mathbf{x}) < 0$.

In order to apply this idea, one must set up a problem so that unknown objects are somehow converted to fixed-dimensional vectors using some sort of projection using NCD. One of the easiest techniques is to designate some d objects as *anchors* and to use them to convert all other objects in the experiment into d-dimensional vectors. This can be achieved by using anchor object a_i to calculate vector dimension i for $1 \leq i \leq d$. That is, for object o we calculate the corresponding $\mathbf{x} = (x_1, \ldots, x_d)$ using $x_i = NCD(o, a_i)$.

For example, in handwritten digit recognition, our training data may originally consist of pairs $((o_1, y_1) \ldots, (o_n, y_n))$, where o_i represents an image (represented with one character per pixel, and one character as a line terminator) of a handwritten digit and $y_i \in \{0, \ldots, 9\}$ is the

corresponding digit. One then takes 80 images $o'_1, \ldots, o'_{80}$, so that each handwritten digit is represented eight times. The training data $(o_1, \ldots, o_n)$ are then converted to $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, where each $\mathbf{x}_i$ is the 80-dimensional vector $(NCD(o_i, o'_1), \ldots, NCD(o_i, o'_{80}))$.

5.2.1 Choosing Anchors

The anchors may be chosen by a human expert or be chosen randomly from a large pool of training objects. Intuitively, it may seem very important which objects are chosen as anchors. While this is sometimes the case, more often it is not. Random anchors usually work well. But most practitioners agree that picking at least one anchor from each significant category under consideration is advisable to avoid falling into low-accuracy performance due to insufficient variation in the anchor set. The anchor can be used with NCD. It can also be used with the Normalized Google Distance (NGD), which we discuss in Chapter 7. We will give details using NGD terms in Section 7.6.4, and details using Optical Character Recognition with NCD in Section 6.8.

In choosing a learning system to use with the anchor features extracted above, it seems best to choose as general a learning system as possible. One particularly important class of learning algorithms are called the universal continuous function learners. These include neural networks and Support Vector Machines. The universal learning property in this context means that they can approximate any continuous function to an arbitrarily high degree of accuracy given enough training data in the neighborhood of the point under consideration. Using a learner with this property ensures a certain degree of reliability and optimality to the results that it generates as compared to other more specialized learning algorithms.

5.3 Trainable Learners of Note

There are at least two good choices for trainable learner components for use with NCD. These are called the *neural network* and the *Support Vector Machine*. Both of these techniques take as input the set of labeled training data as well as some specialized model parameters that must be set through some means. Usually the specialized parameters are set by a human expert or set through an automatic procedure using cross-validation based parameter scanning [11].

Both learner systems produce a single label out of the set L as a result given any input d-dimensional test vector. Each have many strengths and weaknesses, and they each give unique performance profiles. They should both be considered when deciding to do classification using NCD, although all experiments in this thesis are based on support vector machines.

5.3.1 Neural Networks

The older and considerably popular choice is artificial neural networks [46]. As mentioned previously, these have the advantage of being universal classifier systems: they have provable learning capability for any continuous function. Additionally, they have a host of decent training algorithms as well as learning modes. Popular choices here are Fast Backpropagation and Self Organizing Maps. SOM's have not been designed explicitly for classification, but can easily

adapted to be used for classification tasks when used in combination with compression metrics like NCD.

There is one major difficulty in all types of neural network investigated; they have some rather tricky parameter settings. A neural network of any sophistication must have a nonlinear component (transfer function) of some sort, and common choices are transcendental sigmoid functions like arctangent. There are several others in use. Another hard issue with many types of neural networks is to decide how many layers should be used. For each layer a specific number of neurons must be set in advance as well. Similarly, there is usually a learning rate and sometimes a momentum parameter that can radically affect the network's gross behavior and overall success or failure. Altogether this implies at least four hard choices before one can train a neural network. There is no simple rule to guide how to choose these parameters because there are all sorts of bad behavior possible from wrong choices. The most common are overlearning and underlearning. Informally, overlearning occurs when there are too many neurons for the task at hand. Underlearning is when there are too few. In the case of overlearning, the neural network will appear to have great accuracy in the training data yet terrible accuracy in the testing phase and show almost no generalization capability. In the underlearning case, accuracy will simply be capped at a number far worse than what may otherwise be achieved. It's quite difficult to tell what the best parameters are for a neural network and in real commercial systems there is usually a considerable amount of sophisticated semi-automatic machinery in place to assist the user in setting these parameters. For example, genetic algorithms are sometimes employed combined with cross-validation to adjust parameters in a semi-automatic fashion. This heavy emphasis on parameter-setting in advance makes neural networks a difficult component to integrate into an easy-to-use parameter-free learning system.

5.3.2 Support Vector Machines

More recently another type of continuous universal learner has come into great popularity. They have the same sort of universal properties as artificial neural networks but with substantially less hassle [11]. They are called support vector machines. They take advantage of dot products in a high dimensional space to efficiently find solutions to a convex optimization problem that winds up bending a decision surface around training points in the least stressful way. In order to use an SVM, one needs to first choose a kernel function and this corresponds roughly to the transfer function in neural networks. Although a linear kernel is an option and polynomials are sometimes used, they do not yield the same (typically desirable) infinite-dimensional properties that the exponential Radial Basis Function kernels do. These are described further below.

For all of our experiments we have found RBF kernels to be about as good as any other choice and often substantially better. There are only two other parameters for SVM's. In our context, these are called C and g. C is the cost for wrong answers, or training points that fall on the wrong side of the decision boundary. Adjusting this can compensate for noisy or mislabeled training data. The g parameter represents kernel width and determines the rate of exponential decay around each of the training points. The very wonderful property of both of these parameters is that they can be determined through a simple two dimensional grid search.

This procedure was found to be much simpler to automate and far more robust than neural

networks. Therefore, most of our experiments have focused on SVM as the trainable learning component. In the next section technical details are presented regarding SVM's.

5.3.1. REMARK. It is important to realize that we still have not exhausted the possibilities for combining NCD with other machine learning algorithms; there are in fact many interesting options. For example, Gaussian processes could also serve as a good next phase learner but have not yet been investigated. Similarly, the previously mentioned quartet-tree search is another option at this same stage in the pipeline, as is multidimensional scaling, nearest neighbor search and most other classical machine learning algorithms. In this light we may consider NCD as a particularly convenient and sometimes highly intelligent feature-extraction (or dimensionality reduction) technique suitable for drop-in replacement in a number of larger automatic learning systems.

5.3.2. REMARK. Many classification systems, including both of those mentioned above, are able to be used in a special mode called *regression mode*. In this mode, the output is not a member of the set of labels L but instead is a real (scalar) value, typically between 0 and 1 or between -1 and 1. This mode allows prediction of continuous variables such as temperature or duration. It is again a fundamental problem in machine learning and comes up often and provides yet more reason to use one of the two universal learners mentioned above with NCD.

5.3.3 SVM Theory

This section provides a brief glimpse at relevant mathematical theory surrounding Support Vector Machines. For more information please see [16]. Support Vector Machines represent a way to learn a classification or regression problem by example, and are comparable to neural networks in that they have the capacity to learn any function. They are *large-margin* classifiers [11]. They take as input a list of k-dimensional vectors, and output a single scalar value. In order to learn, an SVM solves a convex optimization problem that is closely related to a simpler classification engine termed the separating hyperplane. In this setting, we are given a set of k-dimensional training vectors $\mathbf{x}_i$ each labeled y_i which is 1 or -1 according to the classification. For a particular problem, a discriminating hyperplane $\mathbf{w}$ is one that creates a decision function that satisfies the constraint for all i:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0.$$

If no such separating hyperplane exists, then we term the learning problem *linearly inseparable*. The constants used in this equation are more fully explained in [16]. In rough terms, the equation represents a linear programming problem that tries to find a simple (fewer support vectors chosen) and accurate (less disagreements with training data) model using a convex optimization technique.

Many real-world learning problems, such as the famous exclusive-or function, are linearly inseparable. This is problematic because a hyperplane can only separate spaces into linearly separable components. There are many strategies for dealing with this issue. Support vector machines use a nonlinear kernel function to address this issue. By creating a kernel function, $k(x,y)$

that satisfies the *Mercer condition*, we may substantially enhance the power of the separating hyperplane. A kernel function defines an inner-product on the input space, and then this inner product may be used to calculate many higher-power terms of combinations of samples in the input space. This forms a higher-dimensional space, and it is well-known that once this space is made large enough, there will be a separating hyperplane. In our SVM experiments, we use a Radial Basis Function (RBF) kernel. This allows the SVM to learn any function given enough training data.

5.3.4 SVM Parameter Setting

There are two parameters that control the learning of the SVM. The first relates to the kernel function. An RBF, or Radial Basis Function, kernel assumes the value 1 whenever the two input vectors are equal. If they are unequal, it decays slowly towards 0 in a radially symmetric way:

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2 / 2g^2}.$$

Here, g is a parameter that controls the rate of decay or width of the kernel function. Because of the exponential form, the effective dimension of an RBF kernel is potentially infinite and thus this kernel can be used to approximate any continuous function to an arbitrary degree of accuracy. This parameter must be set before the learning can begin with an SVM. Another parameter relates to how misclassified points are handled in the training data; Though it is always possible to simply continue to make the kernel width smaller and the expanded space larger until the SVM becomes essentially a lookup-table, this is often not the best strategy for learning. An alternative is to define a cost parameter and allow this to adjust the tolerance for misclassified points in the training data. This allows the SVM to generalize well even in the presence of noisy data. This cost parameter, often called c, must also be defined before training can begin.

We select g and c using a grid searching technique. For each of these parameters, it is appropriate to search dozens of powers of two. Together, this creates a grid with hundreds of different parameter settings. We use five-fold cross-validation to select which of these grid points defines the optimal parameter setting. First the data is divided into five random partitions: A, B, C, D, E. Then, for each candidate parameter setting or grid point, we run five different training runs. On the first run, we train on B, C, D, and E, and then we determine an accuracy using part A. Next, we train on A, C, D, E and test with B. We continue in this way and then average all five test scores to arrive at an estimate of how well the learning process performed. Once this process is done for all training data, we may just choose one of the grid points that attains a maximum accuracy.

These parameters (C and g) do not usually need to be exact; instead one can simply do a stepwise search along a grid of points in log space; thus it is fine to try just 64 points for C ranging from 2^{-31} to 2^{32} doubling each time. A similar procedure works to choose g. In each case one may use five-fold cross-validation to estimate the accuracy of the given parameter setting and then just choose the maximum at the end. Five-fold cross validation is a popular procedure that tries to estimate how well a model (of any type) that is trained on a set of training data will perform on unknown testing data. The basic assumption is simply that the training data and the

testing data are similar. In order to estimate the testing accuracy using only the training algorithm and the training data, first label the training set randomly with five labels, numbered 1 through 5. Next, train a model (SVM, neural network, or otherwise) on four of the five parts, but leave part 1 out. Use this part 1 for testing the model made from the other 4 parts. Tally this score and repeat the procedure, but this time withhold part 2 for testing. Repeat this procedure five times, once for each part, and then average the accuracy scores to arrive at an estimate of the testing accuracy to be expected for a given set of parameters. This entire five-fold cross-validation procedure is repeated for each point (particular parameter setting) in the parameter grid space in order to find the optimal settings using just the training data.

Chapter 6

Experiments with NCD

This chapter demonstrates the surprisingly general and robust nature of the methods so far discussed through many real examples from diverse areas such as music and evolution. The combination of NCD and the quartet method yield interesting results.

In Section 6.1, we introduce the general concept of feature-based similarity, and explain how NCD can be used with it. In Section 6.2 we present experimental validation that our method works. Starting in Section 6.4, we introduce a plethora of experiments in a wide array of fields, beginning with automatic music analysis. Next, we study evolutionary genomics, literature and language analysis, radio astronomy, and optical character recognition. At the end of this chapter the reader will have encountered a highly serviceable survey of experimental results using objective data compressors based on files, without external information input from the internet.

6.1 Similarity

We are presented with unknown data and the question is to determine the similarities among them and group like with like together. Commonly, the data are of a certain type: music files, transaction records of ATM machines, credit card applications, genomic data. In these data there are hidden relations that we would like to get out in the open. For example, from genomic data one can extract letter- or block frequencies (the blocks are over the four-letter alphabet); from music files one can extract various specific numerical features, related to pitch, rhythm, harmony etc. One can extract such features using for instance Fourier transforms [114] or wavelet transforms [44]. The feature vectors corresponding to the various files are then classified or clustered using existing classification software, based on various standard statistical pattern recognition classifiers [114], Bayesian classifiers [33], hidden Markov models [19], ensembles of nearest neighbor classifiers [44] or neural networks [33, 101]. For example, in music one feature would be to look for rhythm in the sense of beats per minute. One can make a histogram where each histogram bin corresponds to a particular tempo in beats-per-minute and the associated peak shows how frequent and strong that particular periodicity was over the entire piece. In [114] we see a gradual change from a few high peaks to many low and spread-out ones going from hip-hip, rock, jazz, to classical. One can use this similarity type to try to cluster pieces in these categories.

However, such a method requires specific and detailed knowledge of the problem area, since one needs to know what features to look for.

Non-Feature Similarities: Our aim is to capture, in a single similarity metric, *every effective metric*: effective versions of Hamming distance, Euclidean distance, edit distances, alignment distance, Lempel-Ziv distance [30], and so on. This metric should be so general that it works in every domain: music, text, literature, programs, genomes, executables, natural language determination, equally and simultaneously. It would be able to simultaneously detect *all* similarities between pieces that other effective metrics can detect.

Compression-based Similarity: Such a "universal" metric was developed by Li and Vitanyi *et al.* [75, 77] as a normalized version of the "information metric" of [79, 9], see Chapter 3. Recall that two objects are deemed close if we can significantly "compress" one given the information in the other, the idea being that if two pieces are more similar, then we can more succinctly describe one given the other. Recall from Chapter 3 that an appropriately "normalized" information distance minorizes every metric in the class of effective similarity metrics. It discovers all effective similarities in the sense that if two objects are close according to some effective similarity, then they are also close according to the normalized information distance. Put differently, the normalized information distance represents similarity according to the dominating shared feature between the two objects being compared. The normalized information distance too is a metric and takes values in $[0,1]$; hence it may be called *"the" similarity metric*. To apply this ideal precise mathematical theory in real life, we have to replace the use of the uncomputable Kolmogorov complexity by an approximation using a standard real-world compressor. Approaches predating this thesis include the first completely automatic construction of the phylogeny tree based on whole mitochondrial genomes, [75, 80, 77], a completely automatic construction of a language tree for over 50 Euro-Asian languages [77], detects plagiarism in student programming assignments [74], gives phylogeny of chain letters [10], and clusters music [26, 25]. Moreover, the method turns out to be robust under change of the underlying compressor-types: statistical (PPMZ), Lempel-Ziv based dictionary (gzip), block based (bzip2), or special purpose (Gencompress).

Related Work: In view of the simplicity and naturalness of our proposal, it is perhaps surprising that compression based clustering and classification approaches did not arise before. But recently there have been several partially independent proposals in that direction: [8, 3] for building language trees—while citing [79, 9]—is by essentially more *ad hoc* arguments about empirical Shannon entropy and Kullback-Leibler distance. This approach is used to cluster music MIDI files by Kohonen maps in [34]. Another recent offshoot based on our work is [61] hierarchical clustering based on mutual information. In a related, but considerably simpler feature based approach, one can compare the word frequencies in text files to assess similarity. In [120] the word frequencies of words common to a pair of text files are used as entries in two vectors, and the similarity of the two files is based on the distance between those vectors. The authors attribute authorship to Shakespeare plays, the Federalist Papers, and the Chinese classic "The Dream of the Red Chamber." The approach to similarity distances based on block occurrence statistics is standard in genomics, and in an experiment below it gives inferior phylogeny trees compared to our compression method (and wrong ones according to current biological wisdom). The possibly new feature in the cited work is that it uses statistics of only the words that the files being

compared have in common. A related, opposite, approach was taken in [59], where literary texts are clustered by author gender or fact versus fiction, essentially by first identifying distinguishing features, like gender dependent word usage, and then classifying according to those features.

Apart from the experiments reported here, the clustering by compression method reported in this thesis has recently been used to analyze network traffic and cluster computer worms and viruses [118]. Finally, recent work [54] reports experiments with our method on all time sequence data used in all the major data-mining conferences in the last decade. Comparing the compression method with all major methods used in those conferences they established clear superiority of the compression method for clustering heterogenous data, and for anomaly detection.

To substantiate our claim of universality, we apply the method to different areas, not using any feature analysis at all. We first give an example in whole-genome phylogeny using the whole mitochondrial DNA of the species concerned. We compare the hierarchical clustering of our method with a more standard method of two-dimensional clustering (to show that our dendrogram method of depicting the clusters is more informative). We give a whole-genome phylogeny of fungi and compare this to results using alignment of selected proteins (alignment being often too costly to perform on the whole-mitochondrial genome, but the disadvantage of protein selection being that different selections usually result in different phylogenies—so which is right?). We identify the virii that are closest to the sequenced SARS virus; we give an example of clustering of language families; Russian authors in the original Russian, the same pieces in English translation (clustering partially follows the translators); clustering of music in MIDI format; clustering of handwritten digits used for optical character recognition; and clustering of radio observations of a mysterious astronomical object, a microquasar of extremely complex variability. In all these cases the method performs very well in the following sense: The method yields the phylogeny of 24 species precisely according to biological wisdom. The probability that it randomly would hit this one outcome, or anything reasonably close, is very small. In clustering 36 music pieces taken equally many from pop, jazz, classic, so that 12-12-12 is the grouping we understand is correct, we can identify convex clusters so that only six errors are made. (That is, if three items get dislodged then six items get misplaced.) The probability that this happens by chance is extremely small. The reason why we think the method does something remarkable is concisely put by Laplace [70]:

> "If we seek a cause wherever we perceive symmetry, it is not that we regard the symmetrical event as less possible than the others, but, since this event ought to be the effect of a regular cause or that of chance, the first of these suppositions is more probable than the second. On a table we see letters arranged in this order `C o n s t a n t i n o p l e`, and we judge that this arrangement is not the result of chance, not because it is less possible than others, for if this word were not employed in any language we would not suspect it came from any particular cause, but this word being in use among us, it is incomparably more probable that some person has thus arranged the aforesaid letters than that this arrangement is due to chance."

Materials and Methods: The data samples we used were obtained from standard data bases accessible on the world-wide web, generated by ourselves, or obtained from research groups in

the field of investigation. We supply the details with each experiment. The method of processing the data was the same in all experiments. First, we preprocessed the data samples to bring them in appropriate format: the genomic material over the four-letter alphabet $\{A,T,G,C\}$ is recoded in a four-letter alphabet; the music MIDI files are stripped of identifying information such as composer and name of the music piece. Then, in all cases the data samples were completely automatically processed by our CompLearn Toolkit, rather than as is usual in phylogeny, by using an eclectic set of software tools per experiment. Oblivious to the problem area concerned, simply using the distances according to the NCD below, the method described in this thesis fully automatically classifies the objects concerned. The CompLearn Toolkit is a suite of simple utilities that one can use to apply compression techniques to the process of discovering and learning patterns in completely different domains. In fact, this method is so general that it requires no background knowledge about any particular subject area. There are no domain-specific parameters to set, and only a handful of general settings.

The CompLearn Toolkit using NCD and not, say, alignment, can cope with full genomes and other large data files and thus comes up with a single distance matrix. The clustering heuristic generates a tree with a certain confidence, called standardized benefit score or $S(T)$ value in the sequel. Generating trees from the same distance matrix many times resulted in the same tree or almost the same tree, for all distance matrices we used, even though the heuristic is randomized. The differences that arose are apparently due to early or late termination with different $S(T)$ values. This is a great difference with previous phylogeny methods, where because of computational limitations one uses only parts of the genome, or certain proteins that are viewed as significant [55]. These are run through a tree reconstruction method like neighbor joining [99], maximum likelihood, maximum evolution, maximum parsimony as in [55], or quartet hypercleaning [13], many times. The percentage-wise agreement on certain branches arising are called "bootstrap values." Trees are depicted with the best bootstrap values on the branches that are viewed as supporting the theory tested. Different choices of proteins result in different best trees. One way to avoid this ambiguity is to use the full genome, [94, 77], leading to whole-genome phylogeny. With our method we do whole-genome phylogeny, and end up with a single overall best tree, not optimizing selected parts of it.

The quality of the results depends on (a) the NCD distance matrix, and (b) how well the hierarchical tree represents the information in the matrix. The quality of (b) is measured by the $S(T)$ value, and is given with each experiment. In general, the $S(T)$ value deteriorates for large sets. We believe this to be partially an artifact of a low-resolution NCD matrix due to limited compression power, and limited file size. The main reason, however, is the fact that with increasing size of a natural data set the projection of the information in the NCD matrix into a binary tree can get increasingly distorted as explained in Chapter 5, page 45. Another aspect limiting the quality of the NCD matrix is more subtle. Recall that the method knows nothing about any of the areas we apply it to. It determines the dominant feature as seen through the NCD filter. The dominant feature of likeness between two files may not correspond to our a priori conception but may have an unexpected cause. The results of our experiments suggest that this is not often the case: In the natural data sets where we have preconceptions of the outcome, for example that works by the same authors should cluster together, or music pieces by the same composers, musical genres, or genomes, the outcomes conform largely to our expectations.

For example, in the music genre experiment the method would fail dramatically if genres were evenly mixed, or mixed with little bias. However, to the contrary, the separation in clusters is almost perfect. The few misplacements that are discernible are either errors (the method was not powerful enough to discern the dominant feature), or the dominant feature between a pair of music pieces is not the genre but some other aspect. The surprising news is that we can generally confirm expectations with few misplacements, indeed, that the data does not contain unknown rogue features that dominate to cause spurious (in our preconceived idea) clustering. This gives evidence that where the preconception is in doubt, like with phylogeny hypotheses, the clustering can give true support of one hypothesis against another one.

Figures: We use two styles to display the hierarchical clusters. In the case of genomics of Eutherian orders and fungi, language trees, it is convenient to follow the dendrograms that are customary in that area (suggesting temporal evolution) for easy comparison with the literature. Although there is no temporal relation intended, the dendrogram representation looked also appropriate for the Russian writers, and translations of Russian writers. In the other experiments (even the genomic SARS experiment) it is more informative to display an unrooted ternary tree (or binary tree if we think about incoming and outgoing edges) with explicit internal nodes. This facilitates identification of clusters in terms of subtrees rooted at internal nodes or contiguous sets of subtrees rooted at branches of internal nodes.

Testing the similarity machine on natural data: We test gross classification of files based on markedly different file types. Here, we chose several files: (i) Four mitochondrial gene sequences, from a black bear, polar bear, fox, and rat obtained from the GenBank Database on the world-wide web; (ii) Four excerpts from the novel *The Zeppelin's Passenger* by E. Phillips Oppenheim, obtained from the Project Gutenberg Edition on the World Wide web; (iii) Four MIDI files without further processing; two from Jimi Hendrix and two movements from Debussy's Suite Bergamasque, downloaded from various repositories on the world-wide web; (iv) Two Linux x86 ELF executables (the *cp* and *rm* commands), copied directly from the RedHat 9.0 Linux distribution; and (v) Two compiled Java class files, generated by ourselves. The compressor used to compute the NCD matrix was bzip2. As expected, the program correctly classifies each of the different types of files together with like near like. The result is reported in Figure 6.1 with $S(T)$ equal to the very high confidence value 0.984. This experiment shows the power and universality of the method: no features of any specific domain of application are used.

6.2 Experimental Validation

We developed the CompLearn Toolkit, and performed experiments in vastly different application fields to test the quality and universality of the method. The success of the method as reported below depends strongly on the judicious use of encoding of the objects compared. Here one should use common sense on what a real world compressor can do. There are situations where our approach fails if applied in a straightforward way. For example: comparing text files by the same authors in different encodings (say, Unicode and 8-bit version) is bound to fail. For the ideal similarity metric based on Kolmogorov complexity as defined in [77] this does not matter at all, but for practical compressors used in the experiments it will be fatal. Similarly, in the music

experiments below we use symbolic MIDI music file format rather than wave format music files. The reason is that the strings resulting from straightforward discretizing the wave form files may be too sensitive to how we discretize.

6.3 Truly Feature-Free: The Case of Heterogenous Data

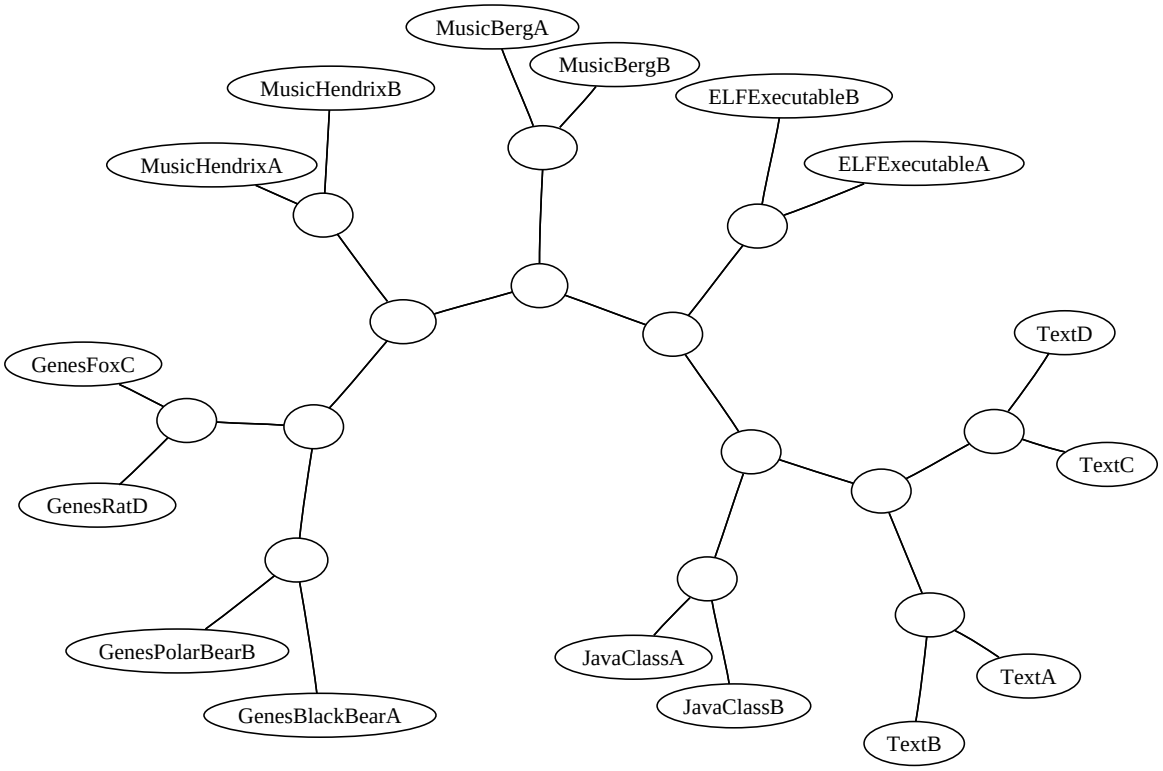

Figure 6.1: Classification of different file types. Tree agrees exceptionally well with NCD distance matrix: $S(T) = 0.984$.

We show the method is truly feature-free, or, anyway, as feature-free as we could possibly want, by showing once more its success in clustering data from truly different domains. No other method can apparently do so with so much success, since all other methods rfely on some definite features they analyze. In contrast, we just compress. Onr may say that the used compressor determines the features analyzed, but this seems ill-tergeted at general-purpose compressors which simply aim at analyzing general features as well as is possible. We test gross classification of files based on markedly different file types, as on page 84, recalling Figure 6.1 displayed there.

1. Four mitochondrial gene sequences, from a black bear, polar bear, fox, and rat.

2. Four excerpts from the novel *The Zeppelin's Passenger* by E. Phillips Oppenheim

3. Four MIDI files without further processing; two from Jimi Hendrix and two movements from Debussy's Suite bergamasque

4. Two Linux x86 ELF executables (the *cp* and *rm* commands)

5. Two compiled Java class files.

As expected, the program correctly classifies each of the different types of files together with like near like. The result is reported in Figure 6.1 with $S(T)$ equal to 0.984. Recall that S is defined as a linear normalized and inverted tree cost score as previously explained in Chapter 5, page 49. This means that this tree is very near an optimal best tree.

6.4 Music Categorization

The first result found relates to music analysis using gzip or bzip2 with preprocessed MIDI files. Surprisingly, the computer was able to reconstruct some common musical notions without any training whatsoever using just compression and quartet tree search.

A human expert, comparing different pieces of music with the aim to cluster likes together, will generally look for certain specific similarities. Previous attempts to automate this process do the same. Generally speaking, they take a file containing a piece of music and extract from it various specific numerical features, related to pitch, rhythm, harmony etc. One can extract such features using for instance Fourier transforms [114] or wavelet transforms [44]. The feature vectors corresponding to the various files are then classified or clustered using existing classification software, based on various standard statistical pattern recognition classifiers [114], Bayesian classifiers [33], hidden Markov models [19], ensembles of nearest-neighbor classifiers [44] or neural networks [33, 101]. For example, one feature would be to look for rhythm in the sense of beats per minute. One can make a histogram where each histogram bin corresponds to a particular tempo in beats-per-minute and the associated peak shows how frequent and strong that particular periodicity was over the entire piece. In [114] we see a gradual change from a few high peaks to many low and spread-out ones going from hip-hip, rock, jazz, to classical. One can use this similarity type to try to cluster pieces in these categories. However, such a method requires specific and detailed knowledge of the problem area, since one needs to know what features to look for.

Our aim is much more general. We do not look for similarity in specific features known to be relevant for classifying music; instead we apply a general mathematical theory of similarity. The aim is to capture, in a single similarity metric, *every effective metric*: effective versions of Hamming distance, Euclidean distance, edit distances, Lempel-Ziv distance, and so on. Such a metric would be able to simultaneously detect *all* similarities between pieces that other effective metrics can detect. As we have seen in Chapter 3, such a "universal" metric indeed exists. It is the NID metric which is approximated by the NCD metric.

In this section we apply this compression-based method to the classification of pieces of music. We perform various experiments on sets of mostly classical pieces given as MIDI (Musical Instrument Digital Interface) files. This contrasts with most earlier research, where the music

was digitized in some wave format or other (the only other research based on MIDI that we are aware of is [33]). We compute the distances between all pairs of pieces, and then build a tree containing those pieces in a way that is consistent with those distances. First, we show that our program can distinguish between various musical genres (classical, jazz, rock) quite well. Secondly, we experiment with various sets of classical pieces. The results are quite good (in the sense of conforming to our expectations) for small sets of data, but tend to get a bit worse for large sets. Considering the fact that the method knows nothing about music, or, indeed, about any of the other areas we have applied it to elsewhere, one is reminded of Dr Johnson's remark about a dog's walking on his hind legs: "It is not done well; but you are surprised to find it done at all."

6.4.1 Details of Our Implementation

Initially, we downloaded 118 separate MIDI (Musical Instrument Digital Interface, a versatile digital music format available on the world-wide-web) files selected from a range of classical composers, as well as some popular music. Each of these files was run through a preprocessor to extract just MIDI Note-On and Note-Off events. These events were then converted to a player-piano style representation, with time quantized in 0.05 second intervals. All instrument indicators, MIDI Control signals, and tempo variations were ignored. For each track in the MIDI file, we calculate two quantities: An *average volume* and a *modal note*. The average volume is calculated by averaging the volume (MIDI Note velocity) of all notes in the track. The modal note is defined to be the note pitch that sounds most often in that track. If this is not unique, then the lowest such note is chosen. The modal note is used as a key-invariant reference point from which to represent all notes. It is denoted by 0, higher notes are denoted by positive numbers, and lower notes are denoted by negative numbers. A value of 1 indicates a half-step above the modal note, and a value of -2 indicates a whole-step below the modal note. The tracks are sorted according to decreasing average volume, and then output in succession. For each track, we iterate through each time sample in order, outputting a single signed 8-bit value for each currently sounding note. Two special values are reserved to represent the end of a time step and the end of a track. This file is then used as input to the compression stage for distance matrix calculation and subsequent tree search.

Because we have already shown examples of the accuracy of the quartet tree reconstruction on artificial and controlled data, we will proceed immediately to natural data of considerably more interest than that already shown in earlier chapters.

6.4.2 Genres: Rock vs. Jazz vs. Classical

Before testing whether our program can see the distinctions between various classical composers, we first show that it can distinguish between three broader musical genres: classical music, rock, and jazz. This should be easier than making distinctions "within" classical music. All musical pieces we used are listed in the tables in the appendix. For the genre-experiment we used 12 classical pieces (the small set from Table 6.1, consisting of Bach, Chopin, and Debussy), 12 jazz

pieces (Table 6.2), and 12 rock pieces (Table 6.3). The tree that our program came up with is given in Figure 6.2. The $S(T)$ score is 0.858.

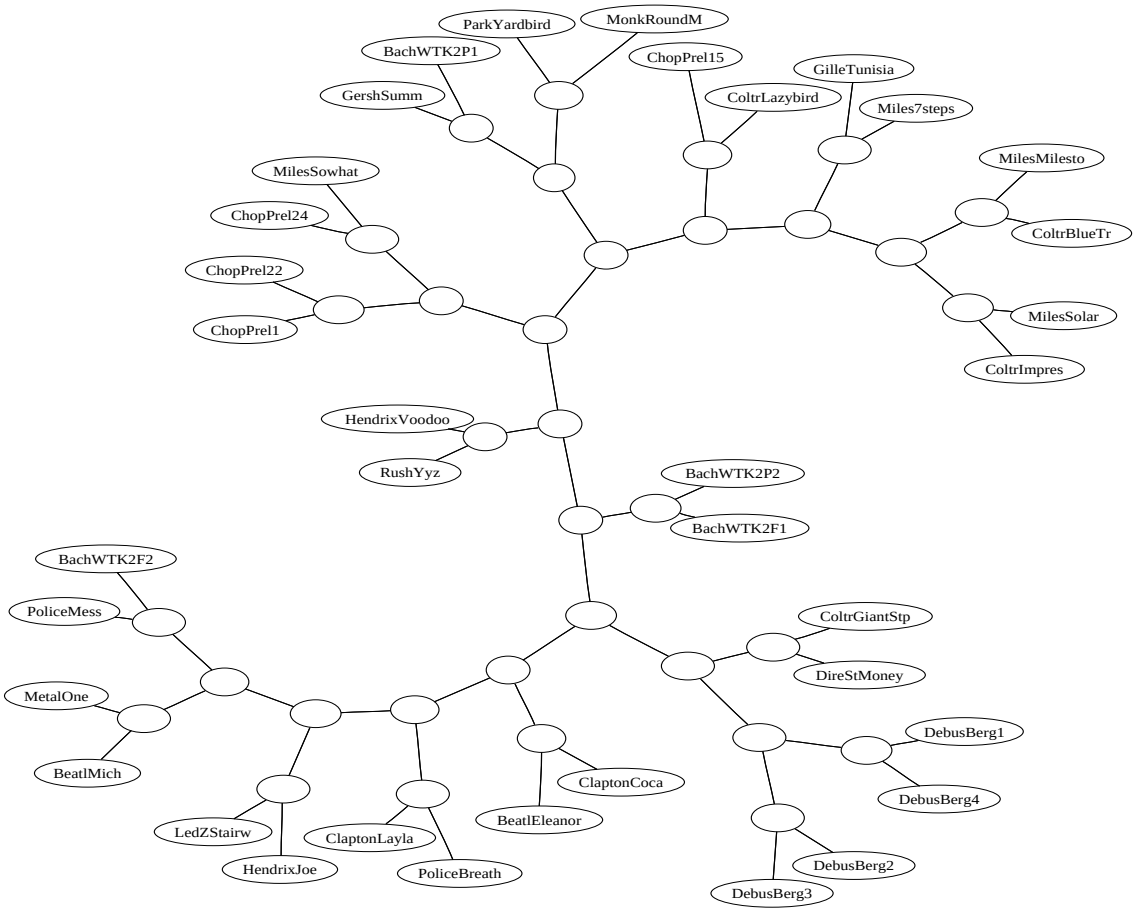

Figure 6.2: Output for the 36 pieces from 3 genres.

The discrimination between the 3 genres is good but not perfect. The upper branch of the tree contains 10 of the 12 jazz pieces, but also Chopin's Prélude no. 15 and a Bach Prelude. The two other jazz pieces, Miles Davis' "So what" and John Coltrane's "Giant steps" are placed elsewhere in the tree, perhaps according to some kinship that now escapes us but can be identified by closer studying of the objects concerned. Of the rock pieces, 9 are placed close together in the rightmost branch, while Hendrix's "Voodoo chile", Rush' "Yyz", and Dire Straits' "Money for nothing" are further away. In the case of the Hendrix piece this may be explained by the fact that it does not fit well in a specific genre. Most of the classical pieces are in the lower left part of the tree. Surprisingly, 2 of the 4 Bach pieces are placed elsewhere. It is not clear why this happens and may be considered an error of our program, since we perceive the 4 Bach pieces to be very close, both structurally and melodically (as they all come from the mono-thematic "Wohltemperierte Klavier"). However, Bach's is a seminal music and has been copied and cannibalized in all kinds of recognizable or hidden manners; closer scrutiny could reveal likenesses in its present company

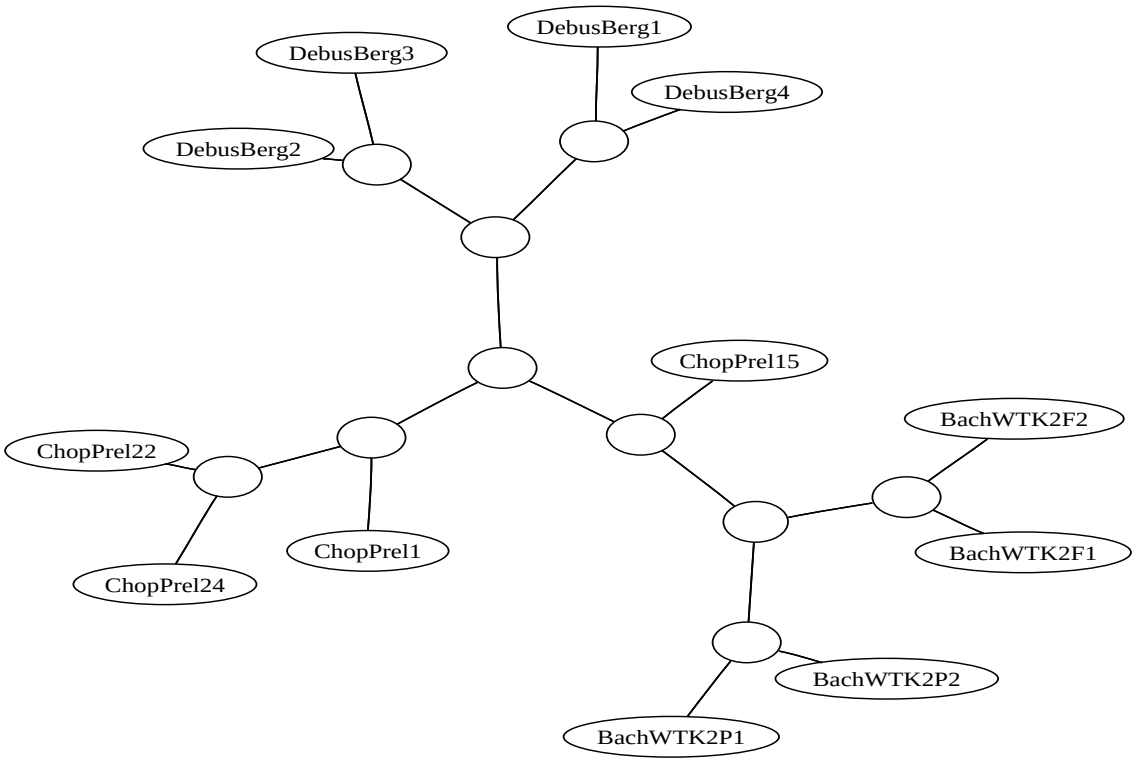

Figure 6.3: Output for the 12-piece set.

that are not now apparent to us. In effect our similarity engine aims at the ideal of a perfect data mining process, discovering unknown features in which the data can be similar.

6.4.3 Classical Piano Music (Small Set)

In Table 6.1 we list all 60 classical piano pieces used, together with their abbreviations. Some of these are complete compositions, others are individual movements from larger compositions. They all are piano pieces, but experiments on 34 movements of symphonies gave very similar results (Section 6.4.6). Apart from running our program on the whole set of 60 piano pieces, we also tried it on two smaller sets: a small 12-piece set, indicated by '(s)' in the table, and a medium-size 32-piece set, indicated by '(s)' or '(m)'.

The small set encompasses the 4 movements from Debussy's Suite bergamasque, 4 movements of book 2 of Bach's Wohltemperierte Klavier, and 4 preludes from Chopin's opus 28. As one can see in Figure 6.3, our program does a pretty good job at clustering these pieces. The $S(T)$ score is also high: 0.958. The 4 Debussy movements form one cluster, as do the 4 Bach pieces. The only imperfection in the tree, judged by what one would intuitively expect, is that Chopin's Prélude no. 15 lies a bit closer to Bach than to the other 3 Chopin pieces. This Prélude no 15, in fact, consistently forms an odd-one-out in our other experiments as well. This is an example of pure data mining, since there is some musical truth to this, as no. 15 is perceived as

by far the most eccentric among the 24 Préludes of Chopin's opus 28.

6.4.4 Classical Piano Music (Medium Set)

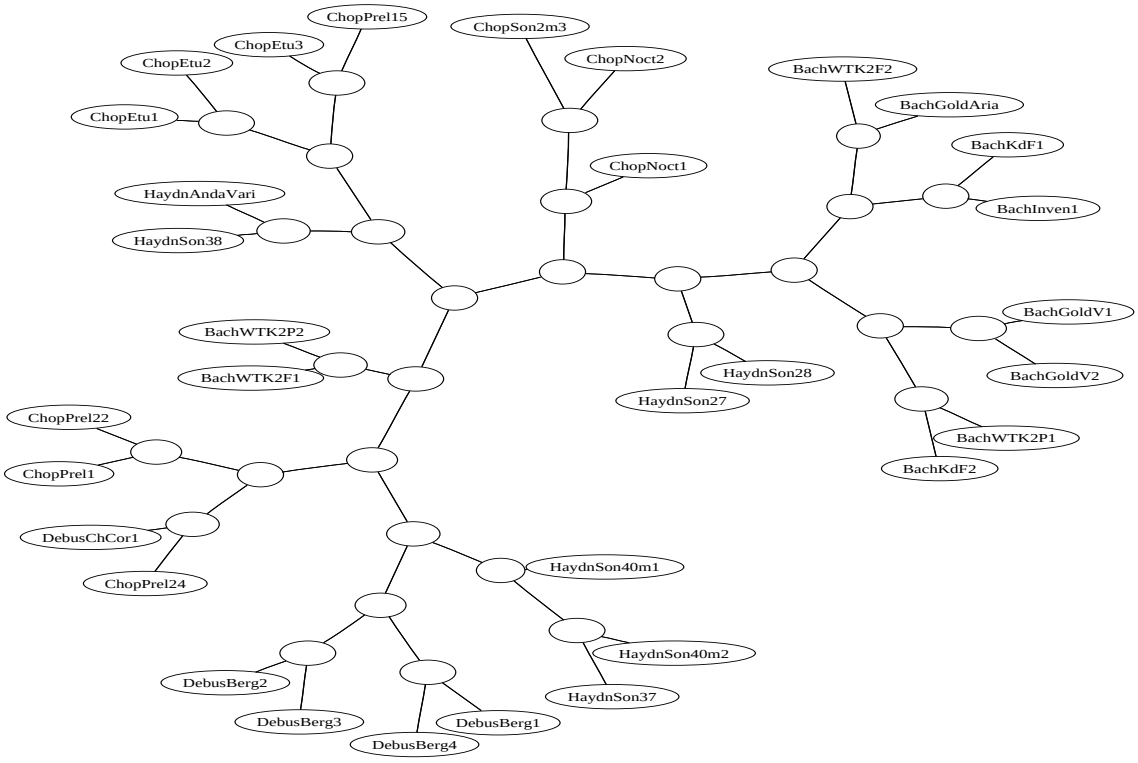

Figure 6.4: Output for the 32-piece set.

The medium set adds 20 pieces to the small set: 6 additional Bach pieces, 6 additional Chopins, 1 more Debussy piece, and 7 pieces by Haydn. The experimental results are given in Figure 6.4. The $S(T)$ score is slightly lower than in the small set experiment: 0.895. Again, there is a lot of structure and expected clustering. Most of the Bach pieces are together, as are the four Debussy pieces from the Suite bergamasque. These four should be together because they are movements from the same piece; The fifth Debussy item is somewhat apart since it comes from another piece. Both the Haydn and the Chopin pieces are clustered in little sub-clusters of two or three pieces, but those sub-clusters are scattered throughout the tree instead of being close together in a larger cluster. These small clusters may be an imperfection of the method, or, alternatively point at musical similarities between the clustered pieces that transcend the similarities induced by the same composer. Indeed, this may point the way for further musicological investigation.

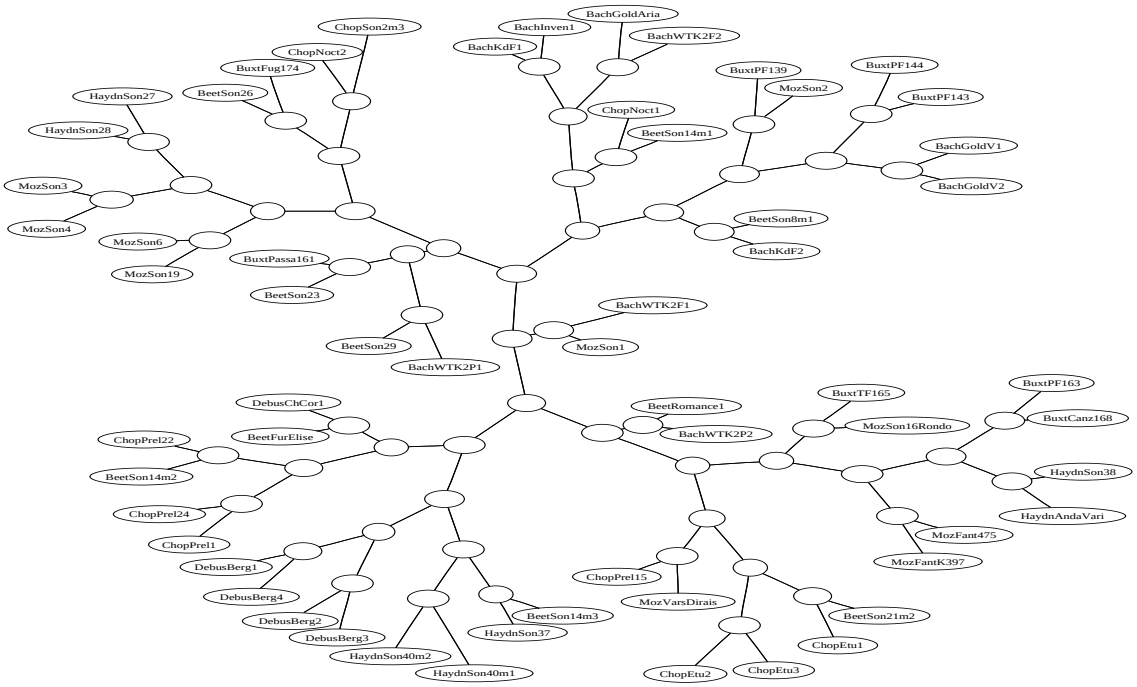

Figure 6.5: Output for the 60-piece set.

6.4.5 Classical Piano Music (Large Set)

Figure 6.5 gives the output of a run of our program on the full set of 60 pieces. This adds 10 pieces by Beethoven, 8 by Buxtehude, and 10 by Mozart to the medium set. The experimental results are given in Figure 6.5. The results are still far from random, but leave more to be desired than the smaller-scale experiments. Indeed, the $S(T)$ score has dropped further from that of the medium-sized set to 0.844. This may be an artifact of the interplay between the relatively small size, and large number, of the files compared: (i) the distances estimated are less accurate; (ii) the number of quartets with conflicting requirements increases; and (iii) the computation time rises to such an extent that the correctness score of the displayed cluster graph within the set time limit is lower than in the smaller samples. Nonetheless, Bach and Debussy are still reasonably well clustered, but other pieces (notably the Beethoven and Chopin ones) are scattered throughout the tree. Maybe this means that individual music pieces by these composers are more similar to pieces of other composers than they are to each other? The placement of the pieces is closer to intuition on a small level (for example, most pairing of siblings corresponds to musical similarity in the sense of the same composer) than on the larger level. This is similar to the phenomenon of little sub-clusters of Haydn or Chopin pieces that we saw in the medium-size experiment.

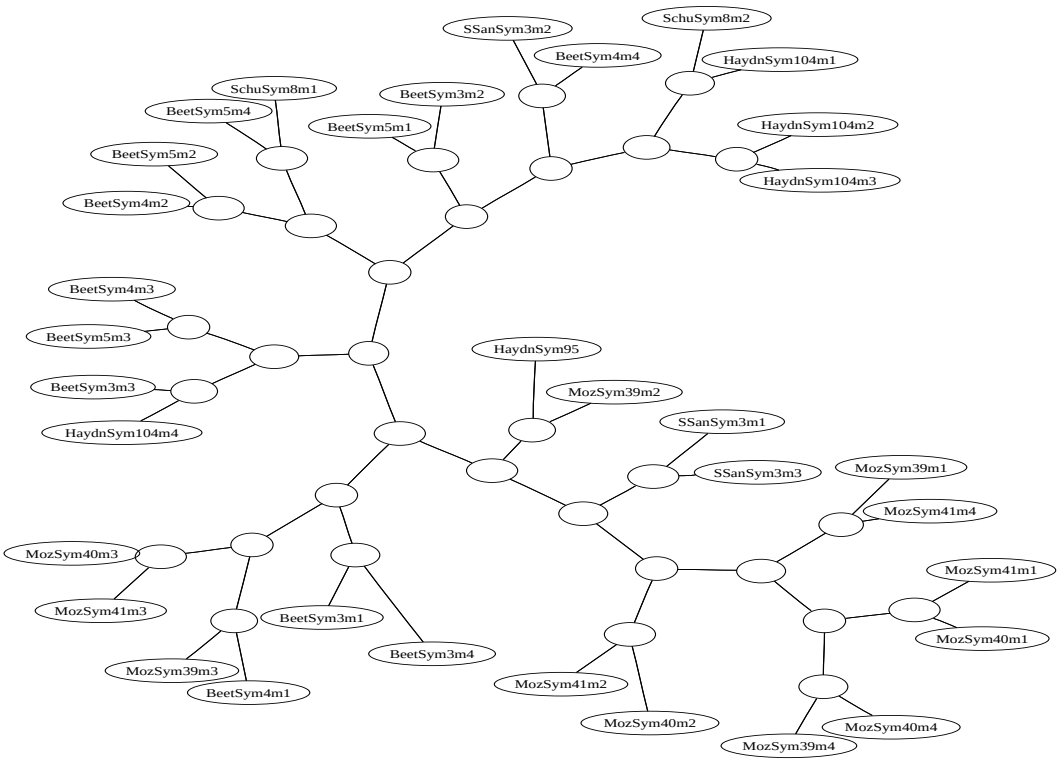

Figure 6.6: Output for the set of 34 movements of symphonies.

6.4.6 Clustering Symphonies

Finally, we tested whether the method worked for more complicated music, namely 34 symphonic pieces. We took two Haydn symphonies (no. 95 in one file, and the four movements of 104), three Mozart symphonies (39, 40, 41), three Beethoven symphonies (3, 4, 5), of Schubert's Unfinished symphony, and of Saint-Saens Symphony no. 3. The results are reported in Figure 6.6, with a quite reasonable $S(T)$ score of 0.860.

6.4.7 Future Music Work and Conclusions

Our research raises many questions worth looking into further:

- The program can be used as a data mining machine to discover hitherto unknown similarities between music pieces of different composers or indeed different genres. In this manner we can discover plagiarism or indeed honest influences between music pieces and composers. Indeed, it is thinkable that we can use the method to discover seminality of composers, or separate music eras and fads.

- A very interesting application of our program would be to select a plausible composer for a newly discovered piece of music of which the composer is not known. In addition to such

a piece, this experiment would require a number of pieces from known composers that are plausible candidates. We would just run our program on the set of all those pieces, and see where the new piece is placed. If it lies squarely within a cluster of pieces by composer such-and-such, then that would be a plausible candidate composer for the new piece.

- Each run of our program is different—even on the same set of data—because of our use of randomness for choosing mutations in the quartet method. It would be interesting to investigate more precisely how stable the outcomes are over different such runs.

- At various points in our program, somewhat arbitrary choices were made. Some examples are: the compression algorithms we use (all practical compression algorithms will fall short of Kolmogorov complexity, but some less so than others); the way we transform the MIDI files (choice of length of time interval, choice of note-representation); the cost function in the quartet method. Other choices are possible and may or may not lead to better clustering.[1] Ideally, one would like to have well-founded theoretical reasons to decide such choices in an optimal way. Lacking those, trial-and-error seems the only way to deal with them.

- The experimental results got decidedly worse when the number of pieces grew. Better compression methods may improve this situation, but the effect is probably due to unknown scaling problems with the quartet method or nonlinear scaling of possible similarities in a larger group of objects (akin to the phenomenon described in the so-called "birthday paradox": in a group of about two dozen people there is a high chance that at least two of the people have the same birthday). Inspection of the underlying distance matrices makes us suspect the latter.

- Our program is not very good at dealing with very small data files (100 bytes or so), because significant compression only kicks in for larger files. We might deal with this by comparing various sets of such pieces against each other, instead of individual ones.

6.4.8 Details of the Music Pieces Used

[1] We compared the quartet-based approach to the tree reconstruction with alternatives. One such alternative that we tried is to compute the Minimum Spanning Tree (MST) from the matrix of distances. MST has the advantage of being very efficiently computable, but resulted in trees that were much worse than the quartet method. It appears that the quartet method is extremely sensitive in extracting information even from small differences in the entries of the distance matrix, where other methods would be led to error.

Composer	Piece	Acronym
J.S. Bach (10)	Wohltemperierte Klavier II: Preludes and fugues 1,2	BachWTK2{F,P}{1,2} (s)
	Goldberg Variations: Aria, Variations 1,2	BachGold{Aria,V1,V2} (m)
	Kunst der Fuge: Variations 1,2	BachKdF{1,2} (m)
	Invention 1	BachInven1 (m)
Beethoven (10)	Sonata no. 8 (Pathetique), 1st movement	BeetSon8m1
	Sonata no. 14 (Mondschein), 3 movements	BeetSon14m{1,2,3}
	Sonata no. 21 (Waldstein), 2nd movement	BeetSon21m2
	Sonata no. 23 (Appassionata)	BeetSon23
	Sonata no. 26 (Les Adieux)	BeetSon26
	Sonata no. 29 (Hammerklavier)	BeetSon29
	Romance no. 1	BeetRomance1
	Für Elise	BeetFurElise
Buxtehude (8)	Prelude and fugues, BuxWV 139,143,144,163	BuxtPF{139,143,144,163}
	Toccata and fugue, BuxWV 165	BuxtTF165
	Fugue, BuxWV 174	BuxtFug174
	Passacaglia, BuxWV 161	BuxtPassa161
	Canzonetta, BuxWV 168	BuxtCanz168
Chopin (10)	Préludes op. 28: 1, 15, 22, 24	ChopPrel{1,15,22,24} (s)
	Etudes op. 10, nos. 1, 2, and 3	ChopEtu{1,2,3} (m)
	Nocturnes nos. 1 and 2	ChopNoct{1,2} (m)
	Sonata no. 2, 3rd movement	ChopSon2m3 (m)
Debussy (5)	Suite bergamasque, 4 movements	DebusBerg{1,2,3,4} (s)
	Children's corner suite (Gradus ad Parnassum)	DebusChCorm1 (m)
Haydn (7)	Sonatas nos. 27, 28, 37, and 38	HaydnSon{27,28,37,38} (m)
	Sonata no. 40, movements 1,2	HaydnSon40m{1,2} (m)
	Andante and variations	HaydnAndaVari (m)
Mozart (10)	Sonatas nos. 1,2,3,4,6,19	MozSon{1,2,3,4,6,19}
	Rondo from Sonata no. 16	MozSon16Rondo
	Fantasias K397, 475	MozFantK{397,475}
	Variations "Ah, vous dirais-je madam"	MozVarsDirais

Table 6.1: The 60 classical pieces used ('m' indicates presence in the medium set, 's' in the small and medium sets).

6.5 Genomics and Phylogeny

In recent years, as the complete genomes of various species become available, it has become possible to do whole genome phylogeny (this overcomes the problem that using different targeted parts of the genome, or proteins, may give different trees [94]). Traditional phylogenetic methods on individual genes depended on multiple alignment of the related proteins and on the model of evolution of individual amino acids. Neither of these is practically applicable to the genome level. In absence of such models, a method which can compute the shared information between two sequences is useful because biological sequences encode information, and the occurrence of evolutionary events (such as insertions, deletions, point mutations, rearrangements, and inversions) separating two sequences sharing a common ancestor will result in the loss of their shared information. Our method (in the form of the CompLearn Toolkit) is a fully automated software

John Coltrane	Blue trane
	Giant steps
	Lazy bird
	Impressions
Miles Davis	Milestones
	Seven steps to heaven
	Solar
	So what
George Gershwin	Summertime
Dizzy Gillespie	Night in Tunisia
Thelonious Monk	Round midnight
Charlie Parker	Yardbird suite

Table 6.2: The 12 jazz pieces used.

The Beatles	Eleanor Rigby
	Michelle
Eric Clapton	Cocaine
	Layla
Dire Straits	Money for nothing
Led Zeppelin	Stairway to heaven
Metallica	One
Jimi Hendrix	Hey Joe
	Voodoo chile
The Police	Every breath you take
	Message in a bottle
Rush	Yyz

Table 6.3: The 12 rock pieces used.

tool based on such a distance to compare two genomes.

6.5.1 Mammalian Evolution:

In evolutionary biology the timing and origin of the major extant placental clades (groups of organisms that have evolved from a common ancestor) continues to fuel debate and research. Here, we provide evidence by whole mitochondrial genome phylogeny for competing hypotheses in two main questions: the grouping of the Eutherian orders, and the Therian hypothesis versus the Marsupionta hypothesis.

Eutherian Orders: It was demonstrated in [77] that a whole mitochondrial genome phylogeny of the Eutherians (placental mammals) can be reconstructed automatically from a set of *unaligned* complete mitochondrial genomes by use of an early form of our compression method, using standard software packages. As more genomic material has become available, the debate in biology has intensified concerning which two of the three main groups of placental mammals

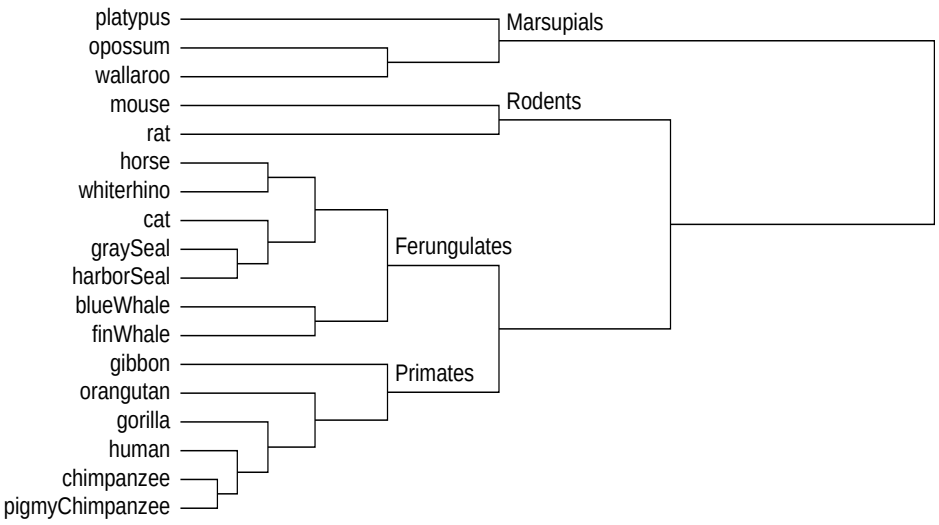

Figure 6.7: The evolutionary tree built from complete mammalian mtDNA sequences of 24 species, using the NCD matrix of Figure 4.14 on page 70 where it was used to illustrate a point of hierarchical clustering versus flat clustering. We have redrawn the tree from our output to agree better with the customary phylogeny tree format. The tree agrees exceptionally well with the NCD distance matrix: $S(T) = 0.996$.

are more closely related: Primates, Ferungulates, and Rodents. In [18], the maximum likelihood method of phylogeny tree reconstruction gave evidence for the (Ferungulates, (Primates, Rodents)) grouping for half of the proteins in the mitochondrial genomes investigated, and (Rodents, (Ferungulates, Primates)) for the other halves of the mt genomes. In that experiment they aligned 12 concatenated mitochondrial proteins, taken from 20 species: the humble rat (*Rattus norvegicus*), house mouse (*Mus musculus*), grey seal (*Halichoerus grypus*), harbor seal (*Phoca vitulina*), cat (*Felis catus*), white rhino (*Ceratotherium simum*), horse (*Equus caballus*), finback whale (*Balaenoptera physalus*), blue whale (*Balaenoptera musculus*), cow (*Bos taurus*), gibbon (*Hylobates lar*), gorilla (*Gorilla gorilla*), human (*Homo sapiens*), chimpanzee (*Pan troglodytes*), pygmy chimpanzee (*Pan paniscus*), orangutan (*Pongo pygmaeus*), Sumatran orangutan (*Pongo pygmaeus abelii*), using opossum (*Didelphis virginiana*), wallaroo (*Macropus robustus*), and the platypus (*Ornithorhynchus anatinus*) as outgroup. In [80, 77] we used the whole mitochondrial genome of the same 20 species, computing the NCD distances (or a closely related distance in [80]), using the GenCompress compressor, followed by tree reconstruction using the neighbor joining program in the MOLPHY package [99] to confirm the commonly believed morphology-supported hypothesis (Rodents, (Primates, Ferungulates)). Repeating the experiment using the hypercleaning method [13] of phylogeny tree reconstruction gave the same result. Here, we repeated this experiment several times using the CompLearn Toolkit using our new quartet method for reconstructing trees, and computing the NCD with various compressors (gzip, bzip2, PPMZ), again always with the same result. These experiments are not reported since they are subsumed by the larger experiment of Figure 6.7.

Marsupionta and Theria: The extant monophyletic divisions of the class Mammalia are the Prototheria (monotremes: mammals that procreate using eggs), Metatheria (marsupials: mammals that procreate using pouches), and Eutheria (placental mammals: mammals that procreate using placentas). The sister relationships between these groups is viewed as the most fundamental question in mammalian evolution [55]. Phylogenetic comparison by either anatomy or mitochondrial genome has resulted in two conflicting hypotheses: the gene-isolation-supported *Marsupionta hypothesis*: ((Prototheria, Metatheria), Eutheria) versus the morphology-supported *Theria hypothesis*: (Prototheria, (Metatheria, Eutheria)), the third possiblity apparently not being held seriously by anyone. There has been a lot of support for either hypothesis; recent support for the Theria hypothesis was given in [55] by analyzing a large nuclear gene (M6P/IG2R), viewed as important across the species concerned, and even more recent support for the Marsupionta hypothesis was given in [51] by phylogenetic analysis of another sequence from the nuclear gene (18S rRNA) and by the whole mitochondrial genome.

Experimental Evidence: To test the Eutherian orders simultaneously with the Marsupionta versus Theria hypothesis, we added four animals to the above twenty: Australian echidna (*Tachyglossus aculeatus*), brown bear (*Ursus arctos*), polar bear (*Ursus maritimus*), using the common carp (*Cyprinus carpio*) as the outgroup. Interestingly, while there are many species of Eutheria and Metatheria, there are only three species of now living Prototheria known: platypus, and two types of echidna (or spiny anteater). So our sample of the Prototheria is large. The addition of the new species might be risky in that the addition of new relations is known to distort the previous phylogeny in traditional computational genomics practice. With our method, using the full genome and obtaining a single tree with a very high confidence $S(T)$ value, that risk is not as great as in traditional methods obtaining ambiguous trees with bootstrap (statistic support) values on the edges. The mitochondrial genomes of the total of 24 species we used were downloaded from the GenBank Database on the world-wide web. Each is around 17,000 bases. The NCD distance matrix was computed using the compressor PPMZ. The resulting phylogeny, with an almost maximal $S(T)$ score of 0.996 supports anew the currently accepted grouping (Rodents, (Primates, Ferungulates)) of the Eutherian orders, and additionally the Marsupionta hypothesis ((Prototheria, Metatheria), Eutheria), see Figure 6.7 (reproducing Figure 4.12 for the readers convenience). Overall, our whole-mitochondrial NCD analysis supports the following hypothesis:

$$\overbrace{(\underbrace{(primates, ferungulates)}_{\text{Eutheria}}(rodents, (Metatheria, Prototheria)))}^{\text{Mammalia}},$$

which indicates that the rodents, and the branch leading to the Metatheria and Prototheria, split off early from the branch that led to the primates and ferungulates. Inspection of the distance matrix Figure 4.14 on page 70 (exhibited earler in the context of hierarchical versus flat clustering) shows that the primates are very close together, as are the rodents, the Metatheria, and the Prototheria. These are tightly-knit groups with relatively close NCD's. The ferungulates are a much looser group with generally distant NCD's. The intergroup distances show that the Prototheria

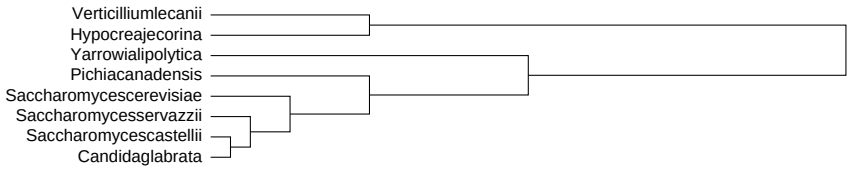

Figure 6.8: Dendrogram of mitochondrial genomes of fungi using NCD. This represents the distance matrix precisely with $S(T) = 0.999$.

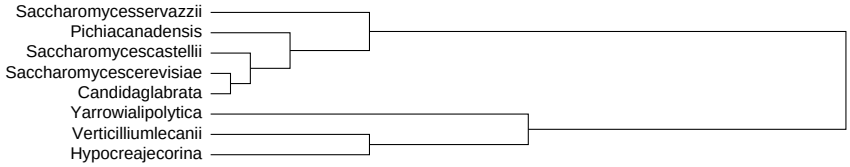

Figure 6.9: Dendrogram of mitochondrial genomes of fungi using block frequencies. This represents the distance matrix precisely with $S(T) = 0.999$.

are furthest away from the other groups, followed by the Metatheria and the rodents. Also the fine-structure of the tree is consistent with biological wisdom.

6.5.2 SARS Virus:

In another experiment we clustered the SARS virus after its sequenced genome was made publicly available, in relation to potential similar virii. The 15 virus genomes were downloaded from The Universal Virus Database of the International Committee on Taxonomy of Viruses, available on the world-wide web. The SARS virus was downloaded from Canada's Michael Smith Genome Sciences Centre which had the first public SARS Coronavirus draft whole genome assembly available for download (SARS TOR2 draft genome assembly 120403). The NCD distance matrix was computed using the compressor bzip2. The relations in Figure 4.9 are very similar to the definitive tree based on medical-macrobio-genomics analysis, appearing later in the New England Journal of Medicine, [63]. We depicted the figure in the ternary tree style, rather than the genomics-dendrogram style, since the former is more precise for visual inspection of proximity relations.

6.5.3 Analysis of Mitochondrial Genomes of Fungi:

As a pilot for applications of the CompLearn Toolkit in fungi genomics research, the group of T. Boekhout, E. Kuramae, V. Robert, of the Fungal Biodiversity Center, Royal Netherlands Academy of Sciences, supplied us with the mitochondrial genomes of *Candida glabrata, Pichia canadensis, Saccharomyces cerevisiae, S. castellii, S. servazzii, Yarrowia lipolytica* (all yeasts), and two filamentous ascomycetes *Hypocrea jecorina* and *Verticillium lecanii*. The NCD distance matrix was computed using the compressor PPMZ. The resulting tree is depicted in Figure 6.8. The interpretation of the fungi researchers is "the tree clearly clustered the ascomycetous yeasts

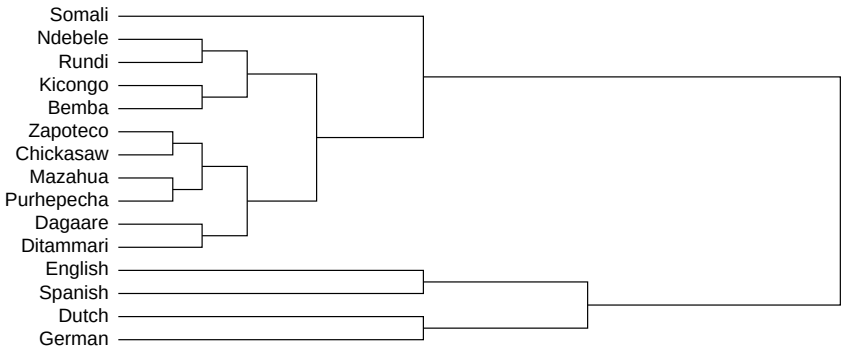

Figure 6.10: Clustering of Native-American, Native-African, and Native-European languages. $S(T) = 0.928$.

versus the two filamentous Ascomycetes, thus supporting the current hypothesis on their classification (for example, see [65]). Interestingly, in a recent treatment of the Saccharomycetaceae, S. servazii, S. castellii and C. glabrata were all proposed to belong to genera different from Saccharomyces, and this is supported by the topology of our tree as well [64]."

To compare the veracity of the NCD clustering with a more feature-based clustering, we also calculated the pairwise distances as follows: Each file is converted to a 4096-dimensional vector by considering the frequency of all (overlapping) 6-byte contiguous blocks. The l2-distance (Euclidean distance) is calculated between each pair of files by taking the square root of the sum of the squares of the component-wise differences. These distances are arranged into a distance matrix and linearly scaled to fit the range $[0, 1.0]$. Finally, we ran the clustering routine on this distance matrix. The results are in Figure 6.9. As seen by comparing with the NCD-based Figure 6.8 there are apparent misplacements when using the Euclidean distance in this way. Thus, in this simple experiment, the NCD performed better, that is, agreed more precisely with accepted biological knowledge.

6.6 Language Trees

Our method improves the results of [8], using a linguistic corpus of "The Universal Declaration of Human Rights (UDoHR)" [27] in 52 languages. Previously, [8] used an asymmetric measure based on relative entropy, and the full matrix of the pair-wise distances between all 52 languages, to build a language classification tree. This experiment was repeated (resulting in a somewhat better tree) using the compression method in [77] using standard biological software packages to construct the phylogeny. We have redone this experiment, and done new experiments, using the CompLearn Toolkit. Here, we report on an experiment to separate radically different language families. We downloaded the language versions of the UDoHR text in English, Spanish, Dutch, German (Native-European), Pemba, Dendi, Ndbele, Kicongo, Somali, Rundi, Ditammari, Dagaare (Native African), Chikasaw, Perhupecha, Mazahua, Zapoteco (Native-American), and didn't preprocess them except for removing initial identifying information. We used an Lempel-Ziv-type compressor *gzip* to compress text sequences of sizes not exceeding the length of the

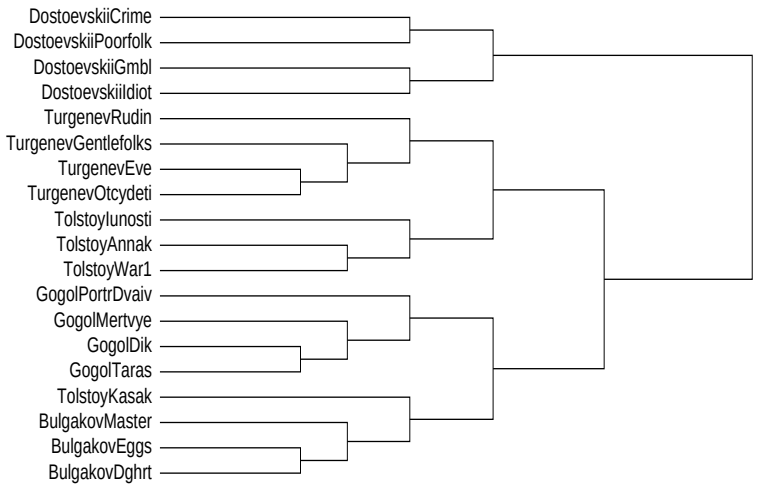

Figure 6.11: Clustering of Russian writers. Legend: I.S. Turgenev, 1818–1883 [Father and Sons, Rudin, On the Eve, A House of Gentlefolk]; F. Dostoyevsky 1821–1881 [Crime and Punishment, The Gambler, The Idiot; Poor Folk]; L.N. Tolstoy 1828–1910 [Anna Karenina, The Cossacks, Youth, War and Piece]; N.V. Gogol 1809–1852 [Dead Souls, Taras Bulba, The Mysterious Portrait, How the Two Ivans Quarrelled]; M. Bulgakov 1891–1940 [The Master and Margarita, The Fatefull Eggs, The Heart of a Dog]. $S(T) = 0.949$.

sliding window *gzip* uses (32 kilobytes), and compute the NCD for each pair of language sequences. Subsequently we clustered the result. We show the outcome of one of the experiments in Figure 6.10. Note that three groups are correctly clustered, and that even the subclusters of the European languages are correct (English is grouped with the Romance languages because it contains up to 40% admixture of words from Latin origin).

6.7 Literature

The texts used in this experiment were down-loaded from the world-wide web in original Cyrillic-lettered Russian and in Latin-lettered English by L. Avanasiev (Moldavian MSc student at the University of Amsterdam). The compressor used to compute the NCD matrix was bzip2. We clustered Russian literature in the original (Cyrillic) by Gogol, Dostojevski, Tolstoy, Bulgakov,Tsjechov, with three or four different texts per author. Our purpose was to see whether the clustering is sensitive enough, and the authors distinctive enough, to result in clustering by author. In Figure 6.11 we see a perfect clustering. Considering the English translations of the same texts, in Figure 6.12, we see errors in the clustering. Inspection shows that the clustering is now partially based on the translator. It appears that the translator superimposes his characteristics on the texts, partially suppressing the characteristics of the original authors. In other experiments we separated authors by gender and by period.

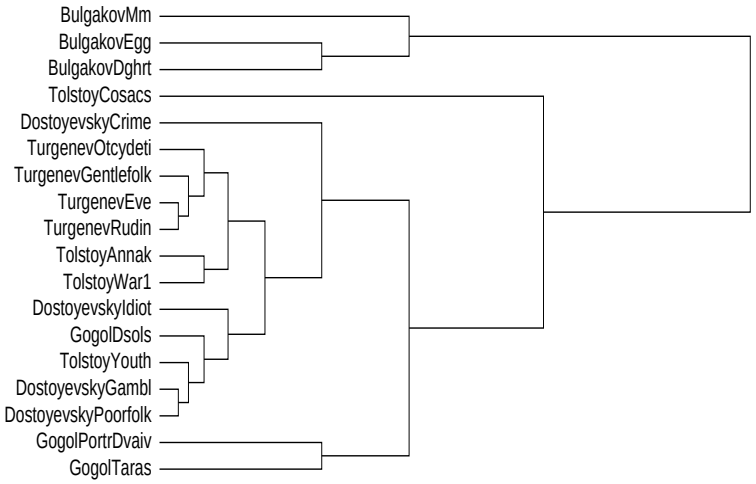

Figure 6.12: Clustering of Russian writers translated in English. The translator is given in brackets after the titles of the texts. Legend: I.S. Turgenev, 1818–1883 [Father and Sons (R. Hare), Rudin (Garnett, C. Black), On the Eve (Garnett, C. Black), A House of Gentlefolk (Garnett, C. Black)]; F. Dostoyevsky 1821–1881 [Crime and Punishment (Garnett, C. Black), The Gambler (C.J. Hogarth), The Idiot (E. Martin); Poor Folk (C.J. Hogarth)]; L.N. Tolstoy 1828–1910 [Anna Karenina (Garnett, C. Black), The Cossacks (L. and M. Aylmer), Youth (C.J. Hogarth), War and Piece (L. and M. Aylmer)]; N.V. Gogol 1809–1852 [Dead Souls (C.J. Hogarth), Taras Bulba ($\approx$ G. Tolstoy, 1860, B.C. Baskerville), The Mysterious Portrait + How the Two Ivans Quarrelled ($\approx$ I.F. Hapgood]; M. Bulgakov 1891–1940 [The Master and Margarita (R. Pevear, L. Volokhonsky), The Fatefull Eggs (K. Gook-Horujy), The Heart of a Dog (M. Glenny)]. $S(T) = 0.953$.

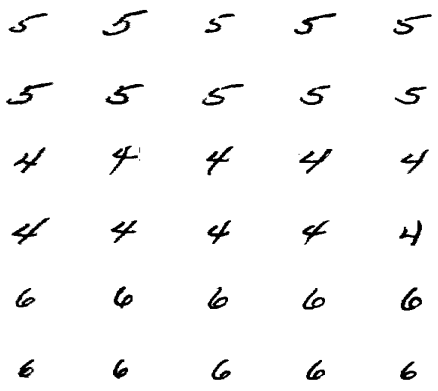

Figure 6.13: Images of handwritten digits used for OCR.

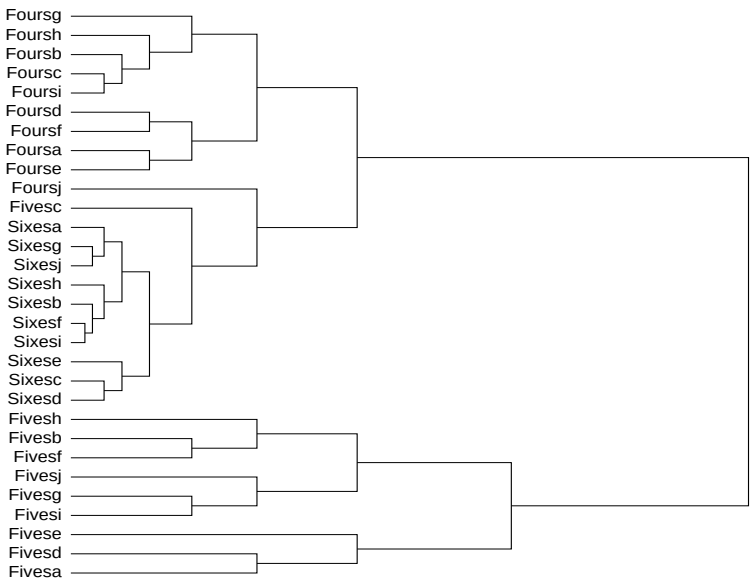

Figure 6.14: Clustering of the OCR images. $S(T) = 0.901$.

6.8 Optical Character Recognition

Perhaps surprisingly, it turns out that scanning a picture in raster row-major order retains enough regularity in both dimensions for the compressor to grasp. A simple task along these lines is to cluster handwritten characters. The handwritten characters in Figure 6.13 were downloaded from the NIST Special Data Base 19 (optical character recognition database) on the world-wide web. Each file in the data directory contains 1 digit image, either a four, five, or six. Each pixel is a single character; '#' for a black pixel, '.' for white. Newlines are added at the end of each line. Each character is 128x128 pixels. The NCD matrix was computed using the compressor PPMZ. Figure 6.14 shows each character that is used. There are 10 of each digit "4," "5," "6," making a total of 30 items in this experiment. All but one of the 4's are put in the subtree rooted at $n1$, all but one of the 5's are put in the subtree rooted at $n4$, and all 6's are put in the subtree rooted at $n3$. The remaining 4 and 5 are in the branch $n23, n13$ joining $n6$ and $n3$. So 28 items out of 30 are clustered correctly, that is, 93%.

Classification In the experiment above we used only 3 digits. Using the full set of decimal digits results in a lower clustering accuracy. However, we can use the NCD as a oblivious feature extraction technique to convert generic objects into finite-dimensional vectors. This is done using the *anchor method* which we introduced in Chapter 5, Section 5.2. We have used this technique to train a support vector machine (SVM) based OCR system to classify handwritten digits by extracting 80 distinct, ordered NCD features from each input image. The images are black and white square rasterized images. The anchors are chosen once and for all at the beginning of the experiment randomly from the training object pool, ensuring that eight examples of each class are

chosen. Once chosen, the anchors are kept in order (so that the first coordinate always refers to the same anchor and so on) and used to translate all other training data files into 80-dimensional vectors. In this initial stage of ongoing research, by our oblivious method of compression-based clustering to supply a kernel for an SVM classifier, we achieved a handwritten single decimal digit recognition accuracy of 85%. The current state-of-the-art for this problem, after half a century of interactive feature-driven classification research, in the upper ninety % level [87, 113]. All experiments are bench marked on the standard NIST Special Data Base 19 (optical character recognition database).

6.9 Astronomy

As a proof of principle we clustered data from unknown objects, for example objects that are far away. In [5] observations of the microquasar GRS 1915+105 made with the Rossi X-ray Timing Explorer were analyzed. The interest in this microquasar stems from the fact that it was the first Galactic object to show a certain behavior (superluminal expansion in radio observations). Photonometric observation data from X-ray telescopes were divided into short time segments (usually in the order of one minute), and these segments have been classified into a bewildering array of fifteen different modes after considerable effort. Briefly, spectrum hardness ratios (roughly, "color") and photon count sequences were used to classify a given interval into categories of variability modes. From this analysis, the extremely complex variability of this source was reduced to transitions between three basic states, which, interpreted in astronomical terms, gives rise to an explanation of this peculiar source in standard black-hole theory. The data we used in this experiment made available to us by M. Klein Wolt (co-author of the above paper) and T. Maccarone, both researchers at the Astronomical Institute "Anton Pannekoek", University of Amsterdam. The observations are essentially time series, and our aim was experimenting with our method as a pilot to more extensive joint research. Here the task was to see whether the clustering would agree with the classification above. The NCD matrix was computed using the compressor PPMZ. The results are in Figure 6.15. We clustered 12 objects, consisting of three intervals from four different categories denoted as $\delta, \gamma, \phi, \theta$ in Table 1 of [5]. In Figure 6.15 we denote the categories by the corresponding Roman letters D, G, P, and T, respectively. The resulting tree groups these different modes together in a way that is consistent with the classification by experts for these observations. The oblivious compression clustering corresponds precisely with the laborious feature-driven classification in [5].

6.10 Conclusion

To interpret what the NCD is doing, and to explain its remarkable accuracy and robustness across application fields and compressors, the intuition is that the NCD minorizes all similarity metrics based on features that are captured by the reference compressor involved. Such features must be relatively *simple* in the sense that they are expressed by an aspect that the compressor analyzes (for example frequencies, matches, repeats). Certain sophisticated features may well be express-

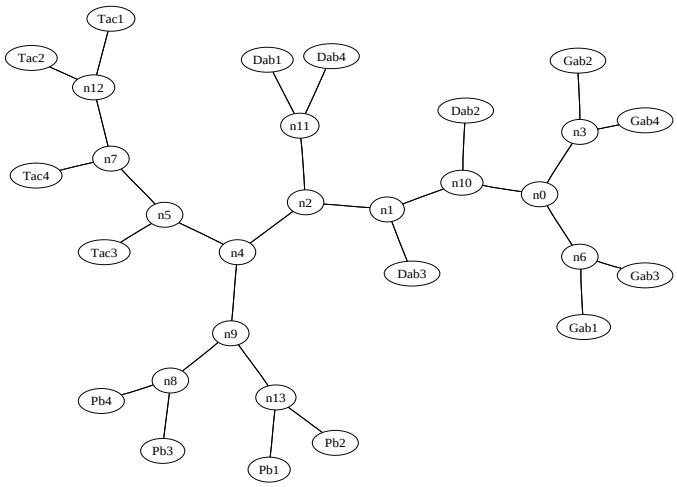

Figure 6.15: 16 observation intervals of GRS 1915+105 from four classes. The initial capital letter indicates the class corresponding to Greek lower case letters in [5]. The remaining letters and digits identify the particular observation interval in terms of finer features and identity. The *T*-cluster is top left, the *P*-cluster is bottom left, the *G*-cluster is to the right, and the *D*-cluster in the middle. This tree almost exactly represents the underlying NCD distance matrix: $S(T) = 0.994$.

ible as combinations of such simple features, and are therefore themselves simple features in this sense. The extensive experimenting above shows that even elusive features are captured.

A potential application of our non-feature (or rather, many-unknown-feature) approach is exploratory. Presented with data for which the features are as yet unknown, certain dominant features governing similarity are automatically discovered by the NCD. Examining the data underlying the clusters may yield this hitherto unknown dominant feature.

Our experiments indicate that the NCD has application in two new areas of support vector machine (SVM) based learning. Firstly, we find that the inverted NCD (1-NCD) is useful as a kernel for generic objects in SVM learning. Secondly, we can use the normal NCD as a feature-extraction technique to convert generic objects into finite-dimensional vectors, see the last paragraph of Section 6.8. In effect our similarity engine aims at the ideal of a perfect data mining process, discovering unknown features in which the data can be similar. This is the subject of current joint research in genomics of fungi, clinical molecular genetics, and radio-astronomy.

The results in this section owe thanks to Loredana Afanasiev, Graduate School of Logic, University of Amsterdam; Teun Boekhout, Eiko Kuramae, Vincent Robert, Fungal Biodiversity Center, Royal Netherlands Academy of Sciences; Marc Klein Wolt, Thomas Maccarone, Astronomical Institute “Anton Pannekoek”, University of Amsterdam; Evgeny Verbitskiy, Philips Research; Steven de Rooij, Ronald de Wolf, CWI; the referees and the editors, for suggestions, comments, help with experiments, and data; Jorma Rissanen and Boris Ryabko for useful discussions, Tzu-Kuo Huang for pointing out some typos and simplifications, and Teemu Roos and Henri Tirry for implementing a visualization of the clustering process.

Chapter 7

Automatic Meaning Discovery Using Google

The NCD investigations of the previous chapters focused on using data compressors to compress data in files. This chapter deals with an entirely different sort of analysis that is not performed on files but rather on *search terms* for the Google web search engine. By using well-known connections between code-lengths and probabilities, we apply the NCD theory to Google's search engine index, providing insight into the subjective relationships enjoyed among a group of words or phrases. The Google Simple Object Access Protocol is used to connect it with the CompLearn system. Remarkably, the system does not use the contents of web pages directly, but instead only uses the estimated results count indicator from the Google search engine to make a probabilistic model of the web. This model is based on sampling each search term in a group as well as all pairs in order to find structure in their co-occurrence. Before explaining the method in detail the reader is invited to have a look at the experiment in Figure 7.1 involving the names of politicians. The tree shows the subjective relationships among several European Commission members. After giving a general introduction to the method, we introduce some relevant background material in Section 7.1.2. We explain the formula that connects NCD to Google in Section 7.3. We provide a sketch of one possible theoretical breakdown concerning the surprising robustness of the results and consequent Google-based distance metric. We prove a certain sort of universality property for this metric. In Section 7.4, we present a variety of experiments demonstrating the sorts of results that may be obtained. We demonstrate positive correlations, evidencing an underlying semantic structure, in both numerical symbol notations and number-name words in a variety of natural languages and contexts. Next, we demonstrate the ability to distinguish between colors and numbers, and to distinguish between 17th century Dutch painters; the ability to understand electrical terms, religious terms, and emergency incidents; we conduct a massive experiment in understanding WordNet categories; and finally we demonstrate the ability to do a simple automatic English-Spanish vocabulary acquisition.

7.1 Introduction

Objects can be given literally, like the literal four-letter genome of a mouse, or the literal text of *War and Peace* by Tolstoy. For simplicity we take it that all meaning of the object is represented

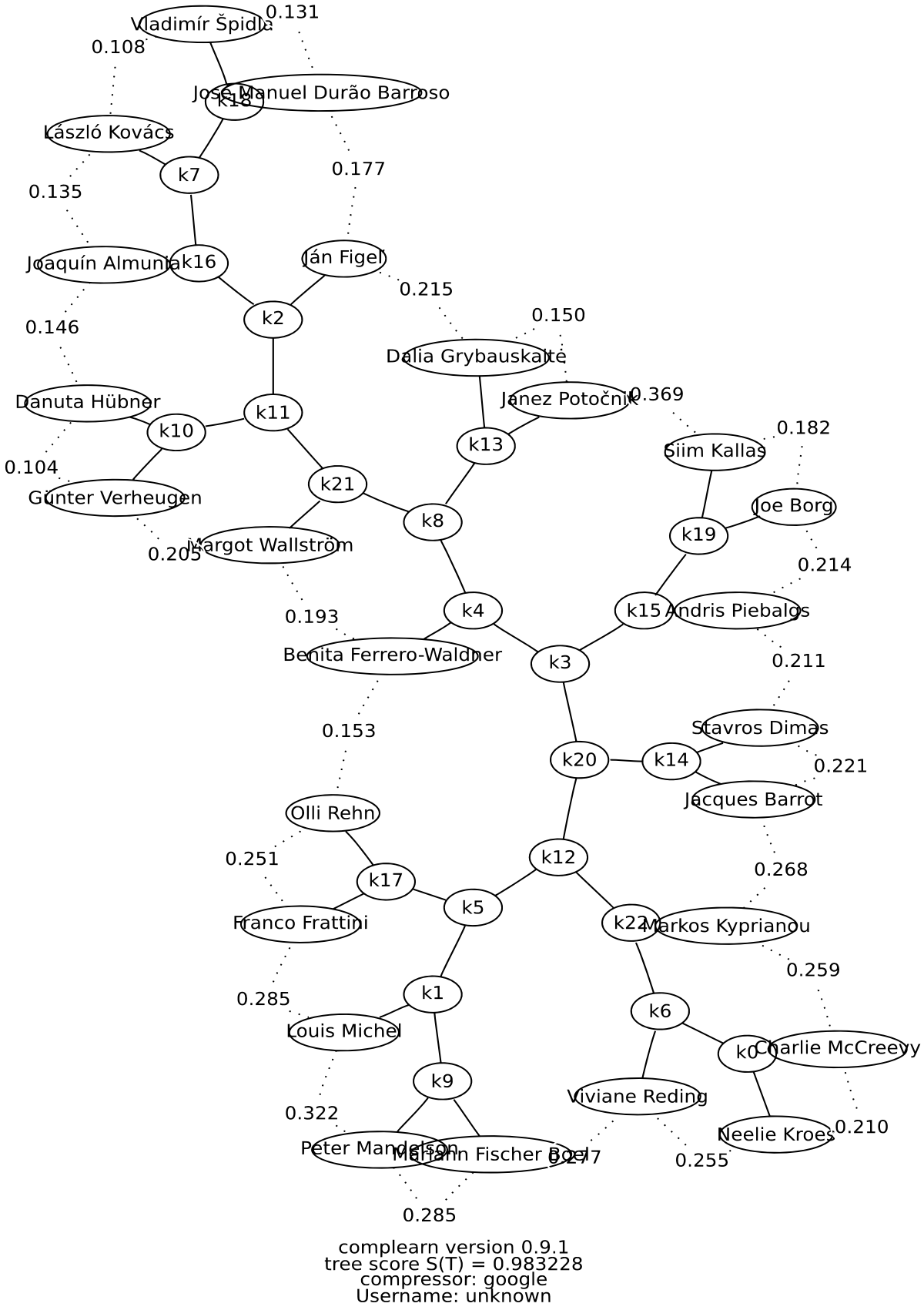

Figure 7.1: European Parliament members

by the literal object itself. Objects can also be given by name, like "the four-letter genome of a mouse," or "the text of *War and Peace* by Tolstoy." There are also objects that cannot be given literally, but only by name, and that acquire their meaning from their contexts in background common knowledge in humankind, like "home" or "red." To make computers more intelligent one would like to represent meaning in computer digestible form. Long-term and labor-intensive efforts like the *Cyc* project [71] and the *WordNet* project [37] try to establish semantic relations between common objects, or, more precisely, *names* for those objects. The idea is to create a semantic web of such vast proportions that rudimentary intelligence, and knowledge about the real world, spontaneously emerge. This comes at the great cost of designing structures capable of manipulating knowledge, and entering high quality contents in these structures by knowledgeable human experts. While the efforts are long-running and large scale, the overall information entered is minute compared to what is available on the world-wide-web.

The rise of the world-wide-web has enticed millions of users to type in trillions of characters to create billions of web pages of on average low quality contents. The sheer mass of the information available about almost every conceivable topic makes it likely that extremes will cancel and the majority or average is meaningful in a low-quality approximate sense. We devise a general method to tap the amorphous low-grade knowledge available for free on the world-wide-web, typed in by local users aiming at personal gratification of diverse objectives, and yet globally achieving what is effectively the largest semantic electronic database in the world. Moreover, this database is available for all by using any search engine that can return aggregate page-count estimates for a large range of search queries, like Google.

Previously, we and others developed a compression-based method to establish a universal similarity metric among objects given as finite binary strings [9, 75, 76, 26, 25, 22], which was widely reported [85, 88, 35]; some of these experiments are shown in chapters 5 and 7. Such objects can be genomes, music pieces in MIDI format, computer programs in Ruby or C, pictures in simple bitmap formats, or time sequences such as heart rhythm data. This method is feature-free in the sense that it does not analyze the files looking for particular features; rather it analyzes all features simultaneously and determines the similarity between every pair of objects according to the most dominant shared feature. The crucial point is that the method analyzes the objects themselves. This precludes comparison of abstract notions or other objects that do not lend themselves to direct analysis, like emotions, colors, Socrates, Plato, Mike Bonanno and Albert Einstein. While the previous method that compares the objects themselves is particularly suited to obtain knowledge about the similarity of objects themselves, irrespective of common beliefs about such similarities, here we develop a method that uses only the name of an object and obtains knowledge about the similarity of objects by tapping available information generated by multitudes of web users. Here we are reminded of the words of D.H. Rumsfeld [96]

> "A trained ape can know an awful lot
> Of what is going on in this world,
> Just by punching on his mouse
> For a relatively modest cost!"

This is useful to extract knowledge from a given corpus of knowledge, in this case the Google database, but not to obtain true facts that are not common knowledge in that database. For

example, common viewpoints on the creation myths in different religions may be extracted by the Googling method, but contentious questions of fact concerning the phylogeny of species can be better approached by using the genomes of these species, rather than by opinion.

7.1.1 Googling for Knowledge

Intuitively, the approach is as follows. The Google search engine indexes around ten billion pages on the web today. Each such page can be viewed as a set of index terms. While the theory we propose is rather intricate, the resulting method is simple enough. We give an example: At the time of doing the experiment, a Google search for "horse", returned 46,700,000 hits. The number of hits for the search term "rider" was 12,200,000. Searching for the pages where both "horse" and "rider" occur gave 2,630,000 hits, and Google indexed 8,058,044,651 web pages. Using these numbers in the main formula (7.3.3) we derive below, with $N = 8,058,044,651$, this yields a Normalized Google Distance between the terms "horse" and "rider" as follows:

$$\mathrm{NGD}(\mathit{horse},\mathit{rider}) \approx 0.443.$$

In the next part of this thesis we argue that the NGD is a normed semantic distance between the terms in question, usually (but not always, see below) inbetween 0 (identical) and 1 (unrelated), in the cognitive space invoked by the usage of the terms on the world-wide-web as filtered by Google. Because of the vastness and diversity of the web this may be taken as related to the current use of the terms in society. We did the same calculation when Google indexed only one-half of the number of pages: 4,285,199,774. It is instructive that the probabilities of the used search terms didn't change significantly over this doubling of pages, with number of hits for "horse" equal 23,700,000, for "rider" equal 6,270,000, and for "horse, rider" equal to 1,180,000. The $\mathrm{NGD}(\mathit{horse},\mathit{rider})$ we computed in that situation was ≈ 0.460. This is in line with our contention that the relative frequencies of web pages containing search terms gives objective information about the semantic relations between the search terms. If this is the case, then the Google probabilities of search terms and the computed NGD 's should stabilize (become scale invariant) with a growing Google database.

7.1.2 Related Work and Background NGD

It has been brought to our attention, that there is a great deal of work in both cognitive psychology [68], linguistics, and computer science, about using word (phrases) frequencies in text corpora to develop measures for word similarity or word association, partially surveyed in [112, 111], going back to at least [72]. These approaches are based on arguments and theories that are fundamentally different from the approach we develop, which is based on coding and compression, based on the theory of Kolmogorov complexity [79]. This allows to express and prove properties of absolute relations between objects that cannot even be expressed by other approaches. The NGD is the result of a new type of theory and as far as we know is not equivalent to any earlier measure. Nevertheless, in practice the resulting measure may still sometimes lead to similar results as existing methods. The current thesis is a next step in a decade of cumulative research in

this area, of which the main thread is [79, 9, 80, 76, 26, 25, 22] with [75, 10] using the related approach of [78].

7.1.3 Outline

Previously, we have outlined the classical information theoretic notions that have underpinned our approach, as well as the more novel ideas of Kolmogorov complexity, information distance, and compression-based similarity metric (Section 7.1.2). Here, we give a technical description of the Google distribution, the Normalized Google Distance, and the universality of these notions (Section 7.3), preceded by Subsection 7.2.1 pressing home the difference between literal object based similarity (as in compression based similarity), and context based similarity between objects that are not given literally but only in the form of names that acquire their meaning through contexts in databases of background information (as in Google based similarity). In Section 7.4 we present a plethora of clustering and various classification experiments to validate the universality, robustness, and accuracy of our proposal. A mass of experimental work, which for space reasons can not be reported here, is available [23]. In section 5.3.3 we explained some basics of the SVM approach we use in the classification experiments, where the Google similarity is used to extract feature vectors used by the kernel.

7.2 Extraction of Semantic Relations with Google

Every text corpus or particular user combined with a frequency extractor defines its own relative frequencies of words and phrases. In the world-wide-web and Google setting there are millions of users and text corpora, each with its own distribution. In the sequel, we show (and prove) that the Google distribution is universal for all the individual web users distributions.

The number of web pages currently indexed by Google is approaching 10^{10}. Every common search term occurs in millions of web pages. This number is so vast, and the number of web authors generating web pages is so enormous (and can be assumed to be a truly representative very large sample from humankind), that the probabilities of Google search terms, conceived as the frequencies of page counts returned by Google divided by the number of pages indexed by Google, may approximate the actual relative frequencies of those search terms as actually used in society. Based on this premise, the theory we develop in this chapter states that the relations represented by the Normalized Google Distance (7.3.3) approximately capture the assumed true semantic relations governing the search terms. The NGD formula (7.3.3) only uses the probabilities of search terms extracted from the text corpus in question. We use the world wide web and Google, but the same method may be used with other text corpora like the King James version of the Bible or the Oxford English Dictionary and frequency count extractors, or the world-wide-web again and Yahoo as frequency count extractor. In these cases one obtains a text corpus and frequency extractor biased semantics of the search terms. To obtain the true relative frequencies of words and phrases in society is a major problem in applied linguistic research. This requires analyzing representative random samples of sufficient sizes. The question of how to sample randomly and representatively is a continuous source of debate. Our contention that the web is such

a large and diverse text corpus, and Google such an able extractor, that the relative page counts approximate the true societal word- and phrases usage, starts to be supported by current real linguistics research [100].

Similarly, the NGD minorizes and incorporates all the different semantics of all the different users and text corpora on the web. It extracts as it were the semantics as used in the society (of all these web users) and not just the bias of any individual user or document. This is only possible using the web, since its sheer mass of users and documents with different intentions averages out to give the true semantic meaning as used in society. This is experimentally evidenced by the fact that when Google doubled its size the sample semantics of rider, horse stayed the same. Determining the NGD between two Google search terms does not involve analysis of particular features or specific background knowledge of the problem area. Instead, it analyzes all features automatically through Google searches of the most general background knowledge data base: the world-wide-web. (In Statistics "parameter-free estimation" means that the number of parameters analyzed is infinite or not a priori restricted. In our setting "feature-freeness" means that we analyze all features.)

7.2.1 Genesis of the Approach

We start from the observation that a compressor defines a code word length for every source word, namely, the number of bits in the compressed version of that source word. Viewing this code as a Shannon-Fano code, Section 2.7, it defines in its turn a probability mass function on the source words. Conversely, every probability mass function of the source words defines a Shannon-Fano code of the source words. Since this code is optimally compact in the sense of having expected code-word length equal to the entropy of the initial probability mass function, we take the viewpoint that a probability mass function is a compressor.

7.2.1. EXAMPLE. For example, the NID (Normalized Information Distance, Chapter 3, Section 3.3) uses the probability mass function $\mathbf{m}(x) = 2^{-K(x)}$, where K is the Kolmogorov complexity function, Chapter 2. This function is not computable, but it has the weaker property of being lower semi-computable: by approximating $K(x)$ from above by better and better compressors, we approximate $\mathbf{m}(x)$ from below. The distribution $\mathbf{m}(x)$ has the remarkable property that it dominates every lower semi-computable probability mass function $P(x)$ (and hence all computable ones) by assigning more probability to every finite binary string x than $P(x)$, up to a multiplicative constant $c_P > 0$ depending on P but not on x ($\mathbf{m}(x) \geq c_P P(x)$). We say that $\mathbf{m}(x)$ is *universal* for the enumeration of all lower semi-computable probability mass functions, [79], a terminology that is closely related to the "universality" of a universal Turing machine in the enumeration of all Turing machines. It is this property that allows us to show [76] that NID is the *most informative* distance (actually a metric) among a large family of (possibly non-metric) "admissible normalized information distances." But we cannot apply these same formal claims to the real-world NCD , except in a sense that is relativized on how well the compressor approximates the ultimate Kolmogorov complexity [22, 26, 25] and as shown in Section 3.3.

In essence, the choice of compressor brings a particular set of assumptions to bear upon an incoming data stream, and if these assumptions turn out to be accurate for the data in question,

then compression is achieved. This is the same as saying that the probability mass function defined by the compressor concentrates high probability on these data. If a pair of files share information in a way that matches the assumptions of a particular compressor, then we obtain a low NCD . Every compressor analyzes the string to be compressed by quantifying an associated family of features. A compressor such as `gzip` detects a class of features, for example matching substrings that are separated by no more than 32 kilobytes. Certain higher-order similarities are detected in the final Huffman coding phase. This explains how `gzip` is able to correctly cluster files generated by Bernoulli processes. A better compressor like `bzip2` detects substring matches across a wider window of 900 kilobytes, and detects an expanded set of higher-order features. Such compressors implicitly assume that the data has no global, structured, meaning. The compressor only looks for statistical biases, repetitions, and other biases in symmetrically defined local contexts, and cannot achieve compression even for very low-complexity meaningful strings like the digits of π. They assume the data source is at some level a simple stationary ergodic random information source which is by definition meaningless. The problem with this is clearly sketched by the great probabilist A.N. Kolmogorov [57, 58]: "The probabilistic approach is natural in the theory of information transmission over communication channels carrying 'bulk' information consisting of a large number of unrelated or weakly related messages obeying definite probabilistic laws. ... [it] can be convincingly applied to the information contained, for example, in a stream of congratulatory telegrams. But what real meaning is there, for example, in [applying this approach to] 'War and Peace'? · Or, on the other hand, must we assume that the individual scenes in this book form a random sequence with 'stochastic relations' that damp out quite rapidly over a distance of several pages?" The compressors apply no external knowledge to the compression, and so will not take advantage of the fact that `U` always follows `Q` in the English language, and instead must learn this fact anew for each separate file (or pair) despite the simple ubiquity of this rule. Thus the generality of common data compressors is also a liability, because the features which are extracted are by construction meaningless and devoid of relevance.

Yet, files exist in the real world, and the files that actually exist in stored format by and large carry a tremendous amount of structural, global, meaning; if they didn't then we would throw them away as useless. They do exhibit heavy biases in terms of the meaningless features, for instance in the way the letters `T` and `E` occur more frequently in English than `Z` or `Q`, but even this fails to capture the heart of the reason of the file's existence in the first place: because of its relevance to the rest of the world. But `gzip` does not know this reason; it is as if everywhere `gzip` looks it only finds a loaded die or biased coin, resolute in its objective and foolish consistency. In order to address this coding deficiency we choose an opposing strategy: instead of trying to apply no external knowledge at all during compression, we try to apply as much as we can from as many sources as possible simultaneously, and in so doing attempt to capture not the *literal* part but instead the *contextualized importance* of each string within a greater and all-inclusive whole.

Thus, instead of starting with a standard data compression program, we start from a probability mass function that reflects knowledge, and construct the corresponding Shannon-Fano code to convert probabilities to code word lengths, and apply the NCD formula. At this moment one database stands out as the pinnacle of computer accessible human knowledge and the most inclusive summary of statistical information: the Google search engine. There can be no doubt that

Google has already enabled science to accelerate tremendously and revolutionized the research process. It has dominated the attention of internet users for years, and has recently attracted substantial attention of many Wall Street investors, even reshaping their ideas of company financing. We have devised a way to interface the Google search engine to our NCD software to create a new type of pseudo-compressor based NCD , and call this new distance the Normalized Google Distance, or NGD . We have replaced the obstinate objectivity of classical compressors with an anthropomorphic subjectivity derived from the efforts of millions of people worldwide. Experiments suggest that this new distance shares some strengths and weaknesses in common with the humans that have helped create it: it is highly adaptable and nearly unrestricted in terms of domain, but at the same time is imprecise and fickle in its behavior. It is limited in that it doesn't analyze the literal objects like the NCD does, but instead uses names for those objects in the form of ASCII search terms or tuples of terms as inputs to extract the meaning of those objects from the total of information on the world-wide-web.

7.2.2. EXAMPLE. An example may help clarify the distinction between these two opposing paradigms. Consider the following sequence of letters:

```
U Q B
```

Assume that the next letter will be a vowel. What vowel would you guess is most likely, in the absence of any more specific information? One common assumption is that the samples are i.i.d. (identical, independently distributed), and given this assumption a good guess is `U` ; since it has already been shown once, chances are good that `U` is weighted heavily in the true generating distribution. In assuming i.i.d.-ness, we implicitly assume that there is some true underlying random information source. This assumption is often wrong in practice, even in an approximate sense. Changing the problem slightly, using English words as tokens instead of just letters, suppose we are given the sequence

```
the quick brown
```

Now we are told that the next word has three letters and does not end the sentence. We may imagine various three letter words that fit the bill. On an analysis as before, we ought to expect `the` to continue the sentence. The computer lists 535 English words of exactly three letters. We may use the `gzip` data compressor to compute the NCD for each possible completion like this: NCD (`the quick brown`,`cow`), NCD (`the quick brown`,`the`), and so on, for all of the 3-letter words. We may then sort the words in ascending order of NCD and this yields the following words in front, all with NCD of 0.61: `own, row, she, the` . There are other three letter words, like `hot`, that have NCD of 0.65, and many with larger distance. With such very small input strings, there are granularity effects associated with the compressor rounding to full bytes, which makes compression resolve only to the level of 8 bits at best. So as we might expect, `gzip` is using a sort of inference substantially similar to the sort that might lead the reader to guess `U` as a possible completion in the first example above.

Consider now what would happen if we were to use Google instead of `gzip` as the data compressor. Here we may change the input domain slightly; before, we operated on general strings,

binary or otherwise. With Google, we will restrict ourselves to ASCII words that can be used as search terms in the Google Search engine. With each search result, Google returns a count of matched pages. This can be thought to define a function mapping search terms (or combinations thereof) to page counts. This, in turn, can be thought to define a probability distribution, and we may use a Shannon Fano code to associate a code length with each page count. We divide the total number of pages returned on a query by the maximum that can be returned, when converting a page count to a probability; at the time of these experiments, the maximum was about 5,000,000,000. Computing the NGD of the phrase `the quick brown`, with each three-letter word that may continue the phrase (ignoring the constraint that that word immediately follows the word `brown`), we arrive at these first five most likely (candidate, NGD)-pairs (using the Google values at the time of writing):

```
fox: 0.532154812757325
vex: 0.561640307093518
jot: 0.579817813761161
hex: 0.589457285818459
pea: 0.604444512168738
```

As many typing students no doubt remember, a popular phrase to learn the alphabet is *The quick brown fox jumps over the lazy dog.* It is valuable because it uses every letter of the English alphabet.

Thus, we see a contrast between two styles of induction: The first type is the NCD based on a *literal* interpretation of the data: the data is the object itself. The second type is the NGD based on interpreting the data as a *name* for an abstract object which acquires its meaning from masses of *contexts* expressing a large body of common-sense knowledge. It may be said that the first case ignores the meaning of the message, whereas the second focuses on it.

7.3 Theory of Googling for Similarity

Every text corpus or particular user combined with a frequency extractor defines its own relative frequencies of words and phrases usage. In the world-wide-web and Google setting there are millions of users and text corpora, each with its own distribution. We will next show that the Google distribution is universal for all the individual web users distributions. The number of web pages currently indexed by Google is approaching 10^{10}. Every common search term occurs in millions of web pages. This number is so vast, and the number of web authors generating web pages is so enormous (and can be assumed to be a truly representative very large sample from humankind), that the probabilities of Google search terms, conceived as the frequencies of page counts returned by Google divided by the number of pages indexed by Google, approximate the actual relative frequencies of those search terms as actually used in society. Based on this premise, the theory we develop in this paper states that the relations represented by the Normalized Google Distance (7.3.3) approximately capture the assumed true semantic relations governing the search terms. The NGD formula (7.3.3) only uses the probabilities of search terms extracted from the

text corpus in question. We use the world-wide-web and Google, but the same method may be used with other text corpora like the King James version of the Bible or the Oxford English Dictionary and frequency count extractors, or the world-wide-web again and Yahoo as frequency count extractor. In these cases one obtains a text corpus and frequency extractor biased semantics of the search terms. To obtain the true relative frequencies of words and phrases in society is a major problem in applied linguistic research. This requires analyzing representative random samples of sufficient sizes. The question of how to sample randomly and representatively is a continuous source of debate. Our contention that the web is such a large and diverse text corpus, and Google such an able extractor, that the relative page counts approximate the true societal word- and phrases usage, starts to be supported by current real linguistics research [100].

7.3.1 The Google Distribution:

Let the set of singleton *Google search terms* be denoted by $\mathcal{S}$. In the sequel we use both singleton search terms and doubleton search terms $\{\{x,y\} : x,y \in \mathcal{S}\}$. Let the set of web pages indexed (possible of being returned) by Google be Ω. The cardinality of Ω is denoted by $M = |\Omega|$, and at the time of this writing $8 \cdot 10^9 \leq M \leq 9 \cdot 10^9$ (and presumably greater by the time of reading this). Assume that a priori all web pages are equi-probable, with the probability of being returned by Google being $1/M$. A subset of Ω is called an *event*. Every *search term* x usable by Google defines a *singleton Google event* $\mathbf{x} \subseteq \Omega$ of web pages that contain an occurrence of x and are returned by Google if we do a search for x. Let $L : \Omega \to [0,1]$ be the uniform mass probability function. The probability of an event $\mathbf{x}$ is $L(\mathbf{x}) = |\mathbf{x}|/M$. Similarly, the *doubleton Google event* $\mathbf{x} \bigcap \mathbf{y} \subseteq \Omega$ is the set of web pages returned by Google if we do a search for pages containing both search term x and search term y. The probability of this event is $L(\mathbf{x} \bigcap \mathbf{y}) = |\mathbf{x} \bigcap \mathbf{y}|/M$. We can also define the other Boolean combinations: $\neg\mathbf{x} = \Omega \backslash \mathbf{x}$ and $\mathbf{x} \bigcup \mathbf{y} = \neg(\neg\mathbf{x} \bigcap \neg\mathbf{y})$, each such event having a probability equal to its cardinality divided by M. If $\mathbf{e}$ is an event obtained from the basic events $\mathbf{x}, \mathbf{y}, \ldots$, corresponding to basic search terms $x, y, \ldots$, by finitely many applications of the Boolean operations, then the probability $L(\mathbf{e}) = |\mathbf{e}|/M$.

7.3.2 Google Semantics:

Google events capture in a particular sense all background knowledge about the search terms concerned available (to Google) on the web.

> The Google event $\mathbf{x}$, consisting of the set of all web pages containing one or more occurrences of the search term x, thus embodies, in every possible sense, all direct context in which x occurs on the web. This constitutes the Google semantics of the term.

7.3.1. REMARK. It is of course possible that parts of this direct contextual material link to other web pages in which x does not occur and thereby supply additional context. In our approach this indirect context is ignored. Nonetheless, indirect context may be important and future refinements of the method may take it into account.

7.3.3 The Google Code:

The event **x** consists of all possible direct knowledge on the web regarding x. Therefore, it is natural to consider code words for those events as coding this background knowledge. However, we cannot use the probability of the events directly to determine a prefix code, or, rather the underlying information content implied by the probability. The reason is that the events overlap and hence the summed probability exceeds 1. By the Kraft inequality [31] this prevents a corresponding set of code-word lengths. The solution is to normalize: We use the probability of the Google events to define a probability mass function over the set $\{\{x,y\} : x,y \in \mathcal{S}\}$ of Google search terms, both singleton and doubleton terms. There are $|\mathcal{S}|$ singleton terms, and $\binom{|\mathcal{S}|}{2}$ doubletons consisting of a pair of non-identical terms. Define

$$N = \sum_{\{x,y\}\subseteq\mathcal{S}} |\mathbf{x}\bigcap\mathbf{y}|,$$

counting each singleton set and each doubleton set (by definition unordered) once in the summation. Note that this means that for every pair $\{x,y\} \subseteq \mathcal{S}$, with $x \neq y$, the web pages $z \in \mathbf{x}\bigcap\mathbf{y}$ are counted three times: once in $\mathbf{x} = \mathbf{x}\bigcap\mathbf{x}$, once in $\mathbf{y} = \mathbf{y}\bigcap\mathbf{y}$, and once in $\mathbf{x}\bigcap\mathbf{y}$. Since every web page that is indexed by Google contains at least one occurrence of a search term, we have $N \geq M$. On the other hand, web pages contain on average not more than a certain constant α search terms. Therefore, $N \leq \alpha M$. Define

$$g(x) = g(x,x), \;\; g(x,y) = L(\mathbf{x}\bigcap\mathbf{y})M/N = |\mathbf{x}\bigcap\mathbf{y}|/N. \tag{7.3.1}$$

Then, $\sum_{\{x,y\}\subseteq\mathcal{S}} g(x,y) = 1$. This g-distribution changes over time, and between different samplings from the distribution. But let us imagine that g holds in the sense of an instantaneous snapshot. The real situation will be an approximation of this. Given the Google machinery, these are absolute probabilities which allow us to define the associated prefix code-word lengths (information contents) for both the singletons and the doubletons. The *Google code* G is defined by

$$G(x) = G(x,x), \;\; G(x,y) = \log 1/g(x,y). \tag{7.3.2}$$

7.3.4 The Google Similarity Distance:

In contrast to strings x where the complexity $C(x)$ represents the length of the compressed version of x using compressor C, for a search term x (just the name for an object rather than the object itself), the Google code of length $G(x)$ represents the shortest expected prefix-code word length of the associated Google event **x**. The expectation is taken over the Google distribution g. In this sense we can use the Google distribution as a compressor for the Google semantics associated with the search terms. The associated NCD , now called the *normalized Google distance (*NGD *)* is then defined by (7.3.3), and can be rewritten as the right-hand expression:

$$\begin{aligned} \mathrm{NGD}(x,y) &= \frac{G(x,y) - \min(G(x),G(y))}{\max(G(x),G(y))} \\ &= \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{\log N - \min\{\log f(x), \log f(y)\}}, \end{aligned} \tag{7.3.3}$$

where $f(x)$ denotes the number of pages containing x, and $f(x,y)$ denotes the number of pages containing both x and y, as reported by Google. This NGD is an approximation to the NID of (3.3.1) using the prefix code-word lengths (Google code) generated by the Google distribution as defining a compressor approximating the length of the Kolmogorov code, using the background knowledge on the web as viewed by Google as conditional information. In practice, use the page counts returned by Google for the frequencies, and we have to choose N. From the right-hand side term in (7.3.3) it is apparent that by increasing N we decrease the NGD , everything gets closer together, and by decreasing N we increase the NGD , everything gets further apart. Our experiments suggest that every reasonable (M or a value greater than any $f(x)$) value can be used as normalizing factor N, and our results seem in general insensitive to this choice. In our software, this parameter N can be adjusted as appropriate, and we often use M for N. The following are the main properties of the NGD (as long as we choose parameter $N \geq M$):

1. The *range* of the NGD is in between 0 and ∞ (sometimes slightly negative if the Google counts are untrustworthy) and state $f(x,y) > \max\{f(x), f(y)\}$:
   (a) If $x = y$ or if $x \neq y$ but frequency $f(x) = f(y) = f(x,y) > 0$, then $\mathrm{NGD}(x,y) = 0$. That is, the semantics of x and y in the Google sense is the same.
   (b) If frequency $f(x) = 0$, then for every search term y we have $f(x,y) = 0$, and the $\mathrm{NGD}(x,y) = \infty/\infty$, which we take to be 1 by definition.
2. The NGD is always nonnegative and $\mathrm{NGD}(x,x) = 0$ for every x. For every pair x,y we have $\mathrm{NGD}(x,y) = \mathrm{NGD}(y,x)$: it is symmetric. However, the NGD is *not a metric*: it does not satisfy $\mathrm{NGD}(x,y) > 0$ for every $x \neq y$. As before, let $\mathbf{x}$ denote the set of web pages containing one or more occurrences of x. For example, choose $x \neq y$ with $\mathbf{x} = \mathbf{y}$. Then, $f(x) = f(y) = f(x,y)$ and $\mathrm{NGD}(x,y) = 0$. Nor does the NGD satisfy the triangle inequality $\mathrm{NGD}(x,y) \leq \mathrm{NGD}(x,z) + \mathrm{NGD}(z,y)$ for all x,y,z. For example, choose $\mathbf{z} = \mathbf{x} \bigcup \mathbf{y}$, $\mathbf{x} \bigcap \mathbf{y} = \emptyset$, $\mathbf{x} = \mathbf{x} \bigcap \mathbf{z}$, $\mathbf{y} = \mathbf{y} \bigcap \mathbf{z}$, and $|\mathbf{x}| = |\mathbf{y}| = \sqrt{N}$. Then, $f(x) = f(y) = f(x,z) = f(y,z) = \sqrt{N}$, $f(z) = 2\sqrt{N}$, and $f(x,y) = 0$. This yields $\mathrm{NGD}(x,y) = \infty$ and $\mathrm{NGD}(x,z) = \mathrm{NGD}(z,y) = 2/\log N$, which violates the triangle inequality for all N.
3. The NGD is *scale-invariant* in the following sense: Assume that when the number N of pages indexed by Google (accounting for the multiplicity of different search terms per page) grows, the number of pages containing a given search term goes to a fixed fraction of N, and so does the number of pages containing a given conjunction of search terms. This means that if N doubles, then so do the f-frequencies. For the NGD to give us an objective semantic relation between search terms, it needs to become stable when the number N grows unboundedly.

7.3.5 Universality of Google Distribution:

A central notion in the application of compression to learning is the notion of "universal distribution," see [79]. Consider an effective enumeration $\mathcal{P} = p_1, p_2, \ldots$ of probability mass functions with domain $\mathcal{S}$. The list $\mathcal{P}$ can be finite or countably infinite.

7.3.2. DEFINITION. A probability mass function p_u occurring in $\mathcal{P}$ is *universal* for $\mathcal{P}$, if for every p_i in $\mathcal{P}$ there is a constant $c_i > 0$ and $\sum_{i \neq u} c_i \geq 1$, such that for every $x \in \mathcal{S}$ we have $p_u(x) \geq c_i \cdot p_i(x)$. Here c_i may depend on the indexes u, i, but not on the functional mappings of the elements of list $\mathcal{P}$ nor on x.

If p_u is universal for $\mathcal{P}$, then it immediately follows that for every p_i in $\mathcal{P}$, the prefix code-word length for source word x, see [31], associated with p_u, minorizes the prefix code-word length associated with p_i, by satisfying $\log 1/p_u(x) \leq \log 1/p_i(x) + \log 1/c_i$, for every $x \in \mathcal{S}$.

In the following we consider partitions of the set of web pages, each subset in the partition together with a probability mass function of search terms. For example, we may consider the list $\mathcal{A} = 1, 2, \ldots, a$ of *web authors producing pages* on the web, and consider the set of web pages produced by each web author, or some other partition. "Web author" is just a metaphor we use for convenience. Let web author i of the list $\mathcal{A}$ produce the set of web pages Ω_i and denote $M_i = |\Omega_i|$. We identify a web author i with the set of web pages Ω_i he produces. Since we have no knowledge of the set of web authors, we consider every possible partition of Ω into one or more equivalence classes, $\Omega = \Omega_1 \bigcup \cdots \bigcup \Omega_a$, $\Omega_i \bigcap \Omega_j = \emptyset$ $(1 \leq i \neq j \leq a \leq |\Omega|)$, as defining a realizable set of web authors $\mathcal{A} = 1, \ldots, a$.

Consider a partition of Ω into $\Omega_1, \ldots, \Omega_a$. A search term x usable by Google defines an event $\mathbf{x}_i \subseteq \Omega_i$ of web pages produced by web author i that contain search term x. Similarly, $\mathbf{x}_i \bigcap \mathbf{y}_i$ is the set of web pages produced by i that is returned by Google searching for pages containing both search term x and search term y. Let

$$N_i = \sum_{\{x,y\} \subseteq \mathcal{S}} |\mathbf{x}_i \bigcap \mathbf{y}_i|.$$

Note that there is an $\alpha_i \geq 1$ such that $M_i \leq N_i \leq \alpha_i M_i$. For every search term $x \in \mathcal{S}$ define a probability mass function g_i, the *individual web author's Google distribution*, on the sample space $\{\{x,y\} : x,y \in S\}$ by

$$g_i(x) = g_i(x,x), \;\; g_i(x,y) = |\mathbf{x}_i \bigcap \mathbf{y}_i|/N_i. \tag{7.3.4}$$

Then, $\sum_{\{x,y\} \subseteq \mathcal{S}} g_i(x,y) = 1$.

7.3.3. THEOREM. *Let $\Omega_1, \ldots, \Omega_a$ be any partition of Ω into subsets (web authors), and let $g_1, \ldots, g_a$ be the corresponding individual Google distributions. Then the Google distribution g is universal for the enumeration $g, g_1, \ldots, g_a$.*

PROOF. We can express the overall Google distribution in terms of the individual web author's distributions:

$$g(x,y) = \sum_{i \in \mathcal{A}} \frac{N_i}{N} g_i(x,y).$$

Consequently, $g(x,y) \geq (N_i/N) g_i(x,y)$. Since also $g(x,y) \geq g(x,y)$, we have shown that $g(x,y)$ is universal for the family $g, g_1, \ldots, g_a$ of individual web author's google distributions, according to Definition 7.3.2. □

7.3.4. REMARK. Let us show that, for example, the uniform distribution $L(x) = 1/s$ ($s = |\mathcal{S}|$) over the search terms $x \in \mathcal{S}$ is not universal, for $s > 2$. By the requirement $\sum c_i \geq 1$, the sum taken over the number a of web authors in the list $\mathcal{A}$, there is an i such that $c_i \geq 1/a$. Taking the uniform distribution on say s search terms assigns probability $1/s$ to each of them. By the definition of universality of a probability mass function for the list of individual Google probability mass functions g_i, we can choose the function g_i freely (as long as $a \geq 2$, and there is another function g_j to exchange probabilities of search terms with). So choose some search term x and set $g_i(x) = 1$, and $g_i(y) = 0$ for all search terms $y \neq x$. Then, we obtain $g(x) = 1/s \geq c_i g_i(x) \geq 1/a$. This yields the required contradiction for $s > a \geq 2$.

7.3.6 Universality of Normalized Google Distance:

Every individual web author produces both an individual Google distribution g_i, and an *individual prefix code-word length* G_i associated with g_i (see [31] for this code) for the search terms.

7.3.5. DEFINITION. The associated *individual normalized Google distance* NGD_i of web author i is defined according to (7.3.3), with G_i substituted for G.

These Google distances NGD_i can be viewed as the individual semantic distances according to the bias of web author i. These individual semantics are subsumed in the general Google semantics in the following sense: The normalized Google distance is *universal* for the family of individual normalized Google distances, in the sense that it is as about as small as the least individual normalized Google distance, with high probability. Hence the Google semantics as evoked by all of the web society in a certain sense captures the biases or knowledge of the individual web authors. In Theorem 7.3.8 we show that, for every $k \geq 1$, the inequality

$$\mathrm{NGD}(x,y) < \beta\, \mathrm{NGD}_i(x,y) + \gamma, \tag{7.3.5}$$

with

$$\begin{aligned}
\beta &= \frac{\max\{G_i(x), G_i(y)\}}{\max\{G(x), G(y)\}} \leq 1 + \frac{\log(2k)}{\max\{G(x), G(y)\}} \\
\gamma &= \frac{\min\{G_i(x), G_i(y)\} - \min\{G(x), G(y)\} + \log N/N_i}{\max\{G(x), G(y)\}} \\
&\leq \frac{\log(2kN/N_i)}{\max\{G(x), G(y)\}},
\end{aligned}$$

is satisfied with g_i-probability going to 1 with growing k.

7.3.6. REMARK. To interpret (7.3.5), we observe that in case $G(x)$ and $G(y)$ are large with respect to $\log k$, then $\beta \approx 1$. If moreover $\log N/N_i$ is large with respect to $\log k$, then approximately $\gamma \leq (\log N/N_i)/\max\{G(x), G(y)\}$. Let us estimate γ for this case under reasonable assumptions. Without loss of generality assume $G(x) \geq G(y)$. If $f(x) = |\mathbf{x}|$, the number of pages returned on

query x, then $G(x) = \log(N/f(x))$. Thus, approximately $\gamma \leq (\log N/N_i)/(\log N/f(x))$. The uniform expectation of N_i is $N/|\mathcal{A}|$, and N divided by that expectation of N_i equals $|\mathcal{A}|$, the number of web authors producing web pages. The uniform expectation of $f(x)$ is $N/|\mathcal{S}|$, and N divided by that expectation of $f(x)$ equals $|\mathcal{S}|$, the number of Google search terms we use. Thus, approximately, $\gamma \leq (\log|\mathcal{A}|)/(\log|\mathcal{S}|)$, and the more the number of search terms exceeds the number of web authors, the more γ goes to 0 in expectation.

7.3.7. REMARK. To understand (7.3.5), we may consider the codelengths involved as the Google database changes over time. It is reasonable to expect that both the total number of pages as well as the total number of search terms in the Google database will continue to grow for some time. In this period, the sum total probability mass will be carved up into increasingly smaller pieces for more and more search terms. The maximum singleton and doubleton codelengths within the Google database will grow. But the universality property of the Google distribution implies that the Google distribution's code length for almost all particular search terms will only exceed the best codelength among any of the individual web authors as in (7.3.5). The size of this gap will grow more slowly than the codelength for any particular search term over time. Thus, the coding space that is suboptimal in the Google distribution's code is an ever-smaller piece (in terms of proportion) of the total coding space.

7.3.8. THEOREM. *For every web author $i \in \mathcal{A}$, the g_i-probability concentrated on the pairs of search terms for which* (7.3.5) *holds is at least* $(1-1/k)^2$.

PROOF. The prefix code-word lengths G_i associated with g_i satisfy $G(x) \leq G_i(x) + \log N/N_i$ and $G(x,y) \leq G_i(x,y) + \log N/N_i$. Substituting $G(x,y)$ by $G_i(x,y) + \log N/N_i$ in the middle term of (7.3.3), we obtain

$$\mathrm{NGD}(x,y) \leq \frac{G_i(x,y) - \min\{G(x),G(y)\} + \log N/N_i}{\max\{G(x),G(y)\}}. \tag{7.3.6}$$

Markov's Inequality says the following: Let p be any probability mass function; let f be any nonnegative function with p-expected value $\mathbf{E} = \sum_i p(i) f(i) < \infty$. For $\mathbf{E} > 0$ we have $\sum_i \{p(i) : f(i)/\mathbf{E} > k\} < 1/k$.

Fix web author $i \in \mathcal{A}$. We consider the conditional probability mass functions $g'(x) = g(x|x \in \mathcal{S})$ and $g'_i(x) = g_i(x|x \in \mathcal{S})$ over singleton search terms in $\mathcal{S}$ (no doubletons): The g'_i-expected value of $g'(x)/g'_i(x)$ is

$$\sum_x g'_i(x) \frac{g'(x)}{g'_i(x)} \leq 1,$$

since g' is a probability mass function summing to ≤ 1. Then, by Markov's Inequality

$$\sum_x \{g'_i(x) : g'(x)/g'_i(x) > k\} < \frac{1}{k} \tag{7.3.7}$$

Since the probability of an event of a doubleton set of search terms is not greater than that of an event based on either of the constituent search terms, and the probability of a singleton event

conditioned on it being a singleton event is at least as large as the unconditional probability of that event, $2g(x) \geq g'(x) \geq g(x)$ and $2g_i(x) \geq g'_i(x) \geq g_i(x)$. If $g(x) > 2kg_i(x)$, then $g'(x)/g'_i(x) > k$ and the search terms x satisfy the condition of (7.3.7). Moreover, the probabilities satisfy $g_i(x) \leq g'_i(x)$. Together, it follows from (7.3.7) that $\sum_x \{g_i(x) : g(x)/(2g_i(x)) > k\} < \frac{1}{k}$ and therefore

$$\sum_x \{g_i(x) : g(x) \leq 2kg_i(x)\} > 1 - \frac{1}{k}.$$

For the x's with $g(x) \leq 2kg_i(x)$ we have $G_i(x) \leq G(x) + \log(2k)$. Substitute $G_i(x) - \log(2k)$ for $G(x)$ (there is g_i-probability $\geq 1 - 1/k$ that $G_i(x) - \log(2k) \leq G(x)$) and $G_i(y) - \log(2k) \leq G(y)$ in (7.3.6), both in the min-term in the numerator, and in the max-term in the denominator. Noting that the two g_i-probabilities $(1 - 1/k)$ are independent, the total g_i-probability that both substitutions are justified is at least $(1 - 1/k)^2$. □

Therefore, the Google normalized distance minorizes every normalized compression distance based on a particular user's generated probabilities of search terms, with high probability up to an error term that in typical cases is ignorable.

7.4 Introduction to Experiments

7.4.1 Google Frequencies and Meaning

In our first experiment, we seek to verify that Google page counts capture something more than meaningless noise. For simplicity, we do not use NGD here, but instead look at just the Google probabilities of small integers in several formats. The first format we use is just the standard numeric representation using digits, for example "43". The next format we use is the number spelled out in English, as in "forty three". Then we use the number spelled in Spanish, as in "cuarenta y tres". Finally, we use the number as digits again, but now paired with the fixed and arbitrary search term *green* . In each of these examples, we compute the probability of search term x as $f(x)/M$. Here, $f(x)$ represents the count of webpages containing search term x. We plotted $\log(f(x)/M)$ against x in Figure 7.2 for x runs from 1 to 120. Notice that numbers such as even multiples of ten and five stand out in every representation in the sense that they have much higher frequency of occurrence. We can treat only low integers this way: integers of the order 10^{23} mostly do not occur since there are not web pages enough to represent a noticeable fraction of them (but Avogadro's number 6.022×10^{23} occurs with high frequency both in letters and digits).

Visual inspection of the plot gives clear evidence that there is a positive correlation between every pair of formats. We can therefore assume that that there is some underlying structure that is independent of the language chosen, and indeed the same structure appears even in the restricted case of just those webpages that contain the search term *green* .

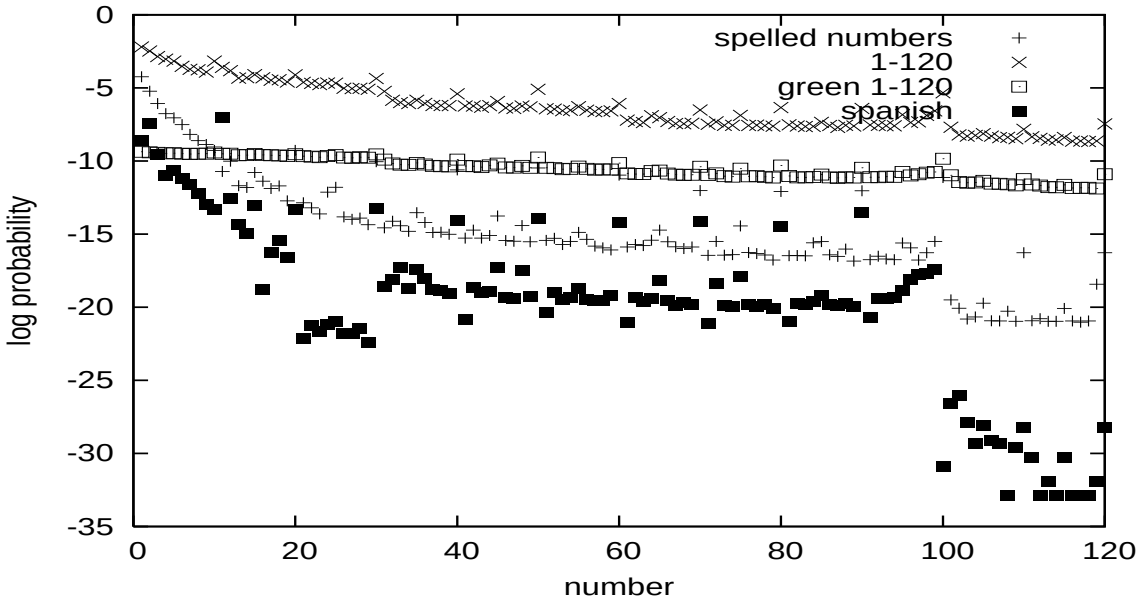

Figure 7.2: Numbers versus log probability (pagecount / M) in a variety of languages and formats.

7.4.2 Some Implementation Details

Before explaining our primary NGD results, a few implementation details should be clarified. When entering searches in Google, a rich syntax is available whereby searches may be precisely constrained, see [50]. We use two important features. If you enter the term `every generation` in Google, it counts precisely the number of pages that contain both the word *every* and the word *generation*, but not necessarily consecutively like `every generation`. If you instead enter `"every generation"`, then this tells Google that both words must appear consecutively. Another feature that is important is the `+` modifier. Google ignores common words and characters such as "where" and "how", as well as certain single digits and single letters. Prepending a `+` before a searchterm indicates that every result must include the following term, even if it is a term otherwise ignored by Google. Experiments show that `every generation` and `+"every" +"generation"` give slightly different results, say 17,800,000 against 17,900,000. Some other experiments show, that whatever the Google manual says, the form `horse rider` is slightly sensitive to adding spaces, while `+"horse" +"rider"` is not. Therefore, we only use the latter form. Our translation from a tuple of search terms into a Google search query proceeds in three steps: First we put double-quotes around every search term in the tuple. Next, we prepend a `+` before every term. Finally, we join together each of the resultant strings with a single space. For example, when using the search terms "horse" and "rider", it is converted to the Google search query `+"horse" +"rider"`.

Another detail concerns avoiding taking the logarithm of 0. Although our theory conveniently allows for ∞ in this case, our implementation makes a simplifying compromise. When returning $f(x)$ for a given search, we have two cases. If the number of pages returned is non-zero, we

return twice this amount. If the pages is equal to 0, we do not return 0, but instead return 1. Thus, even though a page does not exist in the Google index, we credit it half the probability of the smallest pages that do exist in Google. This greatly simplifies our implementation and seems not to result in much distortion in the cases we have investigated.

7.4.3 Three Applications of the Google Method

In this chapter we give three applications of the Google method: unsupervised learning in the form of hierarchical clustering, supervised learning using support vector machines, and matching using correlation. For the hierarchical clustering method we refer to Section 4.1. and the correlation method is well known. For the supervised learning, several techniques are available. For the SVM method used in this thesis, we refer to the excellent exposition [16], and give a brief summary in Appendix 5.3.3.

7.5 Hierarchical Clustering

For these examples, we used our software tool available from http://complearn.sourceforge.net/, the same tool that has been used in other chapters to construct trees representing hierarchical clusters of objects in an unsupervised way. However, now we use the normalized Google distance (NGD) instead of the normalized compression distance (NCD). Recapitulating, the method works by first calculating a distance matrix using NGD among all pairs of terms in the input list. Then it calculates a best-matching unrooted ternary tree using a novel quartet-method style heuristic based on randomized hill-climbing using a new fitness objective function optimizing the summed costs of all quartet topologies embedded in candidate trees.

7.5.1 Colors and Numbers

In the first example, the objects to be clustered are search terms consisting of the names of colors, numbers, and some tricky words. The program automatically organized the colors towards one side of the tree and the numbers towards the other, Figure 7.3. It arranges the terms which have as only meaning a color or a number, and nothing else, on the farthest reach of the color side and the number side, respectively. It puts the more general terms black and white, and zero, one, and two, towards the center, thus indicating their more ambiguous interpretation. Also, things which were not exactly colors or numbers are also put towards the center, like the word “small”. We may consider this an example of automatic ontology creation.

7.5.2 Dutch 17th Century Painters

In the example of Figure 7.4, the names of fifteen paintings by Steen, Rembrandt, and Bol were entered. The names of the associated painters were not included in the input, however they were added to the tree display afterword to demonstrate the separation according to painters. This type of problem has attracted a great deal of attention [97]. A more classical solution is offered in

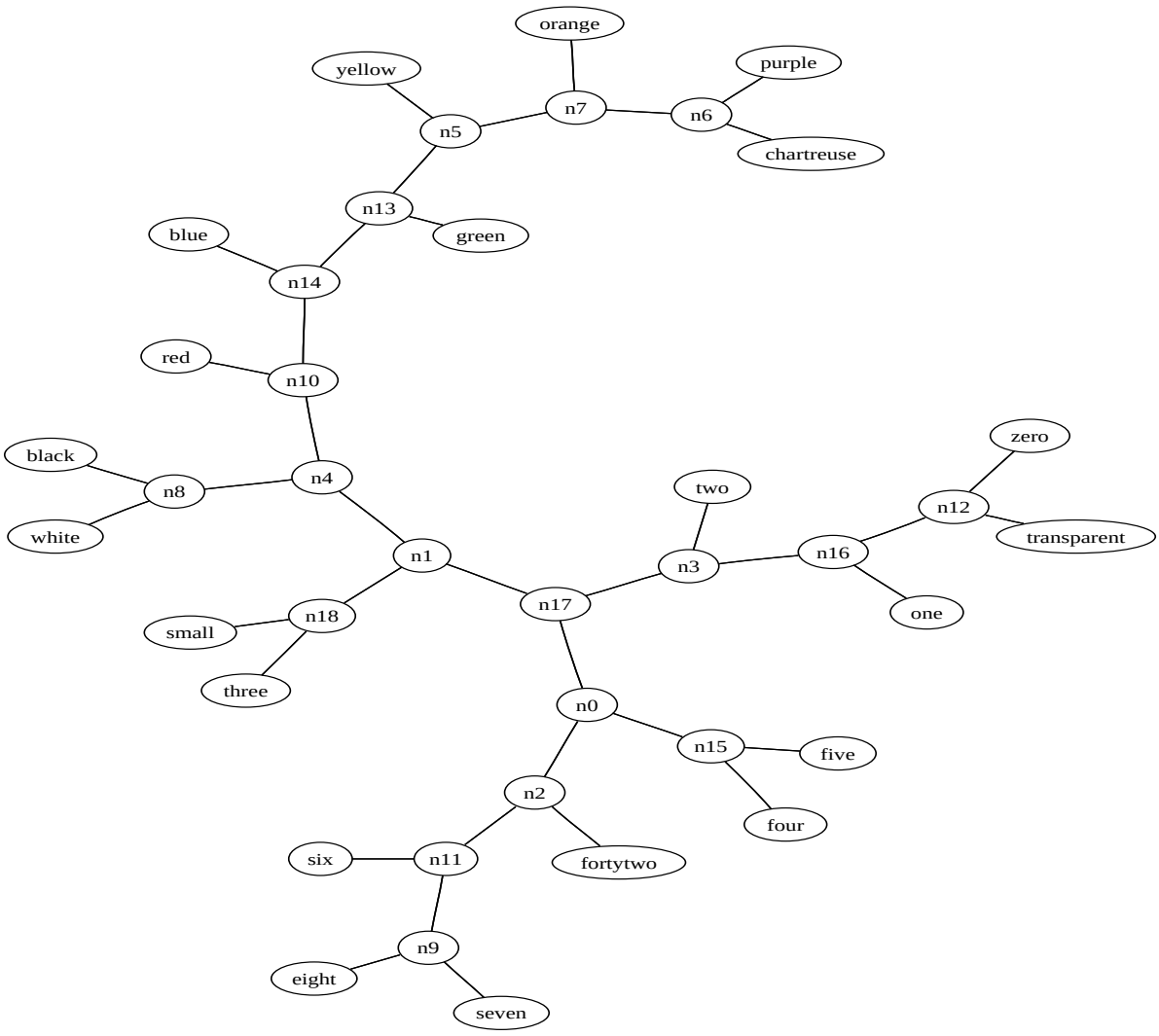

Figure 7.3: Colors and numbers arranged into a tree using NGD .

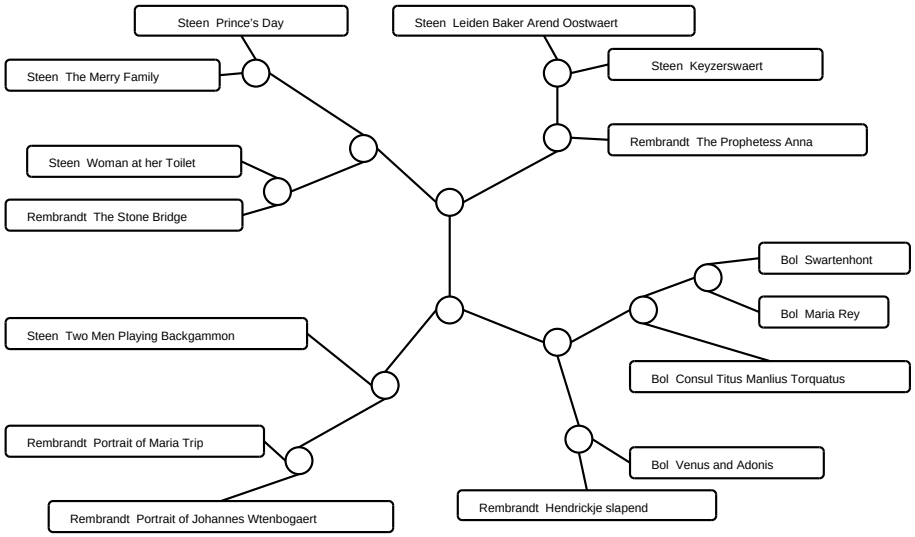

Figure 7.4: Fifteen paintings tree by three different painters arranged into a tree hierarchical clustering. In the experiment, only painting title names were used; the painter prefix shown in the diagram above was added afterwords as annotation to assist in interpretation. The painters and paintings used follow. **Rembrandt van Rijn** : *Hendrickje slapend; Portrait of Maria Trip; Portrait of Johannes Wtenbogaert ; The Stone Bridge ; The Prophetess Anna* ; **Jan Steen** : *Leiden Baker Arend Oostwaert ; Keyzerswaert ; Two Men Playing Backgammon ; Woman at her Toilet ; Prince's Day ; The Merry Family* ; **Ferdinand Bol** : *Maria Rey ; Consul Titus Manlius Torquatus ; Swartenhont ; Venus and Adonis* .

[1], where a domain-specific database is used for similar ends. The present automatic oblivious method obtains results that compare favorably with the latter feature-driven method.

7.5.3 Chinese Names

In the example of Figure 7.5, several Chinese names were entered. The tree shows the separation according to concepts like regions, political parties, people, etc. See Figure 7.6 for English translations of these characters. This figure also shows a feature of the CompLearn system that has not been encountered before: the CompLearn system can draw dotted lines with numbers inbetween each adjacent node along the perimeter of the tree. These numbers represent the NCD distance between adjacent nodes in the final (ordered tree) output of the CompLearn system. The tree is presented in such a way that the sum of these values in the entire ring is minimized. This generally results in trees that makes the most sense upon initial visual inspection, converting an unordered binary tree to an ordered one. This feature allows for a quick visual inspection around the edges to determine the major groupings and divisions among coarse structured problems. It grew out of an idea originally suggested by Lloyd Rutledge at CWI [97].

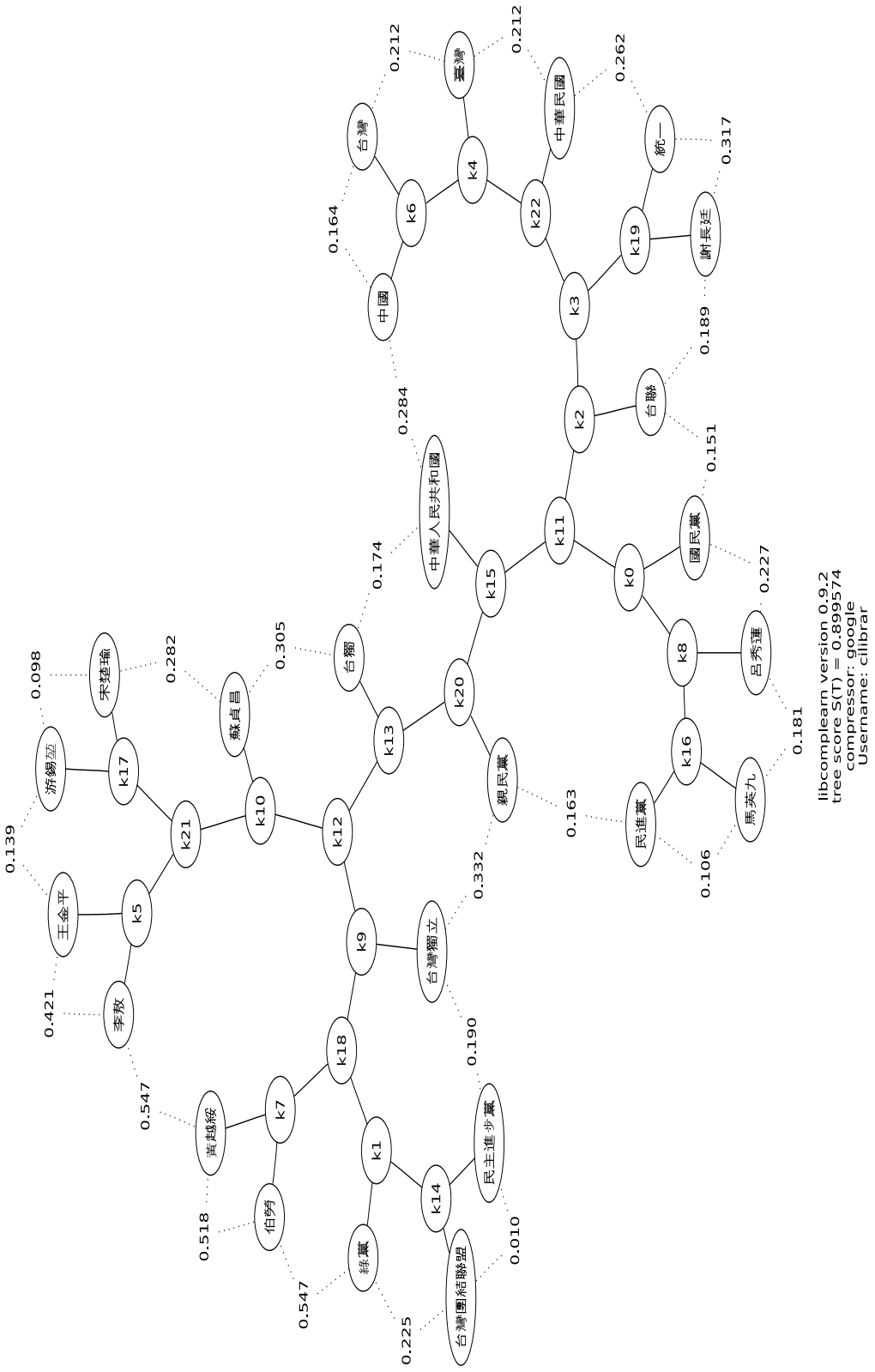

Figure 7.5: Several people's names, political parties, regions, and other Chinese names.

中國 China
中華人民共和國 People's Republic of China
中華民國 Republic of China
伯勞 shrike (bird) [outgroup]
台灣 Taiwan (with simplified character "tai")
台灣團結聯盟 Taiwan Solidarity Union [Taiwanese political party]
台灣獨立 Taiwan independence
台獨 (abbreviation of the above)
台聯 (abbreviation of Taiwan Solidarity Union)
呂秀蓮 Annette Lu
國民黨 Kuomintang
宋楚瑜 James Soong
李敖 Li Ao
民主進步黨 Democratic Progressive Party
民進黨 (abbreviation of the above)
游錫堃 Yu Shyi-kun
王金平 Wang Jin-pyng
統一 unification [Chinese unification]
綠黨 Green Party
臺灣 Taiwan (with traditional character "tai")
蘇貞昌 Su Tseng-chang
親民黨 People First Party [political party in Taiwan]
謝長廷 Frank Hsieh
馬英九 Ma Ying-jeou
黃越綏 a presidential advisor and 2008 presidential hopeful

Figure 7.6: English Translation of Chinese Names

Training Data

Positive Training	(22 cases)			
avalanche	bomb threat	broken leg	burglary	car collision
death threat	fire	flood	gas leak	heart attack
hurricane	landslide	murder	overdose	pneumonia
rape	roof collapse	sinking ship	stroke	tornado
train wreck	trapped miners			
Negative Training	(25 cases)			
arthritis	broken dishwasher	broken toe	cat in tree	contempt of court
dandruff	delayed train	dizziness	drunkenness	enumeration
flat tire	frog	headache	leaky faucet	littering
missing dog	paper cut	practical joke	rain	roof leak
sore throat	sunset	truancy	vagrancy	vulgarity
Anchors	(6 dimensions)			
crime	happy	help	safe	urgent
wash				

Testing Results

	Positive tests	Negative tests
Positive Predictions	assault, coma, electrocution, heat stroke, homicide, looting, meningitis, robbery, suicide	menopause, prank call, pregnancy, traffic jam
Negative Predictions	sprained ankle	acne, annoying sister, campfire, desk, mayday, meal

Accuracy

15/20 = 75.00%

Figure 7.7: Google-SVM learning of "emergencies."

7.6 SVM Learning

We augment the Google method by adding a trainable component of the learning system. This allows us to consider classification rather than clustering problems. Here we use the Support Vector Machine (SVM) as a trainable component. For a brief introduction to SVM's see Section 5.3.3. We use LIBSVM software for all of our SVM experiments.

The setting is a binary classification problem on examples represented by search terms. We require a human expert to provide a list of at least 40 *training words*, consisting of at least 20 positive examples and 20 negative examples, to illustrate the contemplated concept class. The expert also provides, say, six *anchor words* $a_1, \ldots, a_6$, of which half are in some way related to the concept under consideration. Then, we use the anchor words to convert each of the 40 training words $w_1, \ldots, w_{40}$ to 6-dimensional *training vectors* $\bar{v}_1, \ldots, \bar{v}_{40}$. The entry $v_{j,i}$ of $\bar{v}_j = (v_{j,1}, \ldots, v_{j,6})$ is defined as $v_{j,i} = \mathrm{NGD}(w_j, a_i)$ $(1 \leq j \leq 40, 1 \leq i \leq 6)$. The training vectors are then used to train an SVM to learn the concept, and then test words may be classified using the same anchors and trained SVM model. We present all positive examples as x-data (input data), paired with $y = 1$. We present all negative examples as x-data, paired with $y = -1$.

7.6.1 Emergencies

In the next example, Figure 7.7, we trained using a list of emergencies as positive examples, and a list of "almost emergencies" as negative examples. The figure is self-explanatory. The accuracy on the test set is 75%.

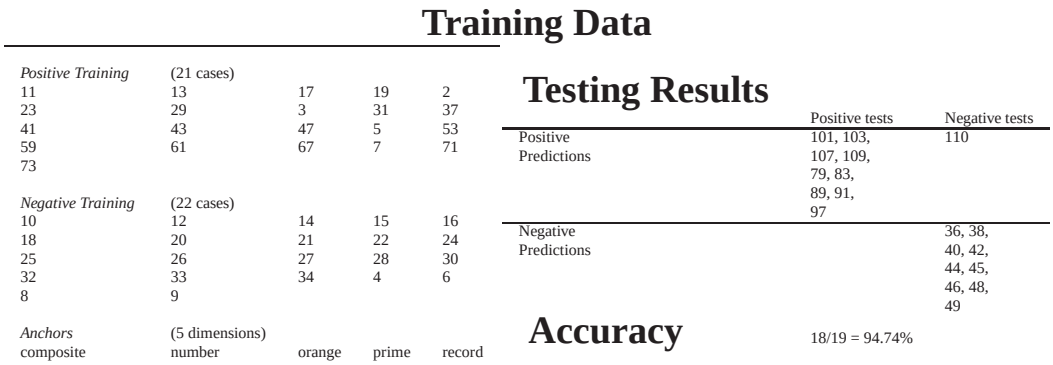

Figure 7.8: Google-SVM learning of primes.

7.6.2 Learning Prime Numbers

In Figure 7.8 the method learns to distinguish prime numbers from non-prime numbers by example:

The prime numbers example illustrates several common features of our method that distinguish it from the strictly deductive techniques. It is common for our classifications to be good but imperfect, and this is due to the unpredictability and uncontrolled nature of the Google distribution.

7.6.3 WordNet Semantics: Specific Examples

To create the next example, we used WordNet. WordNet is a semantic concordance of English. It also attempts to focus on the meaning of words instead of the word itself. The category we want to learn, the concept, is termed “electrical”, and represents anything that may pertain to electronics, Figure 7.9. The negative examples are constituted by simply everything else. Negative samples were chosen randomly and uniformly from a dictionary of English words. This category represents a typical expansion of a node in the WordNet hierarchy. The accuracy on the test set is 100%: It turns out that “electrical terms” are unambiguous and easy to learn and classify by our method.

In the next example, Figure 7.10, the concept to be learned is “religious”. Here the positive examples are terms that are commonly considered as pertaining to religious items or notions, the negative examples are everything else. The accuracy on the test set is 88.89%. Religion turns out to be less unequivocal and unambiguous than “electricity” for our method.

Notice that what we may consider to be errors, can be explained, or point at, a secondary meaning or intention of these words. For instance, some may consider the word “shepherd” to be full of religious connotation. And there has been more than one religion that claims to involve “earth” as a component. Such examples suggest to use the method for exploratory semantics: establishing less common, idiosyncratic, or jargon meaning of words.

Training Data

Positive Training	(58 cases)			
Cottrell precipitator	Van de Graaff generator	Wimshurst machine	aerial	antenna
attenuator	ballast	battery	bimetallic strip	board
brush	capacitance	capacitor	circuit	condenser
control board	control panel	distributer	electric battery	electric cell
electric circuit	electrical circuit	electrical condenser	electrical device	electrical distributor
electrical fuse	electrical relay	electrograph	electrostatic generator	electrostatic machine
filter	flasher	fuse	inductance	inductor
instrument panel	jack	light ballast	load	plug
precipitator	reactor	rectifier	relay	resistance
security	security measures	security system	solar array	solar battery
solar panel	spark arrester	spark plug	sparking plug	suppresser
transmitting aerial	transponder	zapper		
Negative Training	(55 cases)			
Andes	Burnett	Diana	DuPonts	Friesland
Gibbs	Hickman	Icarus	Lorraine	Madeira
Quakeress	Southernwood	Waltham	Washington	adventures
affecting	aggrieving	attractiveness	bearer	boll
capitals	concluding	constantly	conviction	damming
deeper	definitions	dimension	discounting	distinctness
exclamation	faking	helplessness	humidly	hurling
introduces	kappa	maims	marine	moderately
monster	parenthesis	pinches	predication	prospect
repudiate	retry	royalty	shopkeepers	soap
sob	swifter	teared	thrashes	tuples
Anchors	(6 dimensions)			
bumbled	distributor	premeditation	resistor	suppressor
swimmers				

Testing Results

	Positive tests	Negative tests
Positive Predictions	cell, male plug, panel, transducer, transformer	
Negative Predictions		Boswellizes, appointer, enforceable, greatness, planet

Accuracy

10/10 = 100.00%

Figure 7.9: Google-SVM learning of “electrical” terms.

Training Data

Positive Training	(22 cases)			
Allah	Catholic	Christian	Dalai Lama	God
Jerry Falwell	Jesus	John the Baptist	Mother Theresa	Muhammad
Saint Jude	The Pope	Zeus	bible	church
crucifix	devout	holy	prayer	rabbi
religion	sacred			
Negative Training	(23 cases)			
Abraham Lincoln	Ben Franklin	Bill Clinton	Einstein	George Washington
Jimmy Carter	John Kennedy	Michael Moore	atheist	dictionary
encyclopedia	evolution	helmet	internet	materialistic
minus	money	mouse	science	secular
seven	telephone	walking		
Anchors	(6 dimensions)			
evil	follower	history	rational	scripture
spirit				

Testing Results

	Positive tests	Negative tests
Positive Predictions	altar, blessing, communion, heaven, sacrament, testament, vatican	earth, shepherd
Negative Predictions	angel	Aristotle, Bertrand Russell, Greenspan, John, Newton, Nietzsche, Plato, Socrates, air, bicycle, car, fire, five, man, monitor, water, whistle

Accuracy

24/27 = 88.89%

Figure 7.10: Google-SVM learning of “religious” terms.

7.6.4 WordNet Semantics: Statistics

The previous examples show only a few hand-crafted special cases. To investigate the more general statistics, a method was devised to estimate how well the NGD -Google-SVM approach agrees with WordNet in a large number of automatically selected semantic categories. Each automatically generated category followed the following sequence.

First we must review the structure of WordNet; the following is paraphrased from the official WordNet documentation available online. WordNet is called a semantic concordance of the English language. It seeks to classify words into many categories and interrelate the meanings of those words. WordNet contains synsets. A synset is a synonym set; a set of words that are interchangeable in some context, because they share a commonly-agreed upon meaning with little or no variation. Each word in English may have many different senses in which it may be interpreted; each of these distinct senses points to a different synset. Every word in WordNet has a pointer to at least one synset. Each synset, in turn, must point to at least one word. Thus, we have a many-to-many mapping between English words and synsets at the lowest level of WordNet. It is useful to think of synsets as nodes in a graph. At the next level we have lexical and semantic pointers. Lexical pointers are not investigated in this thesis; only the following semantic pointer types are used in our comparison: A semantic pointer is simply a directed edge in the graph whose nodes are synsets. The pointer has one end we call a *source* and the other end we call a *destination*. The following relations are used:

1. *hyponym* : X is a hyponym of Y if X is a (kind of) Y.

2. *part meronym* : X is a part meronym of Y if X is a part of Y.

3. *member meronym* : X is a member meronym of Y if X is a member of Y.

4. *attribute* : A noun synset for which adjectives express values. The noun *weight* is an attribute, for which the adjectives *light* and *heavy* express values.

5. *similar to* : A synset is similar to another one if the two synsets have meanings that are substantially similar to each other.

Using these semantic pointers we may extract simple categories for testing. First, a random semantic pointer (or edge) of one of the types above is chosen from the WordNet database. Next, the source synset node of this pointer is used as a sort of root. Finally, we traverse outward in a breadth-first order starting at this node and following only edges that have an identical semantic pointer type; that is, if the original semantic pointer was a hyponym, then we would only follow hyponym pointers in constructing the category. Thus, if we were to pick a hyponym link initially that says a *tiger* is a *cat*, we may then continue to follow further hyponym relationships in order to continue to get more specific types of cats. See the WordNet homepage [37] documentation for specific definitions of these technical terms. For examples of each of these categories consult the experiments listed in the Appendix at [23].

Once a category is determined, it is expanded in a breadth first way until at least 38 synsets are within the category. 38 was chosen to allow a reasonable amount of training data to be presented with several anchor dimensions, yet also avoiding too many. Here, *Bernie's Rule*[1] is helpful: it states that the number of dimensions in the input data must not exceed one tenth the number of training samples. If the category cannot be expanded this far, then a new one is chosen. Once a suitable category is found, and a set of at least 38 members has been formed, a training set is created using 25 of these cases, randomly chosen. Next, three are chosen randomly as anchors. And finally the remaining ten are saved as positive test cases. To fill in the negative training cases, random words are chosen from the WordNet database. Next, three random words are chosen as unrelated anchors. Finally, 10 random words are chosen as negative test cases.

For each case, the SVM is trained on the training samples, converted to 6-dimensional vectors using NGD . The SVM is trained on a total of 50 samples. The kernel-width and error-cost parameters are automatically determined using five-fold cross validation. Finally testing is performed using 20 examples in a balanced ensemble to yield a final accuracy.

There are several caveats with this analysis. It is necessarily rough, because the problem domain is difficult to define. There is no protection against certain randomly chosen negative words being accidentally members of the category in question, either explicitly in the greater depth transitive closure of the category, or perhaps implicitly in common usage but not indicated in WordNet. In several cases, such as "radio wave" and "DC" in the "big science" experiment, there appears to be an arguable case to support the computer's classification in cases where this phenomenon occurs. Another detail to notice is that WordNet is available through some web pages, and so undoubtedly contributes something to Google pagecounts. Further experiments

[1] Allegedly named after Bernie Widrow in the context of neural network training.

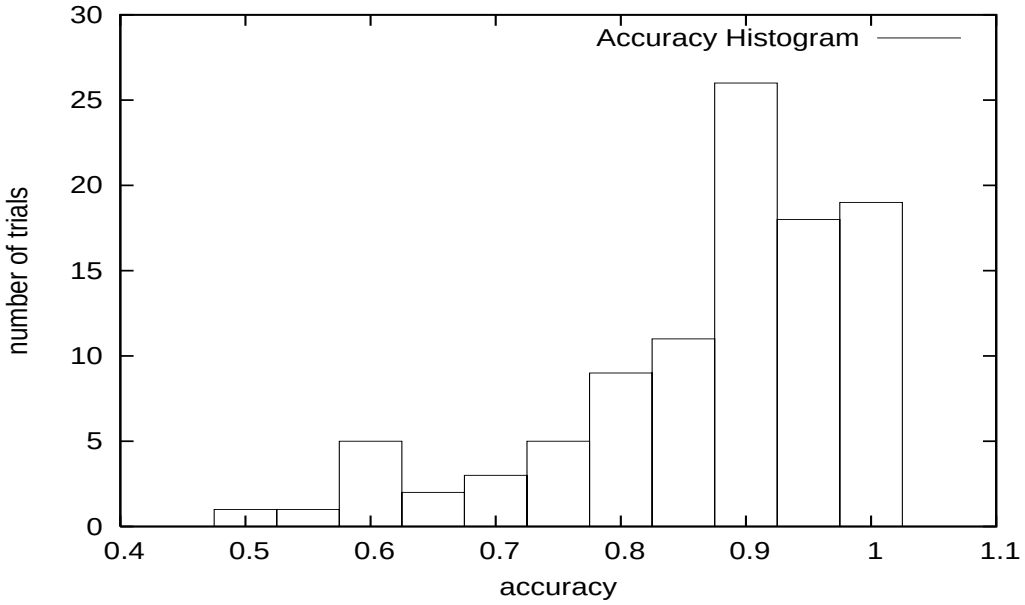

Figure 7.11: Histogram of accuracies over 100 trials of WordNet experiment.

comparing the results when filtering out WordNet images on the web suggest that this problem does not usually affect the results obtained, except when one of the anchor terms happens to be very rare and thus receives a non-negligible contribution towards its page count from WordNet views. In general, our previous NCD based methods, as in [22], exhibit large-granularity artifacts at the low end of the scale; for small strings we see coarse jumps in the distribution of NCD for different inputs which makes differentiation difficult. With the Google-based NGD we see similar problems when page counts are less than a hundred.

We ran 100 experiments. The actual data are available at [23]. A histogram of agreement accuracies is shown in Figure 7.11. On average, our method turns out to agree well with the WordNet semantic concordance made by human experts. The mean of the accuracies of agreements is 0.8725. The variance is ≈ 0.01367, which gives a standard deviation of ≈ 0.1169. Thus, it is rare to find agreement less than 75%. These results confirm that we are able to perform a rudimentary form of *generalization* within a *conceptual domain* programmatically using Google. For hand-crafted examples it performed comparably, and so this suggests that there may be latent semantic knowledge. Is there a way to use it?

7.7 Matching the Meaning

Yet another potential application of the NGD method is in natural language translation. (In the experiment below we do not use SVM's to obtain our result, but determine correlations instead.) Suppose we are given a system that tries to infer a translation-vocabulary among English and Spanish. Assume that the system has already determined that there are five words that appear in two different matched sentences, but the permutation associating the English and Spanish

Given starting vocabulary	
English	**Spanish**
tooth	diente
joy	alegria
tree	arbol
electricity	electricidad
table	tabla
money	dinero
sound	sonido
music	musica
Unknown-permutation vocabulary	
plant	bailar
car	hablar
dance	amigo
speak	coche
friend	planta

Figure 7.12: English-Spanish Translation Problem.

words is, as yet, undetermined. This setting can arise in real situations, because English and Spanish have different rules for word-ordering. Thus, at the outset we assume a pre-existing vocabulary of eight English words with their matched Spanish translation. Can we infer the correct permutation mapping the unknown words using the pre-existing vocabulary as a basis? We start by forming an NGD matrix using additional English words of which the translation is known, Figure 7.12. We label the columns by the translation-known English words, the rows by the translation-unknown words. The entries of the matrix are the NGD 's of the English words labeling the columns and rows. This constitutes the English basis matrix. Next, consider the known Spanish words corresponding to the known English words. Form a new matrix with the known Spanish words labeling the columns in the same order as the known English words. Label the rows of the new matrix by choosing one of the many possible permutations of the unknown Spanish words. For each permutation, form the NGD matrix for the Spanish words, and compute the pairwise correlation of this sequence of values to each of the values in the given English word basis matrix. Choose the permutation with the highest positive correlation. If there is no positive correlation report a failure to extend the vocabulary. In this example, the computer inferred the correct permutation for the testing words, see Figure 7.13.

7.8 Conclusion

A comparison can be made with the *Cyc* project [71]. Cyc, a project of the commercial venture Cycorp, tries to create artificial common sense. Cyc's knowledge base consists of hundreds of microtheories and hundreds of thousands of terms, as well as over a million hand-crafted assertions written in a formal language called CycL [90]. CycL is an enhanced variety of first

	English	Spanish
Predicted (optimal) permutation	plant	planta
	car	coche
	dance	bailar
	speak	hablar
	friend	amigo

Figure 7.13: Translation Using NGD.

order predicate logic. This knowledge base was created over the course of decades by paid human experts. It is therefore of extremely high quality. Google, on the other hand, is almost completely unstructured, and offers only a primitive query capability that is not nearly flexible enough to represent formal deduction. But what it lacks in expressiveness Google makes up for in size; Google has already indexed more than eight billion pages and shows no signs of slowing down.

Epistemology: In the case of context-free statistical compression such as `gzip` , we are trying to approximate the Kolmogorov complexity of a string. Another way of describing the calculation is to view it as determining a probability mass function (viewing the compressed string as Shannon-Fano code, Section 2.7), approximating the *universal distribution*, that is, the negative exponential of the Kolmogorov complexity [79]. The universal probability of a given string can equivalently be defined as the probability that the reference universal Turing machine outputs the string if its input program is generated by fair coin flips. In a similar manner, we can associate a particular Shannon-Fano code, the *Google code*, with the Google probability mass function. Coding every search term by its Google code, we define a "Google compressor." Then, in the spirit of Section 7.3, we can view the Google probability mass function as a universal distribution for the individual Google probability mass functions generated by the individual web authors, substituting "web authors" for "Turing machines".

Concerning the SVM method: The Google-SVM method does not use an individual word in isolation, but instead uses an ordered list of its NGD relationships with fixed anchors. This then removes the possibility of attaching to the isolated (context-free) interpretation of a literal term. That is to say, the inputs to our SVM are not directly search terms, but instead an image of the search term through the lens of the Google distribution, and relative to other fixed terms which serve as a grounding for the term. In most schools of ontological thought, and indeed in the WordNet database, there is imagined a two-level structure that characterizes language: a many-to-many relationship between word-forms or utterances and their many possible meanings. Each link in this association will be represented in the Google distribution with strength proportional to how common that usage is found on the web. The NGD then amplifies and separates the many contributions towards the aggregate page count sum, thereby revealing some components of the latent semantic web. In almost every informal theory of cognition we have the idea of connectedness of different concepts in a network, and this is precisely the structure that our

experiments attempt to explore.

Universality: The Google distribution is a comparable notion, in the context of the world-wide-web background information, to the universal distribution: The universal distribution multiplicatively dominates all other distributions in that it assigns a higher weight to some elements when appropriately scaled. This suggests that it covers everything without bound. Google surely represents the largest publicly-available single corpus of aggregate statistical and indexing information so far created. Only now has it been cheap enough to collect this vast quantity of data, and it seems that even rudimentary analysis of this distribution yields a variety of intriguing possibilities. One of the simplest avenues for further exploration must be to increase training sample size, because it is well-known that SVM accuracy increases with training sample size. It is likely that this approach can never achieve 100% accuracy like in principle deductive logic can, because the Google distribution mirrors humankind's own imperfect and varied nature. But it is also clear that in practical terms the NGD can offer an easy way to provide results that are good enough for many applications, and which would be far too much work if not impossible to program in a foolproof deductive way.

The Road Ahead: We have demonstrated that NGD can be used to extract meaning in a variety of ways from the statistics inherent to the Google database. So far, all of our techniques look only at the page count portion of the Google result sets and achieve surprising results. How much more amazing might it be when the actual contents of search results are analyzed? Consider the possibility of using WordNet familiarity counts to filter returned search results to select only the least familiar words, and then using these in turn as further inputs to NGD to create automatic discourse or concept diagrams with arbitrary extension. Or perhaps this combination can be used to expand existing ontologies that are only seeded by humans. Let us list some of the future directions and potential application areas:

1. There seems to also be an opportunity to apply these techniques to generic language acquisition, word sense disambiguation, knowledge representation, content-filtration and collaborative filtering, chat bots, and discourse generation.

2. There are potential applications of this technique to semi-intelligent user-interface design; for predictive completion on small devices, speech recognition, or handwriting recognition.

3. A user interface possibility is the idea of concept-class programming for non-programmers, or software to form a conceptual predicate by way of example without forcing the user to learn a formal programming language. This might be used, for example, in a network content filtration system that is installed by non-programmer parents to protect their young children from some parts of the internet. Or perhaps an IT manager is able to adjust the rule determining if a particular email message is a well-known virus and should be filtered without writing explicit rules but just showing some examples.

4. How many people are able to write down a list of prime numbers as shown in an earlier test case, Figure 7.8, compared to how many people are able to write a program in a real programming language that can calculate prime numbers? Concept clustering by example is significantly simpler than any formal programming language and often yields remarkably accurate results without any effort at hand-tuning parameters.

5. The colors versus numbers tree example, Figure 7.3, is rife with possibilities. A major challenge of the Semantic Web and XML as it stands is in integrating diverse ontologies created by independent entities [67]. XML makes the promise of allowing seamless integration of web services via customized structured tags. This promise is for the most part unrealized at this point on the web, however, because there is not yet sufficient agreement on what sets of XML tags to use in order to present information; when two different parties each build databases of recipes, but one organizes the recipes according to their country of origin and another according to their sweetness or savory flavor, these two databases cannot "understand" one another insofar as they may exchange recipes. XML allows us to format our data in a structured way, but fails to provide for a way for different structure conventions to interoperate. There have been many attempts to solve this and none have been satisfactory. Usually solutions involve mapping the separate schemas into some sort of global schema and then creating a global standardization problem that requires significant coordinated effort to solve. Another approach is to create a meta-language like DAML that allows for automatic translation among certain very similar types of ontologies, however this requires a great deal of effort and forethought on the part of the schema designers in advance and is brittle in the face of changing ontologies. By using NGD we may create a democratic and natural ontology for almost any application in an unsupervised way. Furthermore, if instead we want finer control over the ontological organization, then a human expert may define a custom ontology and then NGD may be used to provide a normal, global, and automatic reference frame within which this ontology may be understood without additional costly human effort. So, for example, NGD may be used in the recipe example above, Figure 7.12, 7.13, to automatically "understand" the difference between a Chinese or Mediterranean recipe, and could thus be used to automatically translate between the two conflicting ontologies.

6. Another future direction is to apply multiple concurrent binary classifiers for the same classification problem but using different anchors. The separate classifications would have to be combined using a voting scheme, boosting scheme, or other protocol in an effort to boost accuracy.

This section owes thanks to Teemu Roos, Hannes Wettig, Petri Mylliymaki, and Henry Tirri at COSCO and The Helsinki Institute for Information Technology for interesting discussions. We also thank Chih-Jen Lin and his excellent group for providing and supporting vigorously, free of charge to all, the very easy to use LIBSVM package. We thank the Cognitive Science Laboratory at Princeton University for providing the wonderful and free WordNet database. And we wish to thank the staff of Google, Inc. for their delightful support of this research by providing an API as well as generous access to their websearch system.

Chapter 8

Stemmatology

Stemmatology studies relations among different variants of a text that has been gradually altered as a result of imperfectly copying the text over and over again. This chapter presents a new method for using this assumption to reconstruct a lineage tree explicating the derivational relationships among the many variations, just as we might reconstruct an evolutionary tree from a set of gene sequences. We propose a new computer assisted method for stemmatic analysis based on compression of the variants. We will provide an overview of the chapter at the end of the next section. The method is related to phylogenetic reconstruction criteria such as maximum parsimony and maximum likelihood. We apply our method to the tradition of the legend of St. Henry of Finland, and report encouraging preliminary results. The obtained family tree of the variants, the stemma, corresponds to a large extent with results obtained with more traditional methods. Some of the identified groups of manuscripts are previously unrecognized ones. Moreover, due to the impossibility of manually exploring all plausible alternatives among the vast number of possible trees, this work is the first attempt at a complete stemma for the legend of St. Henry. The used methods are being released as open-source software, and are entirely distinct from the CompLearn system. They are presented here only for rough comparison.

8.1 Introduction

St. Henry, according to the medieval tradition Bishop of Uppsala (Sweden) and the first Bishop of Finland, is the key figure of the Finnish Middle Ages. He seems to have been one of the leaders of a Swedish expedition to Finland probably around 1155. After this expedition Henry stayed in Finland with sad consequences: he was murdered already next year. He soon became the patron saint of Turku cathedral and of the bishopric covering the whole of Finland. He remained the only 'local' one of the most important saints until the reformation. Henry is still considered to be the Finnish national saint. The knowledge of writing was almost totally concentrated into the hands of the Church and the clergymen during the early and high Middle Ages. On the other hand, the official and proper veneration of a saint needed unavoidably a written text containing the highlights of the saint's life and an account of his miracles to be recited during the services in the church. The oldest text concerning St. Henry is his legend written in Latin. It contains

Figure 8.1: An excerpt of a 15th century manuscript ‘H’ from the collections of the Helsinki University Library, showing the beginning of the legend of St. Henry on the right: *“Incipit legenda de sancto Henrico pontifice et martyre; lectio prima; Regnante illustrissimo rege sancto Erico, in Suecia, uenerabilis pontifex beatus Henricus, de Anglia oriundus, ...”* [47].

both his life and a collection of his miracles and seems to have been ready by the end of the 13th century at the very latest. The text is the oldest literary work preserved in Finland and can thus be seen as the starting point of the Finnish literary culture. Whereas the influence of St. Henry on the Christianization of Finland has been one of the focusing points of the Finnish and Swedish medievalists for hundreds of years, only the most recent research has really concentrated on his legend as a whole. According to the latest results, the Latin legend of St. Henry is known in 52 different medieval versions preserved in manuscripts and incunabula written in the early 14th–early 16th centuries (Fig. 8.1).[1]

The reasons for such a substantial amount of versions differing from each other are several. On one hand, the texts were copied by hand until the late 15th and early 16th centuries, which resulted in a multitude of unintended scribal errors by the copyists. In addition, the significance of

[1]For identification of the sources as well as a modern edition of the legend see [47].

the cult of St. Henry varied considerably from one part of the Latin Christendom to the other. In the medieval bishopric of Turku covering the whole of medieval Finland St. Henry was venerated as the most important local saint, whose adoration required the reciting of the whole legend during the celebrations of the saint's day. In Sweden, for instance, St. Henry was not so important a saint, which led to different kinds of abridgments fitted into the needs of local bishoprics and parishes. As a consequence, the preserved versions of the legend are all unique.

With the aid of traditional historically oriented auxiliary sciences like codicology and paleography it is possible to find out — at least roughly — where and when every version was written. Thus, the versions form a pattern representing the medieval and later dissemination of the text. Even if the existent manuscripts containing the different versions represent but a tiny part of the much larger number of manuscripts and versions written during the Middle Ages, they still provide us with an insight into a variety of aspects of medieval culture. The versions help to reconstruct the actual writing process and the cultural ties that carried the text from one place to another. When one combines the stemma — i.e. the family tree — of a text with a geographical map and adds the time dimension, one gets important information that no single historical source can ever provide a historian with. The potential of this kind of an approach is emphasized when researching hagiographical texts — i.e. saints' lives, for instance — since they were the most eagerly read and most vastly disseminated literary genre of the Middle Ages.

Taking into consideration the possibilities of stemmatology, it is not surprising that the historians and philologists have tried to establish a reliable way to reconstruct the stemma of the text and its versions for centuries. The main difficulty has been the great multitude of textual variants that have to be taken into consideration at the same time. An example from the legend material of St. Henry shall elucidate the problems: there are over 50 manuscripts and incunabula to be taken into consideration; in the relatively short text there are nearly one thousand places where the versions differ from each other. Since the multitude of variants rises easily to tens of thousands, it has been impossible for researchers using traditional methods of paper and pen to form the stemma and thus get reliable answers to the questions related to the writing and disseminating of the text. There have been some previous attempts to solve the problems of stemmatology with the aid of computer science. In addition, the powerful computer programs developed for the needs of the computer aided cladistics in the field of evolutionary biology have been used. In many cases this has proven to be a fruitful approach, extending the possibilities of stemmatics to the analysis of more complex textual traditions that are outside the reach of manual analysis. Moreover, formalizing the often informal and subjective methods used in manual analysis makes the methods and results obtained with them more transparent and brings them under objective scrutiny. Still, many issues in computer assisted stemmatic analysis remain unsolved, underlining the importance of advances towards general and reliable methods for shaping the stemma of a text.

Overview of this Chapter: The chapter is organized as follows: In Section 8.2 we present a criterion for stemmatic analysis that is based on compression of the manuscripts. We then outline an algorithm, in Section 8.3, that builds stemmata by comparing a large number of tree-shaped stemmata and choosing the one that minimizes the criterion. The method is demonstrated on a simple example in Section 8.4, where we also present our main experiment using some 50 variants of the legend of St. Henry, and discuss some of the restrictions of the method and

potential ways to overcome them. Conclusions are presented in Section 8.5. We also compare our method to a related method in the CompLearn package in Appendix A.

8.2 A Minimum-Information Criterion

One of the most applied methods in biological phylogeny is so-called maximum parsimony. A maximally parsimonious tree minimizes the total number of differences between connected nodes — i.e., species, individuals, or manuscripts that are directly related — possibly weighted by their importance. Stemmatology analysis is based on variable readings that result from unintentional errors in copying or intentional omissions, insertions, or other modifications. In his seminal work on computer assisted stemmatology, O'Hara used a parsimony method of the PAUP software [110] in Robinson's Textual Criticism challenge [93]. For further applications of maximum parsimony and related method, see [49, 69, 107, 117] and references therein.

The compression-based *minimum information* criterion shares many properties of the very popular maximum parsimony method. Both can also be seen as instances of the *minimum description length* (MDL) principle of Rissanen [91] — although this is slightly anachronistic: the maximum parsimony method predates the more general MDL principle — which in turn is a formal version of Occam's razor. The underlying idea in the minimum information criterion is to minimize the amount of information, or *code-length*, required to reproduce all the manuscripts by the process of copying and modifying the text under study. In order to describe a new version of an existing manuscript, one needs an amount of information that depends on both the amount and the type of modifications made. For instance, a deletion of a word or a change of word order requires less information to describe compared to introducing a completely new expression. In order to be concrete, we need a precise, numerical, and computable measure for the amount of information. The commonly accepted definition of the amount information in individual objects is Kolmogorov complexity [57, 79], defined as the length of the shortest computer program to describe the given object, as explained in Chapter 3. However, Kolmogorov complexity is defined only up to a constant that depends on the language used to encode programs, and what is more, fundamentally uncomputable. In the spirit of a number of earlier authors [7, 10, 20, 22, 45, 82, 115] we approximate Kolmogorov complexity by using a compression program, also as we did in previous chapters. Currently, we use `gzip` based on the LZ77 [122] algorithm, and plan to experiment with other compressors in subsequent work. In particular, given two strings, x and y, the amount of information in y conditional on x, denoted by $C(y \mid x)$ is given by the length of the compressed version of the concatenated string x,y minus the length of the compressed version of x alone[2]. A simple example illustrating these concepts is given below in Section 8.4.

In addition to the MDL interpretation, our method can be seen as (an approximation of) maximum likelihood, another commonly used criterion in phylogeny. The maximum likelihood criterion requires that we have a probabilistic model for evolution, assigning specific probabilities for each kind of change. The joint likelihood of the whole graph is then evaluated as a product

[2]We insert a newline in the end of each string and between x and y.

of likelihoods of the individual changes. The tree achieving the highest joint likelihood given the observed data is then preferred. In the case of manuscripts such a model is clearly more difficult to construct that in biology, where the probabilities of mutation can be estimated from experimental data. Nevertheless, a model for manuscript evolution is presented in [106]. Code-length is isomorphic to (behaves in the same way as) likelihood: sums of code-lengths have a direct correspondence with products of likelihoods. If the probability induced by the information cost, $2^{-C(y|x)}$, is approximately proportional to the likelihood of creating a copy y based on the original x, then minimizing the total information cost approximates maximizing the likelihood.

Let $G = (V,E)$ be an undirected graph where V is a set of nodes corresponding to the text variants, $E \subset V \times V$ is a set of edges. We require that the graph is a connected bifurcating tree, i.e., that (i) each node has either one or three neighbors, and (ii) the tree is acyclic. Such a graph G can be made directed by picking any one of the nodes as a root and directing each edge away from the root. Given a directed graph $\vec{G}$, the total information cost of the tree is given by

$$\begin{aligned} C(\vec{G}) &= \sum_{v \in V} C(v \mid \mathrm{Pa}(v)) \\ &= \sum_{v \in V} C(\mathrm{Pa}(v), v) - C(\mathrm{Pa}(v)), \end{aligned} \tag{8.2.1}$$

where $\mathrm{Pa}(v)$ denotes the parent node of v unless v is the root in which case $\mathrm{Pa}(v)$ is the empty string. Assuming that order has no significant effect on the complexity of a concatenated string, i.e., we have $C(x,y) \approx C(y,x)$, as seems to be the case in our data, it can easily verified that for acyclic bifurcating trees, the above can rewritten as

$$C(G) \approx \sum_{(v,w) \in E} C(v,w) - 2 \sum_{v \in V_I} C(v), \tag{8.2.2}$$

where the first summation has a term for each edge in the graph, and the second summation goes over the set of interior nodes V_I. The formula is a function of the undirected structure G only: the choice of the root is irrelevant. The factor two in the latter term comes from using *bi*furcating trees.

For practical reasons we make three modifications to this criterion. First, as we explain in the next section, due to algorithmic reasons we need to splice the texts in smaller segments, not longer than roughly 10–20 words (we used 11). Secondly, we found that the cost assigned by `gzip` to reproducing an identical copy of a string is too high in the sense that it is sometimes 'cheaper' to omit a large part of the text for a number of generations and to re-invent it later in an identical form. Therefore we define the cost of making an identical copy to be zero. Thirdly, it is known that the variation between an ampersand ('&') and the word *et*, and the letters *v* and *u* was mostly dependent on the style of the copyist and changed with time and region, and thus, bears little information relevant to stemmatic analysis. This domain knowledge was taken into account by replacing, in both of the above cases, all occurrences of the former by the latter[3]. Thus, we

[3]Howe *et al.* [49] use as an example the words *kirk* and *church* in 15th century English whose variation mainly reflects local dialect.

use the following modified cost function

$$C'(\vec{G}) = \sum_{v \in V} \sum_{i=1}^{n} C'(v_i \mid \mathrm{Pa}_i(v)), \tag{8.2.3}$$

where n is the number of segments into which each text is spliced, v_i and $\mathrm{Pa}_i(v)$ are the ith segment of variant v and its parent, respectively, all strings are modified according to the above rules (ampersand to *et*, and v to u), and $C'(x \mid y)$ equals the `gzip` cost if x and y differ, and zero otherwise. This modified cost also allows a form similar to (8.2.2) and hence, is practically independent of the choice of the root.

8.3 An Algorithm for Constructing Stemmata

Since it is known that many of the text variants have been lost during the centuries between the time of the writing of the first versions and present time, it is not realistic to build a tree of only the about 50 variants that we have as our data. This problem is even more prominent in biology where we can only make observations about organisms that still exist (excluding fossil evidence). The common way of handling this problem is to include in the tree a number of 'hidden' nodes, i.e., nodes representing individuals whose characteristics are unobserved. We construct bifurcating trees that have N observed nodes as leafs, and $N-2$ hidden nodes as the interior nodes.

Evaluating the criterion (8.2.3) now involves the problem of dealing with the hidden nodes. Without knowing the values of $\mathrm{Pa}_i(v)$, it is not possible to compute $C'(v \mid \mathrm{Pa}_i(v))$. We solve this problem by searching simultaneously for the best tree structure $\vec{G}$ and for the optimal contents of the hidden nodes with respect to criterion (8.2.3). As mentioned above, we patch up the contents of the interior nodes from segments of length 10–20 words appearing in some of the available variants. In principle we would like to do this on a per-word-basis, which would not be a notable restriction since it is indeed reasonable to expect that a reconstruction only consists of words appearing in the available variants — any other kind of behavior would require rather striking innovation. However, since we evaluate the `gzip` cost in terms of the segments, it is likely give better values when the segments are longer than one word. Secondly, one of the most common modifications is change in word order. Using 10-20 word segments we assign less cost to change in word order than to genuine change of words, unless the change happens to cross a segment border.

Perhaps surprisingly, given a tree structure, finding the optimal contents is feasible. The method for efficiently optimizing the contents of the hidden nodes is an instance of dynamic programming and called 'the Sankoff algorithm' [40] or 'the Felsenstein's algorithm' [104]. As Siepel and Haussler [104] note, it is in fact an instance of a 'message-passing' or 'elimination' algorithm in graphical models (see also [42]). The basic idea is to maintain for each node a table of minimal costs for the whole subtree starting at the node, given that the contents of the node take any given value. For instance, let us fix a segment, and denote by $x^1, \ldots, x^m$ the different versions of the segment that appear in some of the observed variants. The minimal cost for the

subtree starting at node i, given that the segment in question of node i contains the string x^j is given by (see [40])

$$\text{cost}_i(j) = \min_k \left[C'(x^k \mid x^j) + \text{cost}_a(k) \right] + \min_l \left[C'(x^l \mid x^j) + \text{cost}_b(l) \right],$$

where a and b are the two children of node i. For leaf nodes the cost is defined as being infinite if j does not match the known content of the node, and zero if j matches or if the content of the node is unknown. Evaluating $\text{cost}_i(j)$ can be done for each segment independently, starting from the leaf nodes and working towards the root. Finally, the (unconditional) complexity of the root is added so that the minimal cost of the segment is obtained by choosing at the root the string x^j that minimizes the sum $\text{cost}_{\text{root}}(j) + C'(x^j)$. The total cost of the tree is then obtained by summing over the minimal costs for each segment. After this, actually filling the contents can be done by propagating back down from the root towards the leafs. It is important to remember that while the algorithm for optimizing the contents of the hidden nodes requires that a root is selected, the resulting cost and the optimal contents of the hidden nodes only depend on the undirected structure (see Eq. (8.2.2)).

There still remains the problem of finding the tree structure, which together with corresponding optimal contents of the hidden nodes minimizes criterion (8.2.3). The obvious solution, trying all possible tree structures and choosing the best one, fails because for N leafs nodes, the number of possible bifurcating trees is as large as (see [40])

$$1 \times 3 \times 5 \times \ldots \times (2N-5).$$

For $N = 52$ this number is about 2.73×10^{78}, which is close to the estimated number of atoms in the universe. Instead, we have to resort to heuristic search, trying to find as good a tree as possible in the time available.

We use a simulated annealing algorithm which starts with an arbitrary tree and iteratively tries to improve it by small random modification, such as exchanging the places of two subtrees[4]. Every modification that reduces the value of the criterion is accepted. In order to escape local optima in the search space, modifications that increase the value are accepted with probability

$$\exp\left(\frac{C'_{\text{old}} - C'_{\text{new}}}{T} \right),$$

where C'_{old} is the cost of the current tree, C'_{new} is the cost of the modified tree, and T is a 'temperature' parameter that is slowly decreased to zero. In our main experiment, reported in the next section, we ran 1,200,000 iterations of annealing, which we found to be sufficient in our setting.

8.4 Results and Discussion

We first illustrate the behavior of the method by an artificial example in Fig. 8.2. Assume that we have observed five pieces of text, shown at the tips of the tree's branches. Because the text

[4]The algorithm also takes advantage of the fact that changes like exchanging subtrees only require partial updating of the dynamic programming table used to evaluate the information cost.

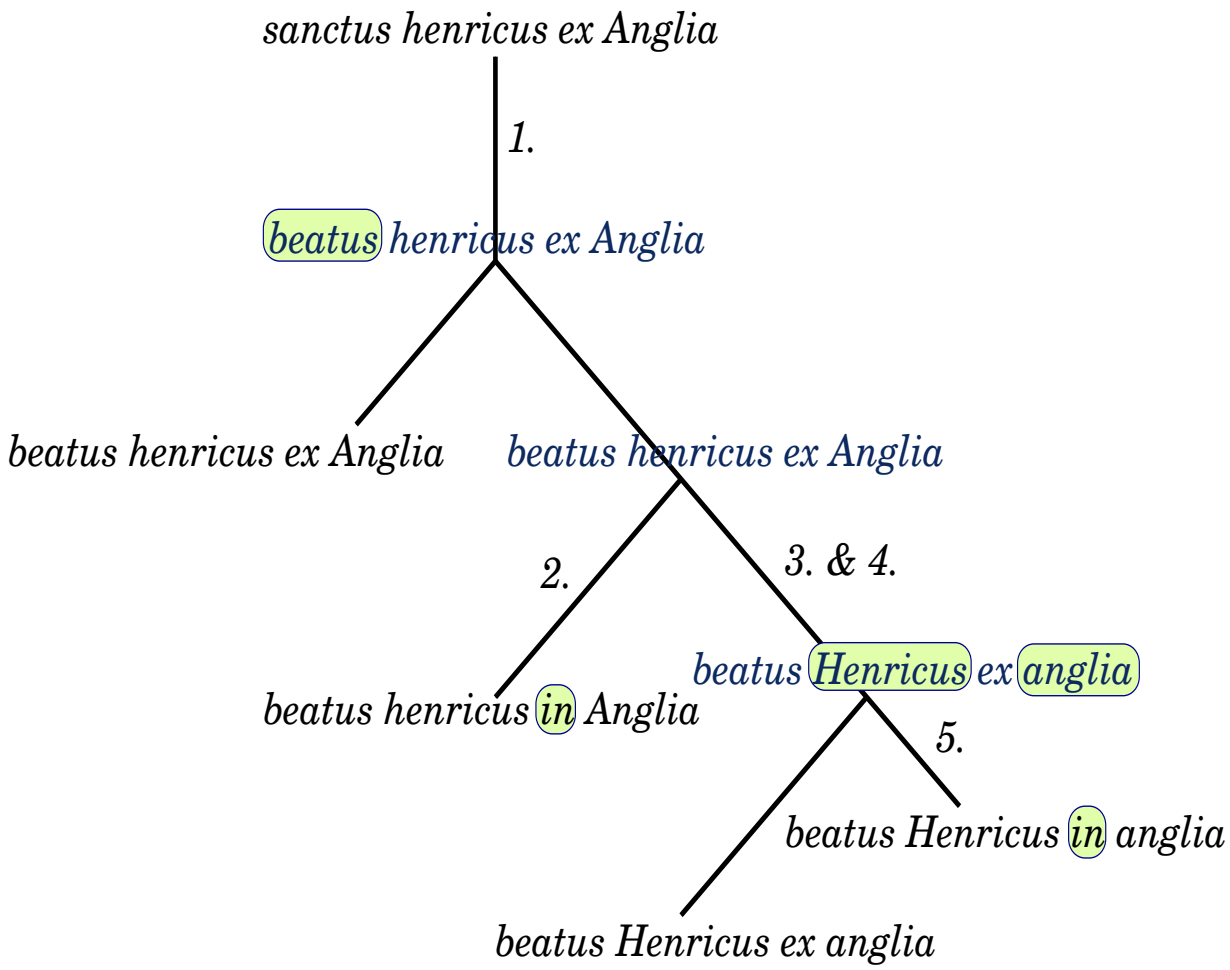

Figure 8.2: An example tree obtained with the compression-based method. Changes are circled and labeled with numbers 1–5. Costs of changes are listed in the box. Best reconstructions at interior nodes are written at the branching points.

is so short, the length of the segment was fixed to one word. One of the trees — not the only one — minimizing the information cost with total cost of 44 units (bytes) is drawn in the figure. Even though, as explained above, the obtained tree is undirected, let us assume for simplicity that the original version is the topmost one (“*sanctus henricus ex Anglia*”). The sum of the (unconditional) complexities of the four words in this string is equal to $8+9+3+7=27$, which happens to coincide with the length of the string, including spaces and a finishing newline. The changes, labeled by number 1–5 in the figure, yield $5+3+3+3+3=17$ units of cost. Thus the total cost of the tree equals $27+17=44$ units.

As our main experiment, we analyzed a set of 49 variants of the legend of St. Henry. We had prepared four out of the nine sections (sections 1,4,5, and 6) in a suitable format. Three variants were excluded since they had only ten words or less in the prepared sections. The remaining

variants contained 33–379 words each. Table 8.5 on page 150 lists the estimated time or writing and place of origin, as well as the number of words in the used sections for each manuscript. The best (wrt. the information cost) tree found is shown in Fig. 8.3. By comparing the tree with earlier results [47], it can be seen that many groups of variants have been successfully placed next to each other. For instance, groups of Finnish variants appearing in the tree that are believed to be related are Ho–I–K–T and R–S. Among the printed versions the pairs BA–BS and BLu–BL are correctly identified and also grouped close the each other[5]. Other pairs of variants appearing in the tree that are believed to be directly related are Li–Q (that are also correctly associated with BA–BS and BL–BLu), JG–B, Dr–M, NR2–JB, LT–E, AJ–D, and Bc–MN–Y. In addition, the subtree including the nine nodes between (and including) BU and Dr is rather well supported by traditional methods. All in all, the tree corresponds very well with relationships discovered with more traditional methods. This is quite remarkable taking into account that in the current experiments we have only used four out of the nine sections of the legend.

In order to quantify confidence in the obtained trees we used on top of our method, block-wise bootstrap [66] and a consensus tree program in the phylogeny inference package PHYLIP [41], Section 9. One hundred bootstrap samples were generated by sampling (with replacement) n segments out of the n segments that make each manuscript. The compression-based method described in this work was run on each bootstrap sample — this took about a week of computation — and the resulting 100 trees were analyzed with the `consense` program in PHYLIP using default settings (modified majority rule). The resulting consensus tree is shown in Fig. 8.4.

It should be noted that the central node with nine neighbors does not corresponds to a single manuscript with nine descendants, but rather, that the relationships between the nine subtrees is unidentified. Because the interpretation of the consensus tree is less direct than the interpretation of the tree in Fig. 8.3 as the family tree of the variants, it is perhaps best to use the consensus tree to quantify the confidence in different parts of the tree in Fig. 8.3. For instance, it can be seen that the pairs BL–BLu, AJ–D, Li–Q, NR2–JB, O–P, L–G, JG–B, and R–S are well supported. More interestingly, The group Ho–I–K–T–A is organized in a different order in Fig. 8.3 and the consensus tree. This group also illustrates one of the problems in the consensus tree method. Namely the confidence in contiguous groups that are in the middle of the tree tends to be artificially low since the group does not make up a subtree, in this case only $3/100$ (Fig. 8.4).

The following potential problems and sources of bias in the resulting stemmata are roughly in decreasing order of severity:

1. The `gzip` algorithm does not even attempt to fully reflect the process of imperfectly copying manuscripts. It remains to be studied how sensible the `gzip` information cost, or costs based on other compression algorithms, are in stemmatic analysis.

2. Trees are not flexible enough to represent all realistic scenarios. More than one original manuscript may have been used when creating a new one — a phenomenon termed *contamination* (or horizontal transfer in genomics). Point 5 below may provide a solution but for non-tree structures the dynamic programming approach does not work and serious computational problems may arise.

[5]The printed versions are especially suspect to contamination since it is likely that more than one manuscript was used when composing a printed version.

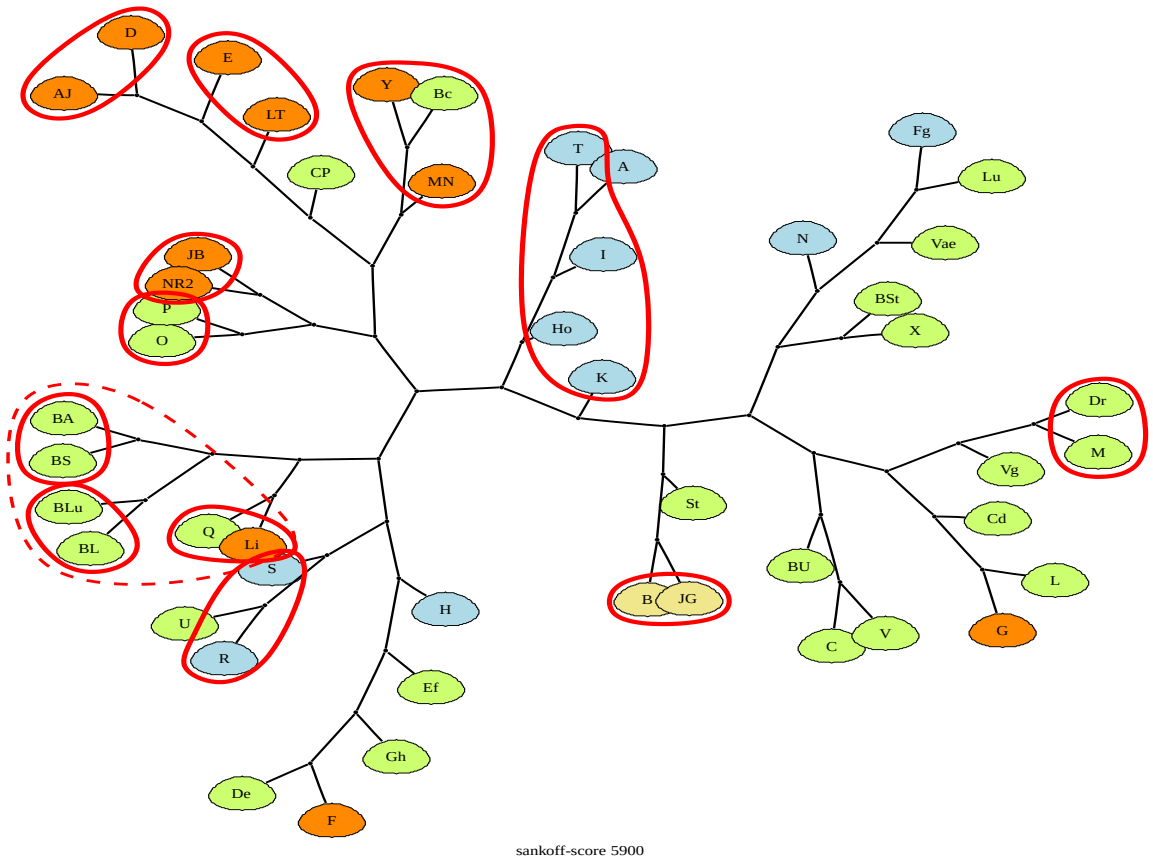

Figure 8.3: Best tree found. Most probable place of origin according to [47], see Table 8.5, indicated by color — Finland (blue): K,Ho,I,T,A,R,S,H,N,Fg; Vadstena (red): AJ,D,E,LT,MN,Y,JB,NR2,Li,F,G; Central Europe (yellow): JG,B; other (green). Some groups supported by earlier work are circled in red.

3. Patching up interior node contents from 10–20 word segments is a restriction. This restriction could be removed for cost functions that are defined as a sum of individual words' contributions. Such cost functions may face problems in dealing with change of word order.

4. The number of copies made from a single manuscript can be other than zero and two. The immediate solution would be to use multifurcating trees in combination with our method, but this faces the problem that the number of internal nodes strongly affects the minimum-information criterion. The modification hinted to at point 5 may provide a solution to this problem.

5. Rather than looking for the tree structure that together with the optimal contents of the interior nodes minimizes the cost, it would be more principled from a probabilistic point of view to 'marginalize' the interior nodes (see [42]). In this case we should also account for possible forms (words or segments) not occurring in any of the observed variants.

6. The search space is huge and the algorithm only finds a local optimum whose quality cannot be guaranteed. Bootstrapping helps to identify which parts of the tree are uncertain due to problems in search (as well as due to lack of evidence).

7. Bootstrapping is known to underestimate the confidence in the resulting consensus tree. This is clearly less serious than *over*estimation.

In future work we plan to investigate ways to overcome some of these limitations, to carry out more experiments with more data in order to validate the method and to compare the results with those obtained with, for instance, the existing methods in CompLearn [21], PHYLIP [41], and PAUP [110]. We are also planning to release the software as a part of the CompLearn package. Among the possibilities we have not yet explored is the reconstruction of a likely original text. In fact, in addition to the stemma, the method finds an optimal — i.e., optimal with respect to the criterion — history of the manuscript including a text version at each branching point of the stemma. Assuming a point of origin, or a root, in the otherwise undirected stemma tree, thus directly suggests a reconstruction of the most original version.

8.5 Conclusions

We proposed a new compression-based criterion, and an associated algorithm for computer assisted stemmatic analysis. The method was applied to the tradition of the legend of St. Henry of Finland, of which some fifty manuscripts are known. Even for such a moderate number, manual stemma reconstruction is prohibitive due to the vast number of potential explanations, and the obtained stemma is the first attempt at a complete stemma of the legend of St. Henry. The relationships discovered by the method are largely supported by more traditional analysis in earlier work, even though we have thus far only used a part of the legend in our experiments. Moreover, our results have pointed out groups of manuscripts not noticed in earlier manual analysis. Consequently, they have contributed to research on the legend of St. Henry carried out by historians and helped in forming a new basis for future studies. Trying to reconstruct the earliest version of the text and the direction of the relationships between the nodes in the stemma is an exciting line of research where a combination of stemmatological, palaeographical, codicological and content based analysis has great potential.

Appendix A: Comparison with the CompLearn package

The CompLearn package [21] (Section 4.7.2) performs similar analysis as our method in a more general context where the strings need not consist of word-by-word aligned text. Recall that it is based on the Normalized Compression Distance (NCD) defined as in (3.5.1), for convenience restated,

$$\mathrm{NCD}(x,y) = \frac{\max\{C(x \mid y), C(y \mid x)\}}{\max\{C(x), C(y)\}},$$

that was developed and analyzed in [9, 10, 20, 22, 79] (Chapter 3). Both our minimum information criterion and NCD are based on (approximations of) Kolmogorov complexity. The core

method in CompLearn uses a quartet tree heuristic in order to build a bifurcating tree with the observed strings as leafs [24] (Chapter 5).

In contrast to our method, where the cost function involves the contents of both the observed strings in the leaves and the unobserved interior nodes, CompLearn only uses the pairwise NCD distances between the observed strings (in [40] the latter kind of methods are called distance matrix methods).

The relation between NCD and the criterion presented in this work may be made more clear by considering the sum-distance $C(y \mid x) + C(x \mid y)$. Bennett *et al.* [9] show that the sum-distance is sandwiched between the numerator of NCD and two times the same quantity, ignoring logarithmic terms:

$$\max\{C(x \mid y), C(y \mid x)\} \leq C(y \mid x) + C(x \mid y) \leq 2\max\{C(x \mid y), C(y \mid x)\}. \tag{8.5.1}$$

Assuming that $C(x,y) \approx C(y,x)$ for all x,y, the sum-distance yields the cost

$$\sum_{(v,w)\in E} C(w \mid v) + C(v \mid w) = 2 \sum_{(v,w)\in E} C(v,w) - 3 \sum_{v\in V_I} C(v) - \sum_{w\in V_L} C(w),$$

where the summations are over the set of edges E, the set of interior nodes V_I, and the set of leaf nodes V_L, respectively. Since the set of leaf nodes is constant in the phylogenetic reconstruction problem, the last term can be ignored. Comparing the first two terms with (8.2.2) shows that the only difference is in the ratio of the factors of the first two terms (2 : 3 above; 1 : 2 in (8.2.2)). Thus, the difference between the the sum-distance and the information cost depends only on the variation of $C(v)$: if all strings are of roughly the same complexity, the difference is small. On the other hand, the difference between the sum-distance and NCD results, up to a factor of two (inequality (8.5.1)), from the normalization by $\max\{C(x), C(y)\}$ in NCD . Thus, if all strings are equally complex, the sum-distance and NCD do not differ 'too much', which in turn implies, *summa summarum*, that the information cost and NCD agree, at least roughly. However, in our case, many of the variants are partially destroyed, and consequently the complexity of the existing texts varies. The difference between the quartet tree heuristic and the Sankoff-style algorithm (Section 8.3) is more difficult to analyze, but clearly, both are designed for the same purpose.

Figure 8.5 shows the tree obtained by CompLearn using a blocksort approximation to Kolmogorov complexity (see the documentation of CompLearn for more information). The tree agrees at least roughly in many places with the tree in Fig. 8.3, for instance, the expected pairs Ho–T, JB–NR2, D–AJ, JG–B, MN–Y, BA–BS, and LT–E are next to or almost next to each other in both trees. We plan to investigate whether the remaining differences between the two trees are due to the cost functions, the search methods, or other features of the methods. At any rate, such agreements corroborate the validity of both methods and provide yet stronger support for the results.

It should be realized that the structure of the output trees in the two cases are different. This is due to differences in constraints and assumptions as well as explicit algorithmic differences as already noted. Thus, not too many conclusions should be drawn from this rough comparison.

Table 8.1. Estimated time of writing and place of origin (alternative place in parentheses) from [47], and total number of words in Sections 1,4,5, and 6.

Code	Time	Place	# of Words
A	1st half of 14th c.	Finland (/Sweden)	364
Ab	14th c.	Finland	7
AJ	1416–1442	Vadstena	185
B	ca. 1460	Cologne	336
BA	1513	Västeröas	185
Bc	15th c.	Sweden	250
BL	1493	Linköping	246
BLu	1517	Lund	185
BS	1498	Skara	185
BSt	1495	Strängnäs	189
BU	1496	Uppsala	329
C	14th to 15th c.	Sweden	375
Cd	15th c.	Sweden (/Finland)	102
CP	1462–1500	Vadstena	59
D	1446–1460	Vadstena	181
De	15th c.	Växjö (/Sweden)	95
Dr	end of 14th c.	Linköping (/Växjö)	371
E	1442–1464	Vadstena	237
Ef	end of 14th c. / beginning of 15th c.	Sweden (/Finland)	82
F	1st half of 15th c.	Vadstena (/Linköping)	339
Fg	14th c.	Finland (Sweden)	44
G	1476–1514	Vadstena	251
Gh	14th c.	Sweden (/Finland)	97
H	end of 14th c. / beginning of 15th c.	Finland	74
Ho	after 1485	Hollola	371
I	end of 15th c. / beginning of 16th c.	Ikaalinen	267
JB	1428–1447	Vadstena	166
JG	ca. 1480	Brussels	341
K	end of 15th c. / beginning of 16th c.	Kangasala	372
L	15th c.	Sweden	132
Li	2nd half of 15th c.	Vadstena	193
LT	1448–1458	Vadstena	266
Lu	1st half of 14th c.	Sweden	149
M	1st half of 15th c.	Bishopric of Linköping	228
MN	1495	Vadstena	372
N	15th c.	Finland	373
NR	1476–1514	Vadstena	0
NR2	after 1489	Vadstena	158
O	middle 14th c.	Ösmo (/Uppsala)	182
P	ca. 1380	Strängnäs (/Vadstena)	379
Q	2nd half of 15th c., before 1493	Bishopric of Linköping (/Vadstena)	176
R	15th c.	Finland	267
S	1st half of 15th c.	Finland	370
St	beginning of 15th c.	Bishopric of Strängnäs (/Sweden)	211
T	ca. 1485	Finland	373
U	15th c.	Uppsala	154
V	1485	Bishopric of Uppsala	301
Vae	14th c.	Sweden (/Finland)	247
Vg	end of 14th c. / beginning of 15th c.	Sweden (/Finland)	33
X	middle or late 15th c.	Bishopric of Uppsala	188
Y	ca. 1500	Vadstena (/Linköping)	372
Z	15th c.	Sweden (/Finland)	10

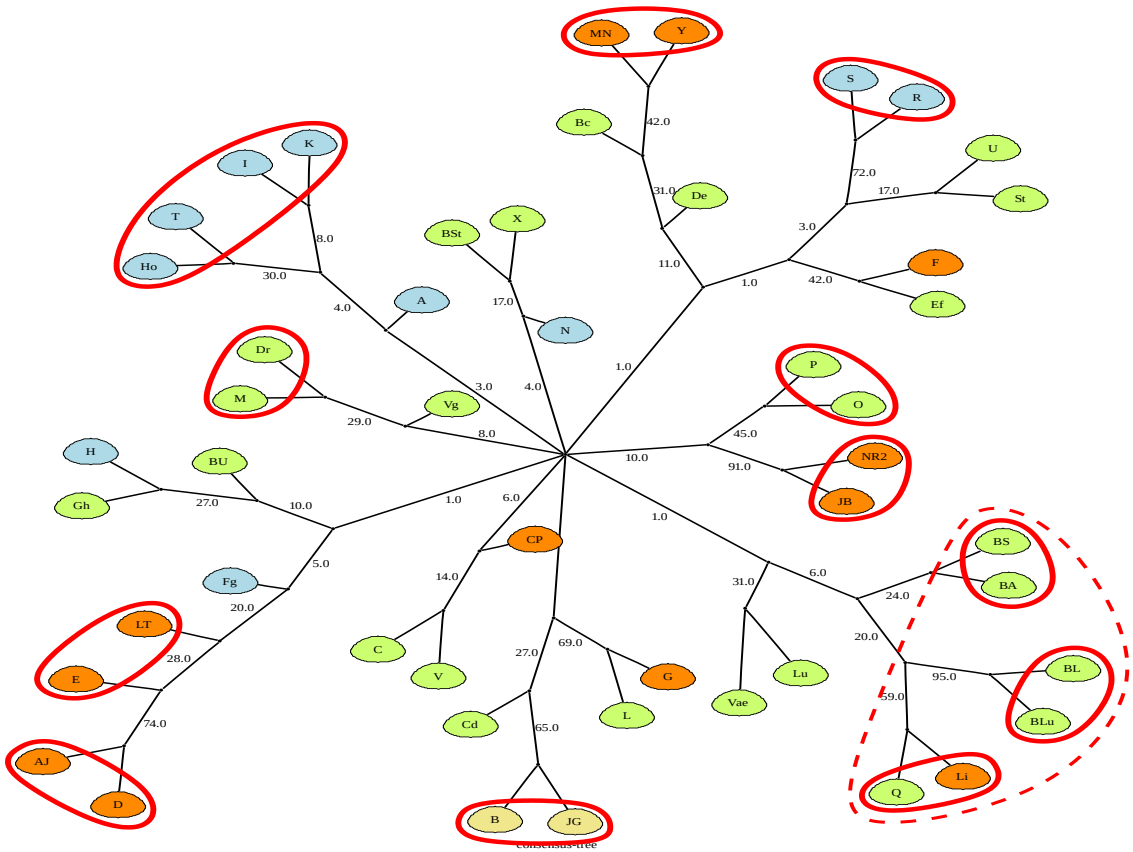

Figure 8.4: Consensus tree. The numbers on the edges indicate the number of bootstrap trees out of 100 where the edge separates the two sets of variants. Large numbers suggest high confidence in the identified subgroup. Some groups supported by earlier work are circled in red.

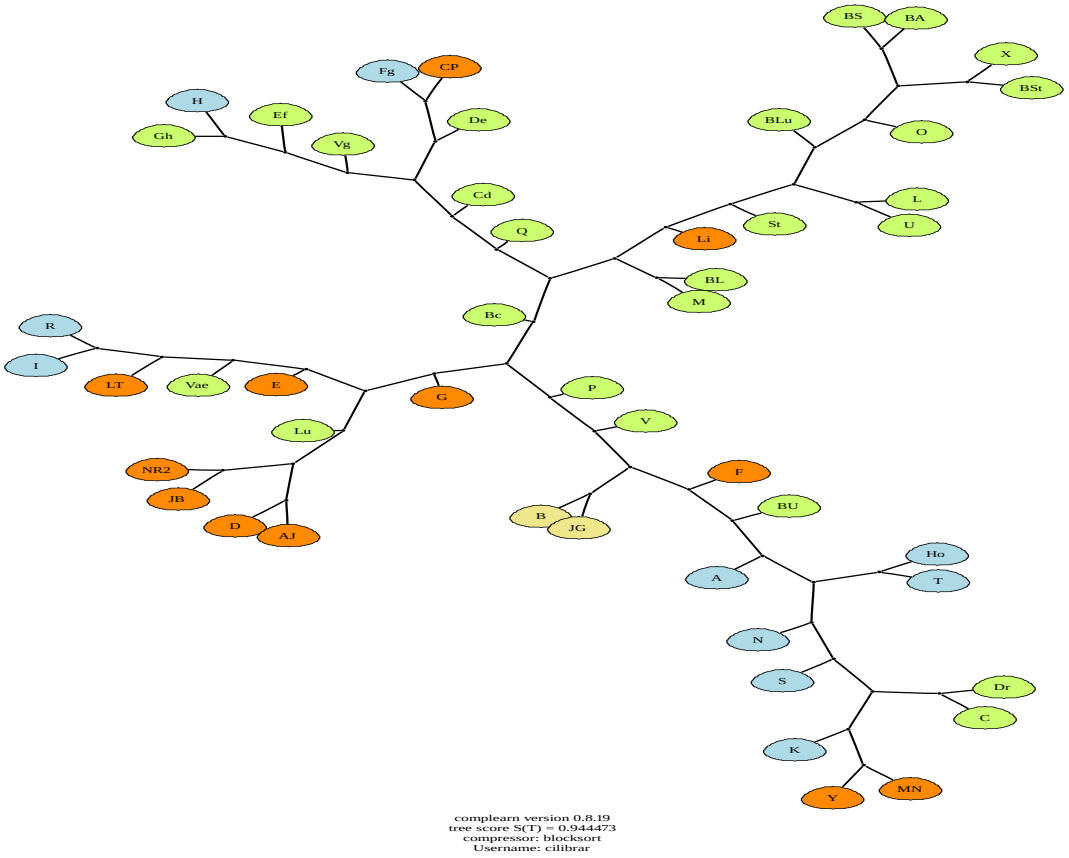

Figure 8.5: CompLearn tree showing many similarities with the tree in Fig. 8.3.

Chapter 9

Comparison of CompLearn with PHYLIP

In tree reconstruction systems like PHYLIP, trees are constructed in a greedy fashion using a randomly permuted ordering. Many different permutations are sampled and tree-splits are classified according to how consistently they come up with a score from 0 to 100 indicating a percentage agreement. This results in a so-called "consensus tree". While this method is far quicker than ours for large trees, it is probably in certain situations more error prone. It does not directly address the whole tree as one structured object but instead says something about the most likely paths where trees are iteratively increased one step at a time. Another difficulty of this approach is that sometimes bootstrap values are just too low, less than 50 for all splits, in which case PHYLIP's answer is considered too suspicious to use. In our technique this is rarely the case because we start with a random tree and make a monotonic sequence of nondeterministic steps to improve it. Our algorithm will always produce something better than random in practice and typically produces something reasonably good for natural data. Another problem with PHYLIP and the like is that there are many parameters and adjustments that must be made in order to properly calculate distances in the matrix creation phase; further, there are many choices of parameters in the tree-building phase so it takes more effort to use.

The most difficult part is a multiple-alignment phase that typically involves a biology expert. For comparison, The kitsch program in PHYLIP was run with the H5N1 distance matrix to build the tree. The kitsch program is generally considered the most accurate in the PHYLIP suite. Using a random seed value of 1 and generating 100 different random orderings for 100 different trees of 100 nodes yields the consensus tree of Figure 9.1. A casual observation and comparison with Figure 4.10 indicates that this tree is largely similar but not identical to the CompLearn output. This is attributed to the fact that the $S(T)$ score is quite high suggesting that the input data distance matrix has very good projection to a binary tree without substantial distortion and thus serves as a relatively "easy" case of the difficult problem domain. CompLearn shows a more clear advantage in the next experiment involving 102 Nobel prize winning writers. Here, NGD is used to generate the distance matrix in a quantitative way similar to the newly popular approaches such as Moretti's [84]:

> Theories are nets, and we should learn to evaluate them for the empirical data they allow us to process and understand: for how they concretely change the way we

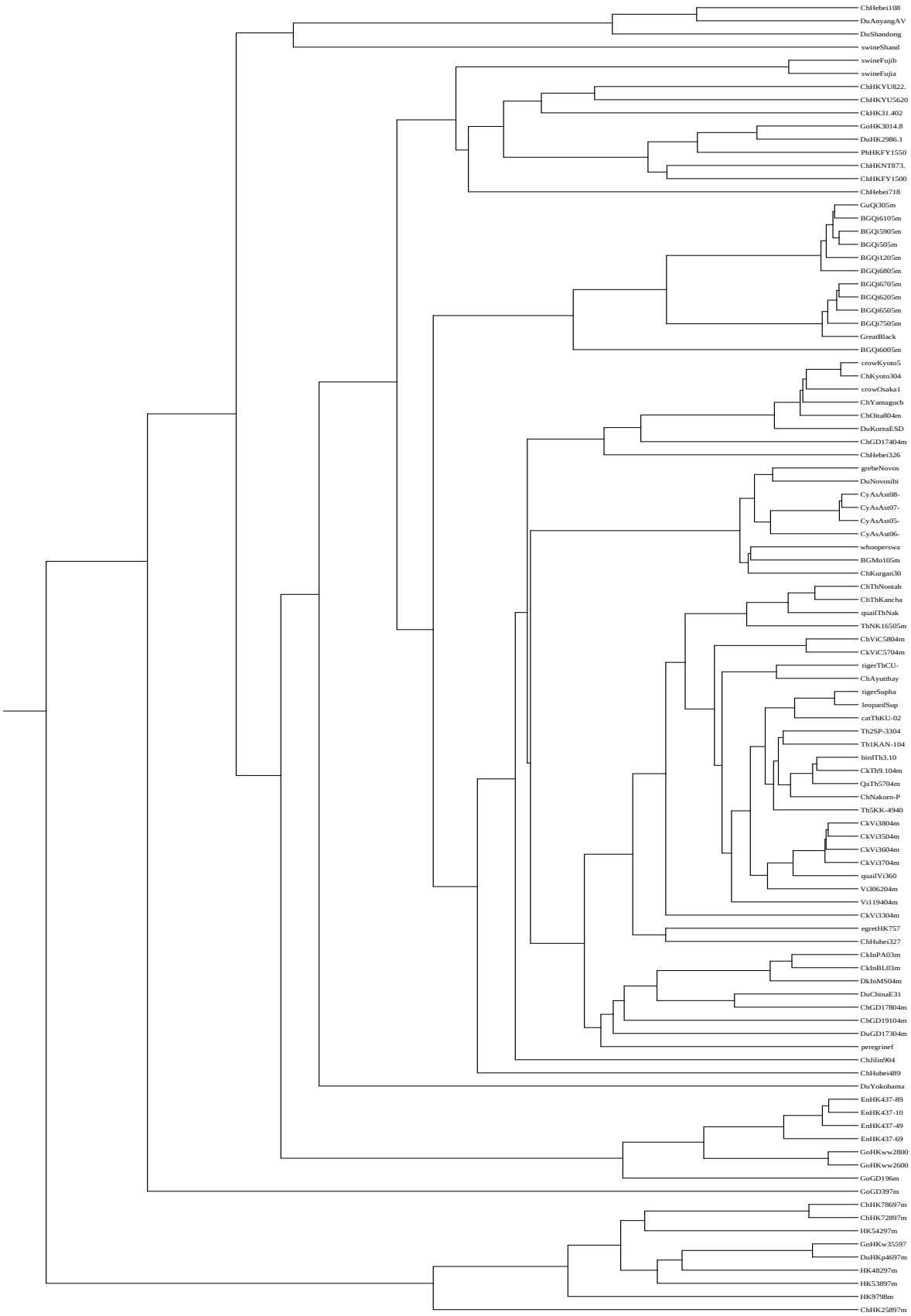

Figure 9.1: Using the **kitsch** program in PHYLIP for comparison of H5N1 tree.

> work, rather than as ends in themselves. Theories are nets; and there are so many interesting creatures that await to be caught, if only we try. – Moretti

Here, the maximal score is not as high as the H5N1 tree. The PHYLIP package does not cope as well with the ambiguities and seems to produce a much quicker yet obviously lower quality tree as shown below. The clustering of American and British writers is more scattered. This run used 100 jumbles (random permutations) to form the consensus. This strategy of simple random permutation based sampling serves to increase the effective power of PHYLIP considerably, but does not work very well to get high-accuracy plots of high-inconsistency data which often occur in practice.

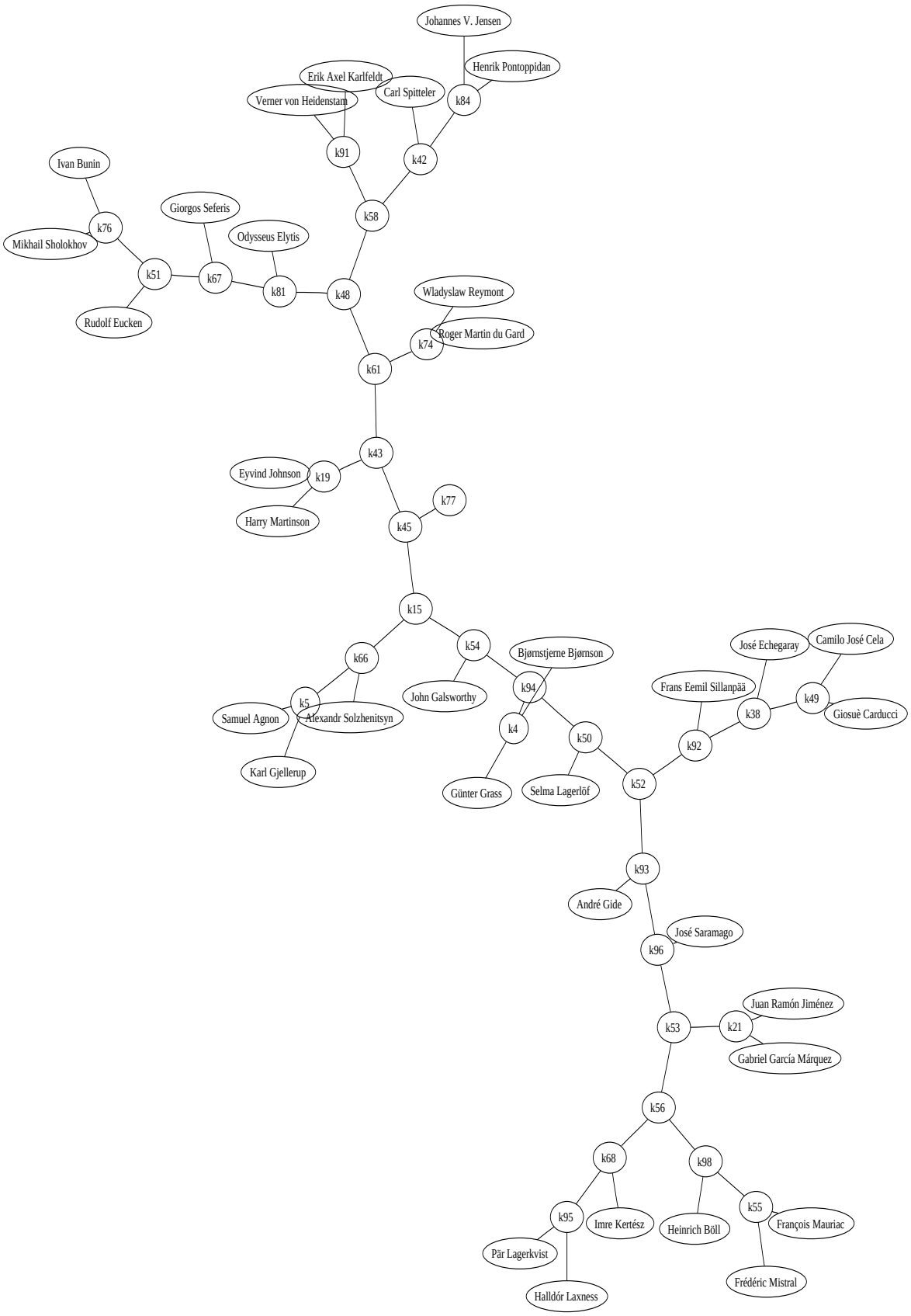

Figure 9.2: 102 Nobel prize winning writers using CompLearn and NGD; S(T)=0.905630 (part 1).

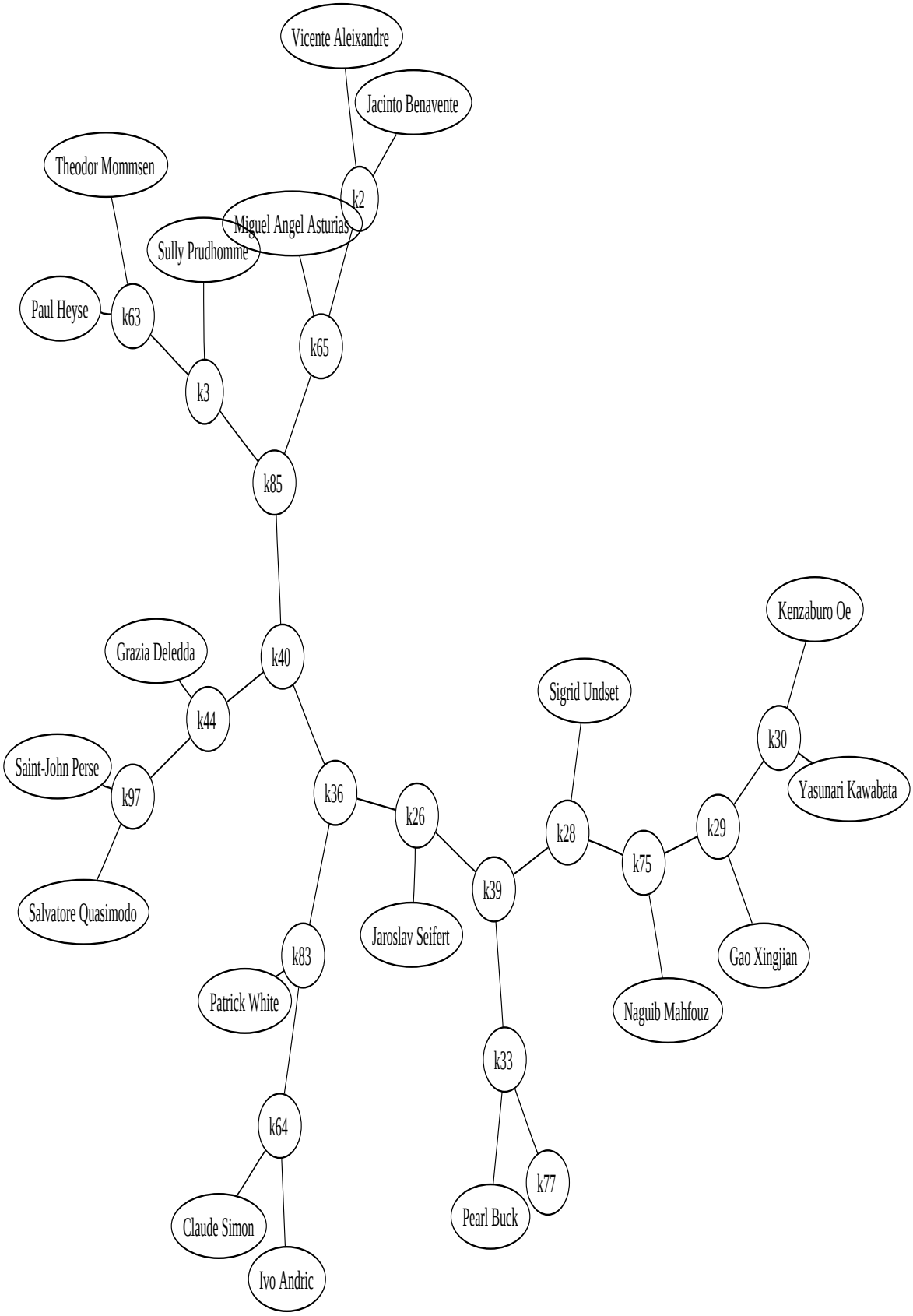

Figure 9.3: 102 Nobel prize winning writers using CompLearn and NGD; S(T)=0.905630 (part 2).

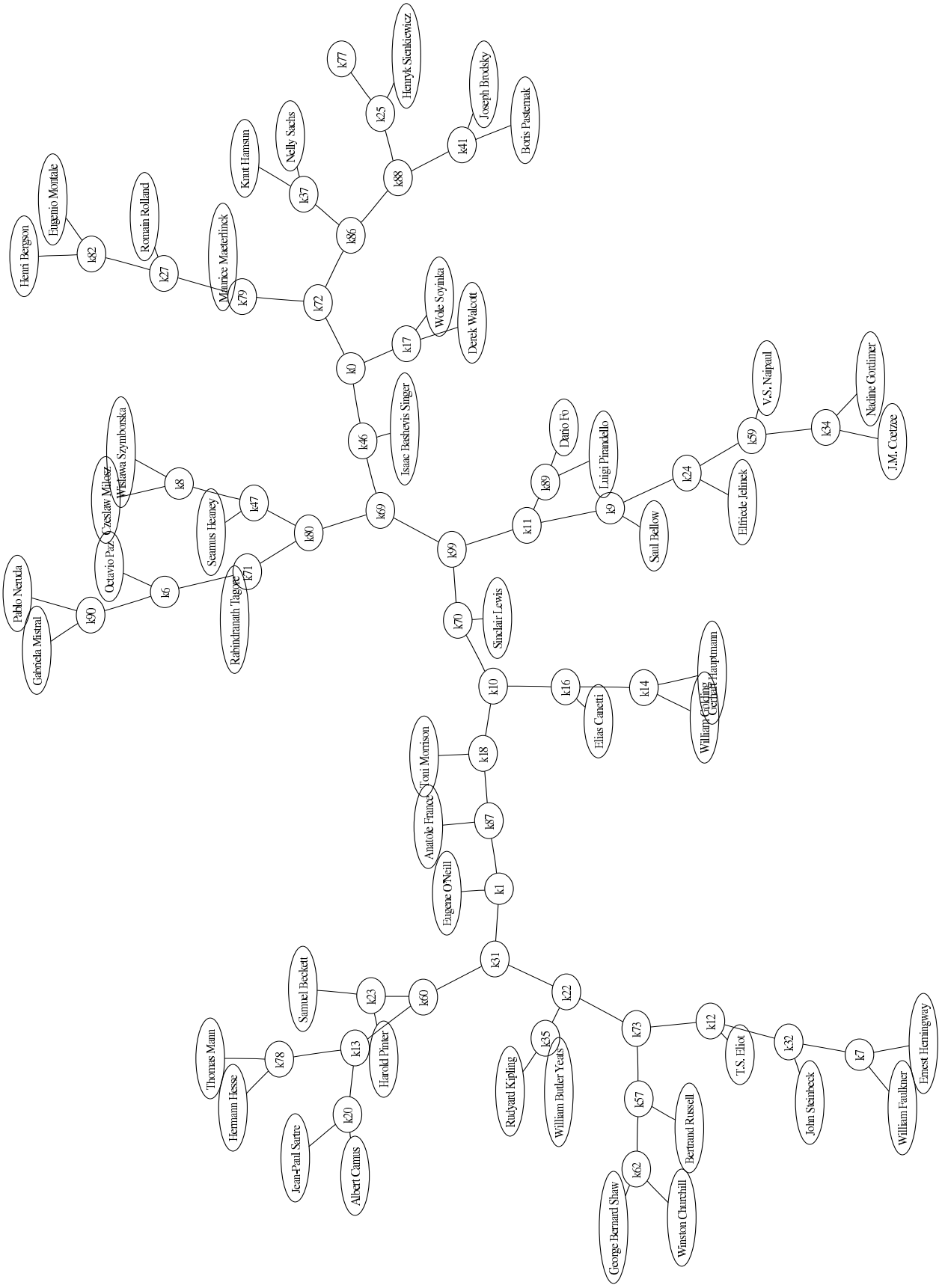

Figure 9.4: 102 Nobel prize winning writers using CompLearn and NGD; S(T)=0.905630 (part 3).

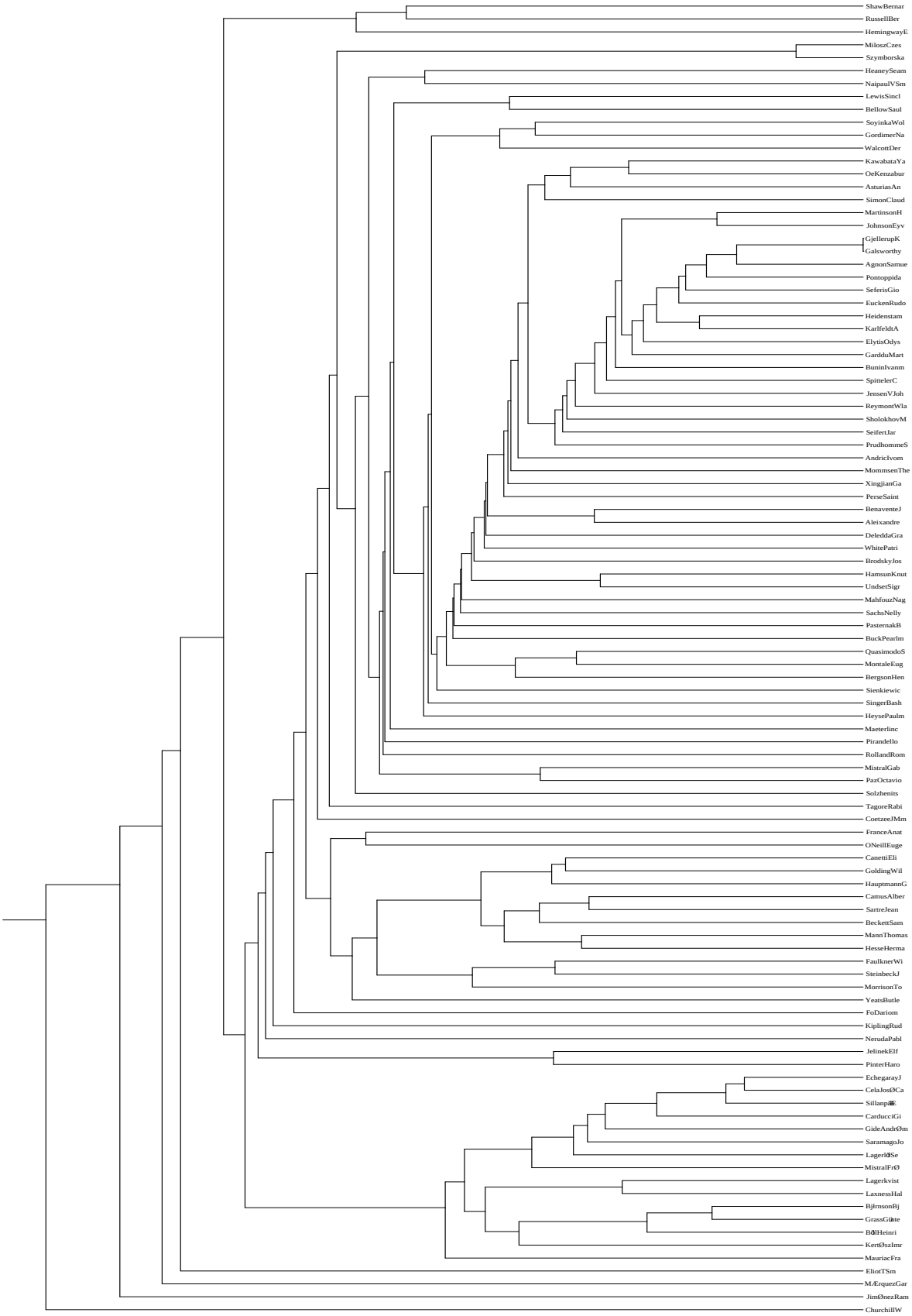

Figure 9.5: 102 Nobel prize winning writers using the PHYLIP **kitsch**.

Chapter 10

CompLearn Documentation

This chapter presents an overview of the CompLearn system. It provides a functional description from a high level followed by a detailed manual for basic installation and use. The material in this chapter will be useful for those wanting to run their own experiments using the CompLearn data mining system.

We start with a current snapshot of some important documentation for the CompLearn software system is presented. It is the core software behind almost all experiments mentioned in this thesis and represents a first general purpose open source data compression based learning system. The following documentation is literally copied from the online website and the reader is referred to `http://complearn.org/` for more info.[1]

What is CompLearn?

CompLearn is a suite of simple-to-use utilities that you can use to apply compression techniques to the process of discovering and learning patterns.

The compression-based approach used is powerful because it can mine patterns in completely different domains. It can classify musical styles of pieces of music and identify unknown composers. It can identify the language of bodies of text. It can discover the relationships between species of life and even the origin of new unknown viruses such as SARS. Other uncharted areas are up to you to explore.

In fact, this method is so general that it requires no background knowledge about any particular classification. There are no domain-specific parameters to set and only a handful of general settings. Just feed and run.

Installation Installation

CompLearn was packaged using **Autoconf**. Installation steps:

```
$ ./configure
$ make
$ make install
```

[1] It should be mentioned that in recent time the CompLearn system has had an improvement made which is not explained in the previously published literature. It concerns the dotted lines around the edge nodes on trees with numerical scores in the middle, see Chapter 7, Section 7.5.3.

To view installation options:

```
$ ./configure --help

complearn(5)

NAME
complearn - file format for the complearn configuration file
SYNOPSIS
$HOME/.complearn/config.yml - controls options for CompLearn

DESCRIPTION

The CompLearn toolkit is a suite of utilities to analyze arbitrary data. The
commands ncd(1), maketree(1), all use the config.yml configuration file. First,
the user home directory ($HOME) is searched for a directory called .complearn.
Within this directory, CompLearn reads a file called config.yml, a text file
structured in the YAML format. If CompLearn can not locate this file, default
values are used. The path to a configuration file may also be specified by the
-c option. A configuration file specified in this manner overrides all other
options.

For more information on the CompLearn project, please see
http://www.complearn.org

The format of this file is as follows:

<VariableName>: <value>

Blank lines are allowed. Comments are designated with a # sign. Variables come
in one of four types: boolean, integer, or string.

The following VariableNames are valid.

compressor: string (ncd (1))
    The builtin compressor to be used. Valid values are: bzip, zlib, and google.

GoogleKey: string (ncd (1), google compressor)
    Key necessary to perform search queries against Google database. Can be
obtained directory from Google at http://www.google.com/apis/.

blocksize: int (ncd (1), bzip compressor)
    An integer from 1 to 9. 9 gives the best compression but takes the most
```

memory. Default 9.

workfactor: int (ncd (1), bzip compressor)
    An integer from 0 to 250 and controls how the compression phase behaves when presented with the worst case, highly repetitive, input date. CompLearn's default value of 30 gives reasonable behavior over a wide range of circumstances.

bzverbosity: int (ncd (1), bzip compressor)
    An integer from 0 and 4. 0 is silent and greater numbers give increasingly verbose monitoring/debugging output. Default 0.

zliblevel: int (ncd (1), zlib compressor)
    An integer from 1 to 9. 1 is the fastest and produces the least compression. 9 is the slowest and produces the most compression. Default 9.

isRooted: bool (maketree (1))
    Whether or not to create a rooted binary tree. Default 0.

isOrdered: bool (maketree (1))
    Whether or not to order the nodes. Default 0.

selfAgreementTermination: bool (maketree (1))
    Whether or not to insist k number of trees must reach an agreed score before the program exits. Default 1.

maxFailCount: int (maketree (1))
    An integer specifying how many failed batches of trees must occur in succession before the program exits. Only used when selfAgreementTermination is off. Default 100000.

EXAMPLE

```
#
# comments are written like this
#
GoogleKey:                A/OGsJTQFHSpufko/rRS/KLA7NAT8UNf
compressor:               bzip
blocksize:                5
workfactor:               100
isRooted:                 1
selfAgreementTermination: 0
```

```
maxFailCount:             50000
# etc



FILES
$HOME/.complearn/config.yml

    configuration file, overrides system default
```

```
Usage: ncd [OPTION] ... [FILE | STRING | DIR ] [FILE | STRING | DIR]

ENUMERATION MODES:
  -f, --file-mode=FILE        file; default mode
  -l, --literal-mode=STRING   string literal
  -p, --plainlist-mode=FILE   list of file names by linebreaks
  -t, --termlist-mode=FILE    list of string literals separated by linebreaks
  -d, --directory-mode=DIR    directory of files
  -w, --windowed-mode=FILE,firstpos,stepsize,width,lastpos
                              file be separated into windows

NCD OPTIONS:
  -C, --compressor=STRING     use builtin compressor
  -L, --list                  list of available builtin compressors
  -g, --google                use Google compression (NGD)
  -D, --delcache              clear the Google cache
  -o, --outfile=distmatname   set the default distance matrix output name
  -r, --realcomp=pathname     use real compressor, passing in pathname of
compressor

OPTIONS:
  -c, --config-file=FILE      in YAML format
  -S, --size                  compressed size 1 FILE, STRING or DIR
  -x, --exp                   print out 2^val instead of val
  -B, --binary                enable binary output mode
  -P, --svd-project           output a singular value decomposition matrix
  -s, --suppress              suppress ASCII output
  -b, --both                  enable both binary and text output mode
  -H, --html                  output in HTML format
  -P, --svd-project           activate SVD projection mode
  -r, --suppressdetails       do not print details to dot file
  -V, --version
  -v, --verbose
  -h, --help
```

```
Usage: maketree [OPTION] ... FILE

MAKETREE OPTIONS:
  -o, --outfile=treename      set the default tree output name
  -R, --rooted                create rooted tree
  -O, --ordered               create ordered tree
  -T, --text-input            format of distance matrix is text
  -F                          disable self agreement termination and enable
                              max fail count

OPTIONS:
  -c, --config-file=FILE      in YAML format
  -S, --size                  compressed size 1 FILE, STRING or DIR
  -x, --exp                   print out 2^val instead of val
  -B, --binary                enable binary output mode
  -P, --svd-project           output a singular value decomposition matrix
  -s, --suppress              suppress ASCII output
  -b, --both                  enable both binary and text output mode
  -H, --html                  output in HTML format
  -P, --svd-project           activate SVD projection mode
  -r, --suppressdetails       do not print details to dot file
  -V, --version
  -v, --verbose
  -h, --help
```

CompLearn FAQ

1. What is CompLearn?

CompLearn is a software system built to support compression-based learning in a wide variety of applications. It provides this support in the form of a library written in highly portable ANSI C that runs in most modern computer environments with minimal confusion. It also supplies a small suite of simple, composable command-line utilities as simple applications that use this library. Together with other commonly used machine-learning tools such as LIBSVM and GraphViz, CompLearn forms an attractive offering in machine-learning frameworks and toolkits. It is designed to be extensible in a variety of ways including modular dynamic-linking plugins (like those used in the Apache webserver) and a language-neutral SOAP interface to supply instant access to core functionality in every major language.

2. Why did the version numbers skip so far between 0.6.4 and 0.8.12?

In early 2005 a major rewrite occurred. This was due to poor organization of the original complearn package, leading to compilation and installation difficulties in far too many situations. This issue was addressed by using a complete rewrite from the ground up of all functionality; earlier versions used a combination of C and Ruby to deliver tree searching. The new version delivers all core functionality, such as NCD and tree searching, in a pure C library. On top of this library is layered a variety of other interfaces such as SOAP and a new in-process direct-extension CompLearn Ruby binding layer. But all dependencies have been reworked and are now modularized so that Ruby and almost every other software package is now optional and a variety of different configurations will compile cleanly.

Another major enhancement in the new complearn is the addition of a Google compressor to calculate NGD. This has opened up whole new areas of Quantitative Subjective Analysis (QSA) to complement our existing more classically pure statistical methods in earlier gzip-style NCD research. By querying the Google webserver through a SOAP layer we may convert page counts of search terms to virtual file lengths that can be used to determine semantic relationships between terms. Please see the paper Automatic Meaning Discovery Using Google for more information.

3. I can’t get the Google compressor to work. When I type ncd -L, it’s not even listed as one of the builtin compressors. What am I doing wrong?

You may not have the csoap library installed, which is necessary for the Google compressor to work. You can check this when you run your ./configure command during the CompLearn installation phase. A "NO" in the CompLearn dependency table for csoap indicates you need to install the csoap library.

You can download csoap at the following link:

http://sourceforge.net/project/showfiles.php?group_id=74977

Once csoap is installed, you will need to run the ./configure command again (for CompLearn), perhaps with a "--with-csoap" option depending on the location of the csoap installation. For more options, you can type

./configure --help

Please see our Dependencies section for more information on CompLearn library dependencies.

4. The Windows demo isn’t working for me? Why not?

If you have cygwin installed on your computer, it’s very likely you need to update it. The CompLearn Windows demo uses version 1.5.17 of the cygwin dll; any previous versions are not compatible with the demo. To update your cygwin, go to http://cygwin.com and hit the Install or Update now link.

You may also need to download and install DirectX.
5. gsl and CompLearn seemed to install perfectly, but ncd can’t load the gsl library.

Users may get the following message if this happens:

ncd: error while loading shared libraries: libgslcblas.so.0: cannot
open shared object file: No such file or directory

If this is the case, your LD_LIBRARY_PATH environment variable may need to be set. For example, you can try the following before running the ncd command:

export LD_LIBRARY_PATH=/usr/local/lib

6. How can this demo work with only 1000 queries a day?

There are two reasons this demo is able to do as much as it does. One is that Google has generously (and free of charge to me) upgraded my Google API account

key daily search limit. You might email them to ask yourself if you have an interesting Google API based search application of your own. The other reason the demo works is because there is a cache of recent page result counts. You can see this cache by looking in the $HOME/.complearn directory. Sometimes larger experiments must be run over the course of two days.

7. How come the counts returned from (any particular) Google API are different that the numbers I see when I enter searches by hand?

I have two possible explanations for this behavior. One is that it would be prohibitively expensive to count the exact total of all pages indexed for most common search terms. Instead they use an estimation heuristic called "prefixing" whereby they just use a short sample of webpages as a representative set for the web and scale up as appropriate. I presume this and also that when you do a search (either by hand or from the API) you can get connected to any one of a number of different search servers, each with a slightly different database. In a rapidly changing large global network it is unlikely that there will be an exact match for the counts on any particular common term because each server must maintain its own distinct "aging snapshot" of the internet.

8. When I compile csoap, I don’t seem to be getting shared libraries. Or even though csoap is installed, complearn doesn’t seem to be detecting the shared library.

Try compiling csoap from source with the following options:

```
   --with-libxml-prefix=/usr --enable-shared --with-pic
```

Then try reconfiguring and recompiling complearn.

Thanks to Tsu Do Nimh for this tip.

9. Is it important to adjust or choose a compressor? How should I do it?

Yes, it is very important to choose a good compressor for your application. The "blocksort" compressor is the current default. It is a virtual compressor using a simple blocksorting algorithm. It will give results something like frequency analysis, spectral analysis, and substring matching combined. It works very well for small strings (or files) of 100 bytes or less. If you have more than about 100 bytes then it is probably better to use one of the other three favorite compressors other than the default:

```
ncd -C zlib
```

will get you "zlib" style compression which is like gzip and is limited to files of up to 15K in size.

```
ncd -C bzip
```

will get you "bzip2" style compression which is like zlib but allows for files up to about 450K in size. The best accuracy is available using the "real compressor" shell option. For this to work you need to use a script like this:

```
#!/bin/bash
cd /tmp
cat >infile
/ufs/cilibrar/bin/ppmd e infile >/dev/null </dev/null 2>/dev/null
cat infile.pmd
rm infile infile.pmd
```

If you install that script in $HOME/bin/catppmd and don’t forget to chmod it executable (using chmod a+rx $HOME/bin/catppmd for instance) then you can use it with the following option to ncd:

```
ncd -r $HOME/bin/catppmd
```

10. Running ./configure gives me the following error: cannot find input file: src/complearn/aclconfig.h.in. Where can I find this file?

You will need to generate this header input file by running the autoheader command. autoheader is packaged with autoconf.

```
autoheader
```

11. I get the configure error: Can’t locate object method "path" via package "Request" at /usr/share/autoconf/Autom4te/C4che.pm line 69, line 111. make[1]: *** [configure] Error 1. Is there an easy way to fix this?

In the top directory of the CompLearn distribution, run the following commands:

```
rm -rf autom4te.cache
```

or

```
make maintainer-clean
```

Bibliography

[1] M.J. Alberink, L.W. Rutledge, and M.J.A. Veenstra. Clustering semantics for hypermedia presentation. Technical Report INS-E0409, CWI, 2004. ISSN 1386-3681. 7.5.2

[2] W. Haken Appel, K. I. and J. Koch. *Every planar map is four colorable, Part I: Discharging*, volume 21. Illinois Journal of Mathematics, 1977. 1.2

[3] Ph. Ball. Algorithm makes tongue tree. *Nature*, Jan 2002. 6.1

[4] T. Bell, J. Cleary, and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984. 2.8

[5] T. Belloni, M. Klein-Wolt, M. Méndez, M. van der Klis, and J. van Paradijs. A model-independent analysis of the variability of GRS 1915+105. A&A, 355:271–290, March 2000. (document), 6.9, 6.15

[6] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg. Constructing phylogenies from quartets: Elucidation of eutherian superordinal relationships. *J. Computational Biology*, 5(3):377–390, 1998. 4.2

[7] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and zipping. *Physical Review Letters*, 88(4):048702–1–048702–4, 2002. 8.2

[8] D. Benedetto, E. Caglioti E., and V. Loreto. Language trees and zipping. *Physical Review Letters*, 88(4), 2002. 6.1, 6.6

[9] C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4):1407–1424, 1998. 3.1, 3.3, 4.7.1, 6.1, 7.1, 7.1.2, 8.5

[10] C.H. Bennett, M. Li, and B. Ma. Chain letters and evolutionary histories. *Scientific American*, pages 76–81, June 2003. 4.7.1, 6.1, 7.1.2, 8.2, 8.5

[11] Alexander J. Smola Bernhard Scholkopf. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001. ISBN 0262194759. 5.3, 5.3.2, 5.3.3

[12] V. Berry, T. Jiang, P. Kearney, M. Li, and T. Wareham. Quartet cleaning: improved algorithms and simulations. In *Algorithms–Proc. 7th European Symp. (ESA99), LNCS vol. 1643*, pages 313–324, Berlin, 1999. Springer Verlag. 4.2, 4.5.1, 4.5.1, 4.6.2

[13] Vincent Berry, David Bryant, Tao Jiang, Paul E. Kearney, Ming Li, Todd Wareham, and Haoyong Zhang. A practical algorithm for recovering the best supported edges of an evolutionary tree (extended abstract). In *Symposium on Discrete Algorithms*, pages 287–296, 2000. 6.1, 6.5.1

[14] Jose Lambert Broek Raymond van den., James S. Holmes. *The concept of equivalence in translation theory: Some critical reflections*. Cuellar Universidad, 1978. 1.2

[15] P. Buneman. The recovery of trees from measures of dissimilarity. In F. Hodson, D. Kenadall, and P. Tautu, editors, *Mathematics in the Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, Edinburgh, Scottland, UK, 1971. 4.5.1, 4.6.3

[16] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. 5.3.3, 7.4.3

[17] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Systems Research Center, May 1994. 2.8

[18] Y. Cao, A. Janke, P. J. Waddell, M. Westerman, O. Takenaka, S. Murata, N. Okada, S. Pääbo, and M. Hasegawa. Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. *Journal of Molecular Evolution*, 47:307–322, 1998. 6.5.1

[19] W. Chai and B. Vercoe. Folk music classification using hidden Markov models. In *Proceedings of International Conference on Artificial Intelligence*, 2001. 6.1, 6.4

[20] X. Chen, S. Kwong, and M. Li. A compression algorithm for DNA sequences and its applications in genome comparison. In K. Asai, S. Miyano, and T. Takagi, editors, *Genome Informatics*, Tokyo, 1999. Universal Academy Press. 8.2, 8.5

[21] R. Cilibrasi, A.L. Cruz, and S. de Rooij. Complearn version 0.8.20, 2005. Distributed at `www.complearn.org`. 4.6.2, 4.6.4, 4.7.2, 4.10.1, 4.10.2, 8.4, 8.5

[22] R. Cilibrasi and P. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005. `http://www.archiv.org/abs/cs.CV/0312044` sec. 4.2, 4.7.1, 7.1, 7.1.2, 7.2.1, 7.6.4, 8.2, 8.5

[23] R. Cilibrasi and P.M.B. Vitányi. Automatic meaning discovery using google: 100 experiments in learning wordnet categories, 2004. http://www.cwi.nl/ ˜cilibrar/googlepaper/appendix.pdf sec. 7.1.3, 7.6.4

[24] R. Cilibrasi and P.M.B. Vitányi. A new quartet tree heuristic for hierarchical clustering. In *EU-PASCAL Statistics and Optimization of Clustering Workshop*, London, 2005. http://arxiv.org/abs/cs.DS/0606048 sec. 8.5

[25] R. Cilibrasi, P.M.B. Vitányi, and R. de Wolf. Algorithmic clustering of music based on string compression. *Computer Music Journal*, pages 49–67, 2003. http://xxx.lanl.gov/abs/cs.SD/0303025 sec. 4.2, 4.7.1, 4.10.2, 6.1, 7.1, 7.1.2, 7.2.1

[26] R. Cilibrasi, P.M.B. Vitányi, and R. de Wolf. Algorithmic clustering of music. In *Proc. IEEE 4th International Conference on Web Delivering of Music (WEDELMUSIC 2004)*, pages 110–117. IEEE Comp. Soc. Press, 2004. 4.2, 4.7.1, 4.10.2, 6.1, 7.1, 7.1.2, 7.2.1

[27] Human Civilization. United Nations General Assembly resolution 217 A (III) of 10 December 1948: Universal Declaration of Human Rights, http://www.un.org/Overview/rights.html sec. 6.6

[28] H. Colonius and H.-H. Schulze. Tree structures for proximity data. *British Journal of Mathematical and Statistical Psychology*, 34:167–180, 1981. 4.5.1

[29] H. Colonius and H.H. Schulze. Trees constructed from empirical relations. *Braunschweiger Berichte as dem Institut fuer Psychologie*, 1, 1977. 4.5.1

[30] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Symposium on Discrete Algorithms*, pages 197–206, 2000. 6.1

[31] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley & Sons, 1991. 3.1, 3.6.2, 3.6.2, 7.3.3, 7.3.5, 7.3.6

[32] Ido Dagan, Lillian Lee, and Fernando Pereira. Similarity-based methods for word sense disambiguation. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 56–63, Somerset, New Jersey, 1997. Association for Computational Linguistics. 3.6.2

[33] R. B. Dannenberg, B. Thom, and D. Watson. A machine learning approach to musical style recognition. In *In Proceedings of the 1997 International Computer Music Conference*, pages 344–347. International Computer Music Association, 1997. 6.1, 6.4

[34] Pedro Ponce De. Musical style identification using self-organising maps, 2004. http://citeseer.ist.psu.edu/638331.html sec. 6.1

[35] J.-P. Delahaye. Classer musiques, langues, images, textes et genomes. *Pour La Science*, 317:98–103, March 2004. 7.1

[36] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley Interscience, second edition, 2001. 4.3, 4.11

[37] G.A. Miller et.al. WordNet, A Lexical Database for the English Language, Cognitive Science Lab, Princeton University, `http://wordnet.princeton.edu/` sec. 7.1, 7.6.4

[38] Ding-Xuan Zhou3 Felipe Cucker, Steve Smale. *Foundations of Computational Mathematics*. Springer-Verlag New York, July 2004. 1.2

[39] J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *J. Molecular Evolution*, 17:368–376, 1981. 4.2

[40] J. Felsenstein. *Inferring phylogenies*. Sinauer Associates, Sunderland, Massachusetts, 2004. 8.3, 8.5

[41] J. Felsenstein. PHYLIP (Phylogeny inference package) version 3.6, 2004. Distributed by the author, Department of Genome Sciences, University of Washington, Seattle. 8.4, 8.4

[42] N. Friedman, M. Ninio, I. Pe'er, and T. Pupko. A structural EM algorithm for phylogenetic inference. *Journal of Computational Biology*, 9:331–353, 2002. 8.3, 5

[43] D. G. Goldstein and G. Gigerenzer. Models of ecological rationality: The recognition heuristic. *Psychological Review*, 109:75–90, 2002. 1.2

[44] M. Grimaldi, P. Cunningham, and A. Kokaram. Classifying music by genre using the wavelet packet transform and a round-robin ensemble, 2002. `http://www.cs.tcd.ie/publications/tech-reports/reports.02/TCD-CS-2002-64.pdf` sec. 6.1, 6.4

[45] S. Grumbach and F. Tahi. A new challenge for compression algorithms: genetic sequences. *Journal of Information Processing and Management*, 30(6):875–866, 1994. 8.2

[46] S. Haykin. *Neural Networks*. Prentice Hall, second edition, 1999. ISBN 0132733501. 5.3.1

[47] T. Heikkilä. *Pyhän Henrikin legenda* (in Finnish). Suomalaisen Kirjallisuuden Seuran Toimituksia 1039, Helsinki, 2005. (document), 8.1, 1, 8.4, 8.3, 8.5

[48] Paul G. Howard and Jeffrey Scott Vitter. Practical implementations of arithmetic coding. Technical Report CS-91-45, Brown University, 1991. 3.6.1

[49] C.J. Howe, A.C. Barbrook, M. Spencer, P. Robinson, B. Bordalejo, and L.R. Mooney. Manuscript evolution. *Trends in Genetics*, 17(3):147–152, 2001. 8.2, 3

[50] Google Inc. The basics of google search. http://www.google.com/help/basics.html sec. 7.4.2

[51] A. Janke, O. Magnell, G. Wieczorek, M. Westerman, and U. Arnason. Phylogenetic analysis of 18s rrna and the mitochondrial genomes of wombat, vombatus ursinus, and the spiny anteater, tachyglossus acelaetus: increased support for the marsupionta hypothesis. *Journal of Molecular Evolution*, 1(54):71–80, 2002. 6.5.1

[52] Nicholas Jardine and Robin Sibson. *Mathematical Taxonomy*. John Wiley & Sons, 1971. Wiley Series In Probabilistic And Mathematical Statistics. 3.6.2

[53] Tao Jiang, Paul E. Kearney, and Ming Li. A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM J. Comput.*, 30(6):1942–1961, 2000. 4.4

[54] E. Keogh, S. Lonardi, , and C.A. Rtanamahatana. Toward parameter-free data mining. In *Proc. 10th ACM SIGKDD Intn'l Conf. Knowledge Discovery and Data Mining*, pages 206–215, Seattle, Washington, USA, 2004. August 22–25, 2004. 4.9, 6.1

[55] J.K. Killian, T.R. Buckley, N. Steward, B.L. Munday, and R.L. Jirtle. Marsupials and eutherians reunited: genetic evidence for the theria hypothesis of mammalian evolution. *Mammalian Genome*, 12:513–517, 2001. 6.1, 6.5.1

[56] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. 4.6.2

[57] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1(1):1–7, 1965. 7.2.1, 8.2

[58] A.N. Kolmogorov. Combinatorial foundations of information theory and the calculus of probabilities. *Russian Math. Surveys*, 38(4):29–40, 1983. 7.2.1

[59] M. Koppel, S. Argamon, and A. Shimoni. Automatically categorizing written texts by author gender. *Literary and Linguistic Computing*, 17(3), 2003. 6.1

[60] L.G. Kraft. A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, Dept. of Electrical Engineering, M.I.T., Cambridge, Mass., 1949. 2.4.1

[61] A. Kraskov, H. Stögbauer, R.G. Adrsejak, and P. Grassberger. Hierarchical clustering based on mutual information. http://arxiv.org/abs/q-bio/0311039 sec, 2003. 6.1

[62] J.B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29:115–129, 1964. 4.11

[63] T.G. Ksiazek, D. Erdman, C.S. Goldsmith, S.R. Zaki, T. Peret, S. Emery, and et al. A novel coronavirus associated with severe acute respiratory syndrome. *New England J. Medicine*, 348:1953–66, 2003. 4.10.1, 6.5.2

[64] C.P. Kurtzman. Phylogenetic circumscription of saccharomyces, kluyveromyces and other members of the saccharomycetaceaea, and the proposal of the new genera lachnacea, nakaseomyces, naumovia, vanderwaltozyma and zygotorulaspora. *FEMS Yeast Res.*, 4:233–245, 2003. 6.5.3

[65] C.P. Kurtzman and J. Sugiyama. Ascomycetous yeasts and yeast-like taxa. In *The mycota VII, Systemtics and evolution, part A*, pages 179–200. Springer-Verlag, Berlin, 2001. 6.5.3

[66] H.R. Künsch. The jackknife and the bootstrap for general stationary observations. *Annals of Statistics*, 17(3):1217–1241, 1989. 8.4

[67] L. Lakshmanan and F. Sadri. Xml interoperability. In *Proc. Intn'l Workshop Web and Databases (WebDB)*, San Diego, California, June 2003. 5

[68] T. Landauer and S. Dumais. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychol. Rev.*, 104:211–240, 1997. 7.1.2

[69] A.-C. Lantin, P. V. Baret, and C. Macé. Phylogenetic analysis of Gregory of Nazianzus' Homily 27. In G. Purnelle, C. Fairon, and A. Dister, editors, *7èmes Journées Internationales d'Analyse statistique des Données Textuelles*, pages 700–707, Louvain-la-Neuve, 2004. 8.2

[70] P.S. Laplace. *A philosophical essay on probabilities*. self, 1819. English translation, Dover, 1951. 6.1

[71] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995. 7.1, 7.8

[72] M.E. Lesk. Word-word associations in document retrieval systems. *American Documentation*, 20(1):27–38, 1969. 7.1.2

[73] H. Levesque and R. Brachman. A fundamental tradeoff in knowledge representation and reasoning. *Readings in Knowledge Representation*, 1985. 1.2

[74] M. Li. Shared information distance or software integrity detection. Computer Science, University of California, Santa Barbara. `http://genome.math.uwaterloo.ca/SID/` sec. 4.7.1, 6.1

[75] M. Li, J. H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154, 2001. 4.7.1, 6.1, 7.1, 7.1.2

[76] M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi. The similarity metric. *IEEE Trans. Information Theory*, 50(12):3250–3264, 2004. 7.1, 7.1.2, 7.2.1

[77] M. Li, X. Chen, X. Li B. Ma, and P. Vitányi. The similarity metric. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, 2003. 3.1.8, 3.3, 3.3, 3.3, 3.5, 4.7.1, 6.1, 6.2, 6.5.1, 6.6

[78] M. Li and P.M.B. Vitányi. Reversibility and adiabatic computation: trading time and space for energy. *Proc. Royal Society of London, Series A*, 452:769–789, 1996. 7.1.2

[79] M. Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, second edition, 1997. 2, 2.7, 2.8, 3.1.2, 3.3, 3.3, 4.6.3, 4.7.1, 6.1, 7.1.2, 7.2.1, 7.3.5, 7.8, 8.2, 8.5

[80] M. Li and P.M.B. Vitányi. Algorithmic complexity. In N.J. Smelser and P.B. Baltes, editors, *International Encyclopedia of the Social and Behavioral Sciences*, pages 376–382. Pergamon, Oxford, 2001/2002. 4.7.1, 6.1, 6.5.1, 7.1.2

[81] T. Liu, J. Tang, and B.M.E. Moret. Quartet methods for phylogeny reconstruction from gene orders. Dept. CS and Engin., Univ. South-Carolina, Columbia, SC, USA. Manuscript. 4.2, 4.5.1

[82] D. Loewenstern, H. Hirsh, P. Yianilos, and M. Noordewier. DNA sequence classification using compression-based induction. Technical Report 95–04, DIMACS, 1995. 8.2

[83] N. Metropolis, A.W. Rosenbluth, and M.N. Rosenbluth. A.H. Teller and E. Teller. *J. Chem. Phys.*, 21:1087–1092, 1953. 4.6.2

[84] Franco Moretti. *Graphs, Maps, Trees: Abstract Models for a Literary History*. Verso Books, 2005. ISBN 1844670260. 9

[85] H. Muir. Software to unzip identity of unknown composers. *New Scientist*, 2003. 12 April 2003. 7.1

[86] H. Niman. Recombinomics website, 2006, `http://www.recombinomics.com/` sec. 4.10.1

[87] L. Oliveira, R. Sabourin, F. Bortolozzi, and C. Suen. Automatic recognition of handwritten numerical strings: A recognition and verication strategy. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(11):1438–1454, 2002. 6.8

[88] K. Patch. Software sorts tunes. *Technology Research News*, 2003. April 23/30, 2003. 7.1

[89] R. Piaggio-Talice, J. Gordon Burleigh, and O. Eulenstein. Quartet supertrees. In O.R.P. Beninda-Edmonds, editor, *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life. Computational Biology, volume 3 (Dress. A., series ed.)*, chapter 4, pages 173–191. Kluwer Academic Publishers, 2004. 4.2, 4.5.1

[90] S. L. Reed and D. B. Lenat. Mapping ontologies into cyc. In *Proc. AAAI Conference 2002 Workshop on Ontologies for the Semantic Web*, Edmonton, Canada, 2002. `http://citeseer.nj.nec.com/509238.html` sec. 7.8

[91] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978. 8.2

[92] J. Rissanen and G. G. Langdon, Jr. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, March 1979. 2.7, 3.6.1

[93] P. Robinson and R.J. O'Hara. Report on the textual criticism challenge 1991. *Bryn Mawr Classical Review*, 3(4):331–337, 1992. 8.2

[94] A. Rokas, B.L. Williams, N. King, and S.B. Carroll. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425:798–804, 2003. 25 October 2003. 4.10.3, 6.1, 6.5

[95] U. Roshan, B.M.E. Moret, T. Warnow, and T.L. Williams. Performance of supertree methods on various datasets decompositions. In O.R.P. Beninda-Edmonds, editor, *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life. Computational Biology, volume 3 (Dress. A., series ed.)*, pages 301–328. Kluwer Academic Publishers, 2004. 4.2

[96] D.H. Rumsfeld. The digital revolution. originally published June 9, 2001, following a European trip. In: H. Seely, The Poetry of D.H. Rumsfeld, 2003, `http://slate.msn.com/id/2081042/` sec. 7.1

[97] L. Rutledge, M. Alberink, R. Brussee, S. Pokraev, W. van Dieten, and M. Veenstra. Finding the story — broader applicability of semantics and discourse for hypermedia generation. In *Proc. 14th ACM Conf. Hypertext and Hypermedia*, pages 67–76, Nottingham, UK, 2003. August 23-27. 7.5.2, 7.5.3

[98] Boris Ryabko and Jaakko Astola. Application of data compression methods to hypothesis testing for ergodic and stationary processes. In *Discrete Mathematics and Theoretical Computer Science*, pages 1365–8050, Nancy, France, 2005. 1.2

[99] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987. 6.1, 6.5.1

[100] Economist Scientist. Corpus collosal: How well does the world wide web represent human language? *The Economist*, 2004. January 20, 2005. `http://www.economist.com/science/displayStory.cfm?story_id=3576374` sec. 7.2, 7.3

[101] P. Scott. Music classification using neural networks, 2001. `http://www.stanford.edu/class/ee373a/musicclassification.pdf` sec. 6.1, 6.4

[102] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical J.*, 27:379–423 and 623–656, 1948. 2.4.3

[103] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948. 2.6.3

[104] A. Siepel and D. Haussler. Phylogenetic estimation of context-dependent substitution rates by maximum likelihood. *Molecular Biology and Evolution*, 21(3):468–488, 2004. 8.3

[105] R.J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22,224–54, 1964. 1.2

[106] M. Spencer and C.J. Howe. How accurate were scribes? A mathematical model. *Literary and Linguistic Computing*, 17(3):311–322, 2002. 8.2

[107] M. Spencer, K. Wachtel, and C.J. Howe. The Greek Vorlage of the Syra Harclensis: A comparative study on method in exploring textual genealogy. *TC: A Journal of Biblical Textual Criticism*, 7, 2002. 8.2

[108] M. Steel. The complexity of reconstructiong trees form qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992. 4.5.1

[109] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13(7):964–969, 1996. 4.2

[110] D.L. Swofford. PAUP*: Phylogenetic analysis using parsimony (*and other methods). version 4., 2003. 8.2, 8.4

[111] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for associating patterns. In *Proc. ACM-SIGKDD Conf. Knowledge Discovery and Data Mining*, pages 491–502, 2002. 7.1.2

[112] E. Terra and C. Clarke. Frequency estimates for statistical word similarity measures. In *Proceedings of Human Language Technology conference / North American chapter of the Association for Computational Linguistics annual meeting*, pages 244–251, May 2003. Edmonton, Alberta. 7.1.2

[113] O. Trier, A. Jain, and T. Taxt. Feature extraction methods for character recognition - A survey. *Pattern Recognition*, 29:641–662, 1996. 6.8

[114] G. Tzanetakis and P. Cook. Music genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002. 6.1, 6.4

[115] J.-S. Varre, J.-P. Delahaye, and É. Rivals. The transformation distance: a dissimilarity measure based on movements of segments. In *Proceedings of German Conference on Bioinformatics*, Koel, Germany, 1998. 8.2

[116] P.M.B. Vitanyi. A discipline of evolutionary programming. *Theoret. Comp. Sci.*, 241(1-2):3–23, 2000. 4.6.2

[117] E. Wattel and M.P. van Mulken. Weighted formal support of a pedigree. In P. van Reenen and M.P. van Mulken, editors, *Studies in Stemmatology*, pages 135–169. Benjamins Publishing, Amsterdam, 1996. 8.2

[118] S. Wehner. Analyzing network traffic and worms using compression. http://arxiv.org/pdf/cs.CR/0504045 sec. 4.7.1, 6.1

[119] J. Weyer-Menkoff, C. Devauchelle, A. Grossmann, and S. Grünewald. Integer linear programming as a tool for constructing trees from quartet data. Preprint from the web submitted to Elsevier Science. 4.2, 4.5.1

[120] A. C.-C. Yang, C.-K. Peng, H.-W. Yien, and A.L. Goldberger. Information categorization approach to literary authorship disputes. *Physica A*, 329:473–483, 2003. 6.1

[121] P.N. Yianilos. Normalized forms for two common metrics. Technical Report 91-082-9027-1, NEC Research Institute, 1991. Revision 7/7/2002. http://www.pnylab.com/pny/ sec. 3.1.7

[122] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. 2.8, 8.2

Index

Chapter 11

Nederlands Samenvatting

Statistische Inferentie met Datacompressie door Rudi Cilibrasi

Dit proefschrift gaat over de theorie en praktijk van datacompressie-programma's die gebruikt worden om een bepaalde vorm van machinaal leren te realiseren. Het gaat hierbij in eerste instantie om het groeperen (clusteren) van objecten die op de een of andere manier op elkaar lijken. In eerste instantie gaat het om objecten die letterlijk in een computerbestand kunnen worden gerepresenteerd, zoals bijvoorbeeld literaire teksten, DNA sequenties, muziekstukken en afbeeldingen. Een van de belangrijkste conclusies van het proefschrift is dat het ook mogelijk is om te werken met objecten die staan voor abstracte begrippen, zoals "liefde" en "geluk."

De eenvoudigste en meest populaire toepassing die al in gebruik was voordat Cilibrasi aan zijn onderzoek begon, was taalboomconstructie. In een vroeg experiment werd de Universele Verklaring van de Rechten van de Mens gebruikt om verrassend accurate etymologische bomen te construeren. Dit gebeurde via een computerprogramma, zonder enige menselijke tussenkomst. Toch stemden deze bomen overeen met de beste inschattingen van taalkundige experts. Het computerprogramma was gebaseerd op twee onderdelen: ten eerste een elementair, en algemeen toepasbaar datacompressieprogramma, en ten tweede, een betrekkelijk eenvoudig boomzoeksysteem.

In zijn werk met voornamelijk Prof. Paul Vitányi heeft Cilibrasi getracht op beide onderdelen vooruitgang te boeken en nieuwe toepassingen te ontwikkelen. Dit werk begon met het schrijven van een eenvoudig conversieprogramma voor zogenaamde "pianola"-bestanden dat werd toegepast op MIDI-bestanden. Hiermee was Cilibrasi in staat om muziek automatisch naar genre of componist te classificeren; het bleek dat algemeen gebruikte compressieprogramma's zoals gzip of bzip2 tot op zekere hoogte in staat waren jazz, rock en klassiek van elkaar te onderscheiden. Ook was het in veel gevallen mogelijk om componisten te identificeren, soms zelfs met huiveringwekkende precisie. Dit experiment was zo succesvol, dat besloten werd tot het ontwikkelen van een breder toepasbare computerapplicatie. Cilibrasi ontwierp een "open bron"-programma genaamd CompLearn (beschikbaar via www.complearn.org). Dit maakte het mogelijk systematisch onderzoek te doen naar de vraag naar hoe universeel de *Genormaliseerde Compressie Afstand* (of NCD, "Normalized Compression Distance") nu eigenlijk was. De NCD

is een afstandsmaat tussen bestanden die aangeeft in hoeverre twee bestanden op elkaar lijken. Verschillende datacompressoren leiden tot verschillende versies van deze afstandsmaat.

In het eerste belangrijke artikel, Clustering by Compression (Groeperen met Compressie), ontwikkelden Vitányi en Cilibrasi een methode om aannames over datacompressoren te formaliseren, en beschreven zij enkele formele eigenschappen van zowel de theoretische "Kolmogorov Complexiteit compressor" als van praktische, daadwerkelijk toepasbare compressoren. Vervolgens pasten zij het CompLearn systeem toe op genetische analyse waarbij zij gebruik maken van complete genoomsequenties. Dit gebeurde wederom zonder enige biologische aannames of menselijke tussenkomst, hetgeen leidde tot de veronderstelling dat de methode enigszins objectiever was dan eerdere, vergelijkbare pogingen. Verder paste Cilibrasi CompLearn en NCD toe op een probleem in de radioastronomie, een domein waarvan hij bijna niets afwist. Gelukkig kwamen de resultaten overeen met die van in de nabijheid werkende astromische experts. Toepassing van het programma op vertaalde Russische literatuur toonde dat het boeken groepeerde aan de hand van een combinatie van vertaler en originele auteur. Misschien het meest verrassende resultaat werd bereikt bij het experiment waarbij afbeeldingen herkend moesten worden: het bleek mogelijk om het standaard compressieprogramma gzip te gebruiken voor het identificeren van afbeeldingen van handgeschreven cijfers uit een NIST gegevensbank. Cilibrasi werkte deze toepassing verder uit door het combineren van de NCD met zogenaamde Support Vector Machines. Dit zijn zelflerende systemen (algoritmen) die elke continue functie kunnen leren. De matrix van NCD afstanden tussen een groep "trainings"-bestanden en een aantal van te voren gekozen "anker"-bestanden werd hier gebruikt als input voor de support vector machine. Het resultaat is een algemeen classificatie- en regressieprogramma dat de kracht van discrete patroonvergelijking in datacompressoren combineert met de flexibiliteit van universeel lerende automaten van continue functies zoals Support Vector Machines of neurale netwerken.

De volgende grote vernieuwing kwam toen Cilibrasi zich realiseerde dat we dezelfde wiskundige formalismen die ten grondslag liggen aan datacompressie ook kunnen toepassen op andere objecten dan bestanden (of reeksen van bits). Het domein zou bijvoorbeeld ook uit zoektermen of tupels van zoektermen kunnen bestaan. Men kan dan een zoekmachine zoals Google gebruiken om het aantal pagina's op het world wide web te bepalen waarin deze (tupels van) zoektermen voorkomen. Het resultaat is dan de zogenaamde "genormaliseerde Google afstand" (NGD), die voor twee willekeurige termen (bijvoorbeeld woorden) aangeeft hoeveel ze, volgens het world wide web, op elkaar lijken. De NGD werd geimplementeerd als onderdeel van de CompLearn software met het zogenaamde Simple Object Access Protocol (SOAP) met de C en Ruby programmeertalen. Het grote verschil met NCD is dat het nu mogelijk is om afstand te bepalen op basis van de *namen* van objecten in plaats van hun, statistisch genomen, letterlijke inhoud. Dit bracht een scala aan nieuwe mogelijkheden teweeg. Cilibrasi vergeleek automatisch gegenereerde monsters van ontologische predicaten met die van WordNet, een project van de universiteit van Princeton. WordNet is een semantische concordantie van de Engelse taal die is gebouwd door menselijke experts. Er was ongeveer 85% overeenstemming tussen de eenvoudige, volledige automatisch lerende automaat en WordNet. Dit maakt automatische ontologie-uitbreiding een veelbelovende onderzoeksrichting. Hiermee zou als het ware *gratis* structuur aangebracht kunnen worden in het web; dit in tegenstelling tot een meer traditionele aanpak met RDF tripletten, het Sematische Web en XML uitwisseling. Al deze methoden

vereisen zeer tijdrovende menselijke arbeid, terwijl de nieuwe, op de NGD gebaseerde technieken, moeiteloos soortgelijke kennis vergaren uit het grote, ongebruikte reservoir van reeds bestaande webpagina's. Een ander interessant experiment was een automatisch Engels-Spaans vertaalsysteem dat werkte zonder menselijke hulp. Cilibrasi was in staat om de Spaanse vertalingen van vijf Engelse woorden door elkaar te husselen en de computer te laten uitvogelen welke Engelse en Spaanse woorden overeenstemden. Slechts door de correlatie tussen de Engelse NGD matrix en alle Spaanse permutaties te berekenen, lukte het de juiste volgorde te vinden.

In het meest recente artikel over Cilibrasi's onderzoek, bepaalden Cilibrasi en Vitányi de wiskundige details van het exacte type niet-deterministisch kwartetboomzoeksysteem dat hij uiteindelijk bouwde. Dit bleek een interessant project te zijn vanuit het oogpunt van de algoritmiek: het is een NP-hard probleem om de beste boom te bepalen volgens het gebruikte evaluatiecriterium, maar desalniettemin bleek het mogelijk voor vele interessante en bruikbare voorbeelden het antwoord te benaderen met een verrassende mate van snelheid en precisie. Daarbij maakte Cilibrasi gebruik van een cluster-berekenend systeem dat gebouwd was volgens de zogenaamde Message Passing Interface (MPI) standaard. Een voorbeeld dat Cilibrasi op dit moment zeer interesseert is een boom van 100 verschillende monsters van het vogelgriepvirus. Terwijl mutaties en recombinaties nieuwe virale variaties vormen, is het systeem in staat gebleken snelle en nuttige antwoorden te geven.

Cilibrasi moedigt geïnteresseerden aan de artikelen te bekijken die gepubliceerd zijn op zijn onderzoeks-homepage, of de online demo te proberen op de CompLearn website. Er kan ook een 3D demo gedownload worden met sleur-en-pleur-datamining. Windows en linux versies van alle programmatuur zijn tevens beschikbaar. Mede door een verbinding te maken met ongeveer 15 andere programmatuursystemen, heeft Cilibrasi een stuk gereedschap geproduceerd dat waardevol kan zijn voor onderzoekers over de hele wereld.

Chapter 12

Biography

Rudi Cilibrasi was born in Brooklyn, New York in 1974. He moved west to California at an early age and grew up in Sacramento mostly in front of a computer. He eventually went south to Pasadena where he attended the California Institute of Technology and received a degree in Egineering and Applied Sciences with a specialization in Computer Science. He contributed to an early version of the Linux kernel as an open-source programmer. After graduating he spent some years in industry and enjoyed developing programming skills in data compression algorithms, networking, cryptography, computer graphics, and artificial intelligence algorithms of all stripes. He returned to school after the dot-com crash and joined CWI in 2001 as a PhD student. After being awarded his doctoral degree he will focus more on his chosen interests: as an open source programmer he created CompLearn. CompLearn is an innovative machine learning system that leverages data compression programs for statistical analysis. He has also been active as a Hepatitis C advocate and activist, calling attention to this and a variety of other public health issues. He has been involved in community disaster response and emergency governance, helping to create both the TsunamiHelp.info wiki as well as the KatrinaHelp.info wiki after these unprecedented disasters. Rudi has a passion for striving for perfection in programming process and is a heavy proponent of the Agile Development camp; in recent years he has forgotten most of his C++ and Java knowledge to strengthen his focus on C and Ruby, which he believes hold promise for a new era of software reliability and power.

Papers by R. Cilibrasi

1. R. Cilibrasi, P.M.B. Vitanyi, A New Quartet Tree Heuristic for Hierarchical Clustering, IEEE/ACM Trans. Computat. Biol. Bioinf., Submitted; and was presented at the EU-PASCAL Statistics and Optimization of Clustering Workshop, 5-6 Juli 2005, London, UK. http://arxiv.org/abs/cs.DS/0606048

2. J. Tromp, R. Cilibrasi, The Limits of Rush-Hour Complexity, March 23, 2006. http://arxiv.org/pdf/cs.CC/0502068.

3. R. Cilibrasi, R. de Wolf, P. Vitanyi. Algorithmic clustering of music based on string compression, *Computer Music J.*, 28:4(2004), 49-67.

4. R. Cilibrasi, P. Vitanyi. Automatic Extraction of Meaning from the Web. IEEE International Symposium on Information Theory, Seattle, Washington, 2006.

5. R. Cilibrasi, P. Vitanyi. Clustering by compression, *IEEE Trans. Information Theory*, 51:4(2005), 1523 - 1545.

6. R. Cilibrasi, L. van Iersel, S. Kelk, J. Tromp. On the Complexity of Several Haplotyping Problems. Proc. WABI 2005: 128-139

7. R. Cilibrasi, L. van Iersel, S. Kelk, J. Tromp. On the complexity of the Single Individual SNP Haplotyping Problem, *Algorithmica*,To appear. http://arxiv.org/pdf/q-bio.GN/0508012.

8. T.Roos, T.Heikki, R.Cilibrasi, P.Myllymaki. Compression-based Stemmatology: A Study of the Legend of St. Henry of Finland, (number HIIT-2005-3), 2005.

9. R. Cilibrasi, P. Vitanyi. Similarity of Objects and the Meaning of Words, *Proc. 3rd Conf. Theory and Applications of Models of Computation (TAMC)*, 15-20 May, 2006, Beijing, China. Lecture Notes in Computer Science, Vol. 3959, Jin-Yi Cai, S. Barry Cooper, and Angsheng Li (Eds.), 2006, 21–45.

10. R. Cilibrasi Domain Independent Hierarchical Clustering *Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica*, 2004, nummer 8.

11. R. Cilibrasi, P. Vitanyi. The Google similarity distance, *IEEE/ACM Transactions on Knowledge and Data Engineering*, To appear.

12. R. Cilibrasi, Zvi Lotker, Alfredo Navarra, Stephane Perennes, Paul Vitanyi. *Lifespan of Peer to Peer Networks: Models and Insight*, 10th International Conference On Principles Of Distributed Systems (OPODIS 2006), December 12-15, 2006, Bordeaux Saint-Emilion, France.

Titles in the ILLC Dissertation Series:

ILLC DS-2001-01: **Maria Aloni**
Quantification under Conceptual Covers

ILLC DS-2001-02: **Alexander van den Bosch**
Rationality in Discovery - a study of Logic, Cognition, Computation and Neuropharmacology

ILLC DS-2001-03: **Erik de Haas**
Logics For OO Information Systems: a Semantic Study of Object Orientation from a Categorial Substructural Perspective

ILLC DS-2001-04: **Rosalie Iemhoff**
Provability Logic and Admissible Rules

ILLC DS-2001-05: **Eva Hoogland**
Definability and Interpolation: Model-theoretic investigations

ILLC DS-2001-06: **Ronald de Wolf**
Quantum Computing and Communication Complexity

ILLC DS-2001-07: **Katsumi Sasaki**
Logics and Provability

ILLC DS-2001-08: **Allard Tamminga**
Belief Dynamics. (Epistemo)logical Investigations

ILLC DS-2001-09: **Gwen Kerdiles**
Saying It with Pictures: a Logical Landscape of Conceptual Graphs

ILLC DS-2001-10: **Marc Pauly**
Logic for Social Software

ILLC DS-2002-01: **Nikos Massios**
Decision-Theoretic Robotic Surveillance

ILLC DS-2002-02: **Marco Aiello**
Spatial Reasoning: Theory and Practice

ILLC DS-2002-03: **Yuri Engelhardt**
The Language of Graphics

ILLC DS-2002-04: **Willem Klaas van Dam**
On Quantum Computation Theory

ILLC DS-2002-05: **Rosella Gennari**
Mapping Inferences: Constraint Propagation and Diamond Satisfaction

ILLC DS-2002-06: **Ivar Vermeulen**
A Logical Approach to Competition in Industries

ILLC DS-2003-01: **Barteld Kooi**
Knowledge, chance, and change

ILLC DS-2003-02: **Elisabeth Catherine Brouwer**
Imagining Metaphors: Cognitive Representation in Interpretation and Understanding

ILLC DS-2003-03: **Juan Heguiabehere**
Building Logic Toolboxes

ILLC DS-2003-04: **Christof Monz**
From Document Retrieval to Question Answering

ILLC DS-2004-01: **Hein Philipp Röhrig**
Quantum Query Complexity and Distributed Computing

ILLC DS-2004-02: **Sebastian Brand**
Rule-based Constraint Propagation: Theory and Applications

ILLC DS-2004-03: **Boudewijn de Bruin**
Explaining Games. On the Logic of Game Theoretic Explanations

ILLC DS-2005-01: **Balder David ten Cate**
Model theory for extended modal languages

ILLC DS-2005-02: **Willem-Jan van Hoeve**
Operations Research Techniques in Constraint Programming

ILLC DS-2005-03: **Rosja Mastop**
What can you do? Imperative mood in Semantic Theory

ILLC DS-2005-04: **Anna Pilatova**
A User's Guide to Proper names: Their Pragmatics and Semanics

ILLC DS-2005-05: **Sieuwert van Otterloo**
A Strategic Analysis of Multi-agent Protocols

ILLC DS-2006-01: **Troy Lee**
Kolmogorov complexity and formula size lower bounds

ILLC DS-2006-02: **Nick Bezhanishvili**
Lattices of intermediate and cylindric modal logics

ILLC DS-2006-03: **Clemens Kupke**
Finitary coalgebraic logics

ILLC DS-2006-04: **Robert Špalek**
Quantum Algorithms, Lower Bounds, and Time-Space Tradeoffs

ILLC DS-2006-05: **Aline Honingh**
The Origin and Well-Formedness of Tonal Pitch Structures

ILLC DS-2006-06: **Merlijn Sevenster**
Branches of imperfect information: logic, games, and computation

ILLC DS-2006-07: **Marie Nilsenova**
Rises and Falls. Studies in the Semantics and Pragmatics of Intonation