Roma Tre University
Ph.D. in Computer Science and Engineering

Pattern Recognition and Data Mining in Graphs and Strings

Guido Drovandi

Pattern Recognition and Data Mining in Graphs and Strings

A thesis presented by
Guido Drovandi
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

March 2011

Committee:
Dr. Paola Bertolazzi

Reviewers:
Prof. Alberto Apostolico
Prof. Jacek Błażewicz

I think I could, if I only knew how to begin.
For, you see, so many out-of-the-way things had
happened lately that Alice had begun to think
that very few things indeed were really impossible.

To my parents, who made it possible
To my sister, of whom I cannot be more proud

Acknowledgments

First of all, I would like to thank Paola Bertolazzi for her advices and support during these three years, and for the possibility to travel to many places.

A special thanks to Emanuel Weitschek for sustaining me during this period and to be a great roommate, and thanks to Stefania De Angelis, Cristina De Cola, and Giovanni Felici for the collaboration and friendship.

I would like to thank Alberto Apostolico for having me at the Georgia Institute of Technology and for sharing his knowledge and experience with me, and thanks to Luca Bonomi and Fabio Cunial for their suggestions during my period abroad.

I would like to thank Sebastiano Vigna for his valuable comments on the graph compression software.

I would like to thank Francesco Corman, Matteo De Felice, Maurizio Di Rocco, and Sandro Meloni for the interesting discussions and for have done this together.

I would like to thank my university mates Angela De Persiis, Giordana Freddi, Chiara Fumi, and Stefano Bizzoni. I really enjoyed the time spent with you. A particular thanks to the Gruppo Vacanze Piemonte.

I would like to thank Laura Bellamico, Matthieu Bourdon, and Tarek Salloum. You guys made the time in Atlanta so fantastic.

I am very grateful to Giuseppe Di Battista and to the mathematician Fabrizio Frati. They have shown me how amusing and fun research can be.

I would like to thank the members of the computer networks laboratory for their hospitality and the amazing atmosphere: Patrizio Angelini, Luca Cittadini, Bernardo Palazzi, Maurizio Patrignani, Maurizio Pizzonia, Tiziana Refice, Massimo Rimondini, and Stefano Vissicchio.

I would like to thank Marco Cipriani, Iuri Fanti, Davide Magistri, and Marco Passariello for having made the hours spent driving fly by.

I would like to thank Pamela Maggi, Roberto Belardo, and Luca Cascioli

with whom I share my way since the high school.

Finally, I would like to thank Dario Costa and all those I love. They give me the motivation and strength to do what I love.

Contents

Introduction

THE objective of this dissertation is to study methods to manage and to retrieve relevant information in graphs and strings, contributing to the advancement of research in this promising field.

Data analysis is one of the most important activity in different disciplines. Retrieving relevant information in huge amount of data is one of the most challenging problems in computer science. World Wide Web, biological experiments, human interactions, are just some examples of structures composed of thousands and thousands of elements. Clearly, it is impossible to have a visual or textual human readable representation of all these data; but it is difficult, even for an expert, to manually distinguish the important information from those that are not pertinent for her analysis. For these reasons it is important to adopt automatic techniques to retrieve only the most relevant information in large amount of data. This activity is known as *data mining*. Data mining deals with structured and unstructured data, that are, respectively, data for which we can give a model or not.

Here we focus our attention on two particular types of structured data: *graphs* and *strings*.

A graph (or a network) is a common data structure composed of a set of nodes linked together by some edges (or links). This structure is widely used since it is suitable to put in connections different entities. For example, a protein interaction network can be represented using a graph where proteins are nodes and interactions between two proteins are edges between nodes. In a social network, a person could be identified by a node, and a connection (business, friendship, etc.) is a simple edge between two individuals. These are just two examples of applications of graphs to different disciplines; however, graphs are one of the most used data structures in many other areas.

Due to the spreading of graphs in many applications, the problem of detecting important subgraphs is of primary importance since it is the base of the problems of graph mining listed below:

- Detection of exceptional subgraphs: a subgraph is considered to be exceptional in a graph when it is over or under represented with respect random graphs with the same dimension.
- Simulation studies: in order to study the evolution of a network, graphs can be created with simulation methods to predict how the graph will appear in the future.
- Realism of samples: to have a similar smaller graph of a given large graph, the small graph needs to contain all the important subgraphs of the large

graph.

- Graph compression: subgraphs represent regularities in the data, and these regularities can be exploited to compress the dimension and to store the graph without loss of information.

All these problems are difficult to model and solve, and they are studied in different research fields as in computer science, physics, mathematics, and others. Some common questions of these problems to which give an answer are "What subgraphs should we look for?" and "What do such subgraphs mean?".

The same problems and questions are also common in string mining. For example, in biological contexts it is important to highlight those substrings that are frequent in DNA or protein sequences, or the most relevant characteristics that can be used to generate realistic biological sequences. In many fields it is of central importance the problem of string compression, in which the most frequent substrings are exploited to obtain a compact representation.

The dissertation is structured in three parts.

Part I deals with the problem of counting subgraphs, also called motifs. In Chapter 1, some methods and algorithms proposed in the literature to search and to count motifs are illustrated. These algorithms have a time complexity that depends exponentially from the dimension of the motif; however, it was conjectured that, in some cases, the problem could be solved in polynomial time. We show that this is not true giving proofs of NP-completeness. In this part we also propose a method to obtain an approximation of the number of occurrences of topological colored motifs, that are motifs in which each node has a different color.

In Part II we consider the compression problem of graphs and strings, in particular those related with sequences of integers.

In Chapter 2 a new compression scheme for graphs is introduced. We illustrate the state of the art of graph compression and introduce a new compression scheme. To obtain a compact representation of a graph, some particular subgraphs are exploited; however, we do not search explicitly for these subgraphs. The scheme proposed in the chapter is suitable for the Web Graph, the graph of the URLs over the World Wide Web. To any extent, the proposed method allows to compress other kind of graphs in contrast with many previous works

that need the URLs for the compression phase.

Chapter 3 is dedicated to the compression of different types of networks: the social networks. These networks have less redundant subgraphs than the Web Graph, so they are less likely to be compressed. For these networks we introduce an extension of the method proposed in Chapter 2 that exploits the high number of reciprocal edges that characterize a social network.

Graph compression aims at reducing the information needed to represent links. After the compression, the graph is represented by a string of characters that must be stored on the disk (graph storage problem). A convenient way is to use prefix codes, that are variable length codes in which a code is not prefix of any other. Chapters from 4 to 6 deal with this problem.

Chapter 4 is based on the definition of a new prefix code that is suitable for the compression of a stream of integers following a power law distribution. This new code is compared with the best known codes proposed in the literature. We show analytically that this encoding follows the entropy of a zeta distribution (one of the most famous power law probability distribution) better than other codes. We focus on this kind of probability distribution since the string of characters that represents a compressed graph follows this particular distribution.

In Chapter 5 we propose a method to build prefix code for an arbitrary probability distribution of the integers. To deal with this problem, we introduce a general prefix code and we study its properties. Finally, using this general code we test the method creating prefix codes for different distributions and comparing the compression results with codes presented in the literature. Also in this case the results are as good as - and in same cases, or better than - those available in the literature.

In Chapter 6 we show that using the general code proposed in the previous chapter it is possible to give a common description to many of the prefix codes proposed in the literature. This result is possible thanks to the introduction of a new method to encode a finite set of integers.

Part III deals with the problem of find information in huge sets of strings. In Chapter 7, a logic mining method is applied to generate logic formulas to distinguish among different classes of strings. Each set of formulas character-

izing a class can be viewed as a pattern of the strings of the same class; in fact, when this pattern is found, the logic formulas are satisfied and we can assign the string to a class. The method is applied with success in the analysis of biological data, such as DNA sequences.

Part I

Network Motifs in Large Graphs

Chapter 1

Motif Counting Methods

How can we distinguish among graphs? What are the main characteristics of a graph? When two graphs are similar to each other? Many real word networks are built from different natural phenomena; the question is whether all of them share the same properties or not, and if it is possible to distinguish between real graphs and artificial graphs. Among the most studied characteristics of real graphs found in the literature, we can recall:

- power law distributions of some properties: for example the degree distribution follows a power law distribution that is there are few nodes of high degree;
- small diameters: the distance between two nodes is bounded above by a small constant; and
- community effects: two adjacent nodes are likely to share their neighbors.

These characteristics are sufficient to distinguish between some classes of graphs, and they are able to motivate some properties of the graph. However, these are global characteristics of the graph. We want to have also local properties. A different method to deal with the questions above is to study the peculiar subgraphs of a graphs. If we find good subgraphs we can characterize some graphs among the others, and possibly cluster those that share some common properties. Important subgraphs in a real graph are those that appear much more or less times in that graph than in a random graph of the same size. Subgraphs are also called motifs or patterns, and in the following we use the term motif that highlights the importance of the subgraph in the

graph. However, to find a motif is not a simple operation, since algorithms are computational expensive and real graphs are huge graphs of hundreds or thousands of nodes, hence a quadratic or cubic algorithm will require a long time to achieve its goal, or become impractical for some graphs.

As anticipated in the introduction, we deal with the problem of counting the number of motifs in a graph. This chapter focuses on some algorithmic techniques to find and to count motifs in graphs, and introduces a probabilistic method to estimate the number of occurrences of a motif.

1.1 Definitions and Notations

A *graph* $G = (V, E)$ is composed of a set of nodes V and a set of edges $E \subseteq V \times V$ that connect together two nodes. A *simple graph* is a graph in which each edge appears at most one time in E and there are no loops in the graph (edges from a node to itself). In a *directed graph* the direction of an edge (u, v) is important, it starts from a source u and arrives to a sink v. In a simple directed graph we can have the edges (u, v) and (v, u) since they have different directions. In an *undirected graph* the direction is not important so edges (u, v) and (v, u) are the same.

A *path* P is a sequence of distinct nodes such that there is an edge between two consecutive nodes (all nodes have degree two with the exception of the first and last node). A k-path or a path of length k is a path composed of k nodes and $k-1$ edges. A *cycle* C is a closed path in which there is and edge between the first and last nodes. A k-cycle or a cycle of length k is a cycle of k nodes and k edges.

The *degree* of a node u in an undirected graph $G = (V, E)$ is the number of edges having u as an endpoint, that is the cardinality of the set $\{(u, x) \in E, \forall x \in V\}$. The *degree distribution* of a graph represents the how many nodes of degree d there are in the graph; with $p(d)$ we indicate the probability that a node in the graph has degree d.

A subgraph $\overline{G} = (\overline{V}, \overline{E})$, where $\overline{V} \subseteq V$ and $\overline{E} \subseteq E$, is a connected graph that represent a small part of G, such that all the edges in $\overline{E}$ connect two nodes in $\overline{V}$. An *induced subgraph* $\overline{G}(\overline{V})$ of G is a motif such that all the edges of E between any two nodes in $\overline{V}$ belong to the subgraph ($(u, v) \in \overline{E}$ iff $u, v \in \overline{V}$ and $(u, v) \in E$).

A motif is considered to be important in a graph G if it occurs much more (or much less) than in a random network R. The most important model to generate a random graph is the Erdös Rényi model [ER60]. The algorithm to

build a random graph is simple. It starts with N nodes and for every pair of nodes it adds an edge with a probability p. The generated graph is indicated with $G(N, p)$ and its degree distribution is:

$$p_k = \binom{N}{k} p^k (1-p)^{N-k} \approx \frac{z^k e^{-z}}{k!}$$

where $z = p(N-1)$.

When we deal with colored graphs, we suppose to have a set $\{c_1, \ldots, c_k\}$ of different colors. A multiset of the set of colors is a set in which each color can appear more than once or it can not appear at all. With $f(c_i)$ we indicate the probability of a node to have color c_i.

1.2 Related Works

To establish if a subgraph appears more or less times in a given graph than in a random graph, one has to count how many times it appears in the given and in the random graph. The same algorithmic techniques can be used in both cases. For a given graph more efficient algorithms, that exploit their characteristic (planarity, treewidth, etc.), have been proposed. Instead for particular models of random graphs it is possible to define analytical formulas that give an estimated value of the number of occurrences of a motif.

In the proposed algorithms we can distinguish exact and approximate methods in colored or non-colored graphs.

Among the exact algorithms in non colored graphs, Alon *et al.* [AYZ97] describe an $O(|V|^\omega)$ algorithm for counting the number of cycles of size at most 7, where $\omega \leq 2.38$ is the exponent in fast matrix multiplication. Instead, the best algorithm, by Björklund *et al.* [BHKK08], for computing exactly the number of k-paths in an n nodes graph run in time $n^{k/2+O(1)}$. In [PCJ04] Przulj *et al.* described how to count all induced subgraphs with up to 5 nodes in a protein-protein interaction network. Faster techniques that count induced subgraphs of size up to 6 were developed by Hormozdiari *et al.* in [HBPS07], and for size up to 7 were shown by Grochow and Kellis [GK07]. Kashtan *et al.* [KIMA04] showed an algorithm for detecting induced network motifs that sample the network. This algorithm detects induced occurrences of small motifs (motifs with at most 7 nodes). Wernicke *et al.* [Wer06] claims that the algorithm by Kashtan *et al.* suffers from a sampling bias and scales poorly with increasing subgraph size. Thus, Wernicke presented an improved algorithm for network motif detection which overcomes these drawbacks. In [DS+07] Dost

et al. showed how to solve the subgraph detection problem for subgraphs of size $O(\log(n))$, provided that the subgraph is either a simple path, a tree, or a bounded treewidth subgraph. In [Epp95] Eppstein solved the subgraph isomorphism problem in planar graphs in linear time, for any pattern of constant size. He partitioned the planar graph into pieces of small tree-width, and applied dynamic programming within each piece. This method also leads to the solution of some problems on planar graph such as connectivity, diameter, induced subgraph isomorphism, and shortest paths.

The problem of approximating the numbers of occurrences of a motif is more tractable. Among the approximate methods, several algorithms for counting and detection of non-induced motifs use the color coding technique by Alon *et al.* [AYZ95]. Color coding is a combinatorial approach to detect simple paths, trees and bounded treewidth subgraphs in graphs. The method is based on the assignment of random colors to the nodes of an input graph, and only subgraphs, where each node has a unique color, are considered. Such colored subgraphs can then be detected through efficient use of dynamic programming, in time polynomial with the size of the input graph (however with a factor exponentially dependent by the dimension of the motif). If the above procedure is repeated sufficiently many times, it is guaranteed that a specific occurrence of the query subgraph will be detected with high probability. Alon and Gutner [AG10] combined the color coding technique with a construction of balanced families of perfect hash functions to obtain a deterministic algorithm to count the number of simple paths or cycles of size k in an input graph G. Alon *et al.* [ADH+08], improving the algorithm of Alon and Gutner, presented a polynomial time algorithm for approximating the number of non-induced occurrences of trees and bounded treewidth subgraphs with $k = O(\log(n))$ nodes with a running time of $2^{O(k \log(\log(k)))} \cdot n^{O(1)}$. Arvind and Raman [AR02] counted the number of subgraphs in a given graph G which are isomorphic to a bounded treewidth graph H. They gave a randomized approximate counting algorithm with a running time of $k^{O(k)} \cdot n^{b+O(1)}$, where n and k are the number of nodes in G and H, respectively, and b is the treewidth of H. The color coding technique works on colored graphs, but the original graph usually is not colored.

Concerning colored motif we can cite [LFS06] by Lacroix *et al.*. They defined a colored graph as a graph with nodes marked by one or more colors, and a colored motif as a multiset of the set of colors. Thus, a motif does not have any topological constraint, and an occurrence of a motif is a connected subset of the nodes of the graphs with the same colors of the motif. They presented an NP-completeness proof of the decision version of the search problem. The problem is NP-complete even if the graph is a tree. In [LFS06] they left as

open problem the complexity of the problem of searching a topological colored motif with fixed colors and no repetition allowed. They conjectured that this problem could be a polynomial tractable problem, however in the next section we give an NP-completeness proof. Authors also proposed an exact algorithm to find colored motifs in graphs. A simple version of the algorithm checks for all the possible $\binom{n}{m}$ candidates, where n is the number of nodes in the graph and m the number of colors in the motif, finding those that have the same colors of the motif and that are connected subgraphs. The solution space of this method is huge. In order to reduce the number of possible candidates, authors introduced a breadth first search. This search, mixed with a backtracking strategy, is performed starting from each node in the graph. At each step of the search, a subset of the nodes visited by the search is marked as part of a candidate. If the subset has at least m nodes then the colors are checked, otherwise the set of marked nodes is incremented performing a new step of the search.

Since the running time of all these algorithms increase exponentially with the size of the motif a non-algorithmic method has been proposed, for random graphs, by Schbath *et al.* [SLS09]. This article introduced a new approach for assessing the exceptionality of colored motifs as defined by Lacroix *et al.* [LFS06]. They established analytical formulas for the mean and the variance of the count of a colored motif in an Erdös-Réenyi random graph. The mean number of occurrences of a motif $m = \{m_1, \ldots, m_k\}$ (m_i are colors) in an Erdös-Réenyi random graph of n nodes and probability p is given by:

$$E(m) = \binom{n}{k}\mu(m)$$

where $\mu(m)$ is the occurrence probability of the motif that is given by the probability that a subset s of k nodes is connected ($g(k,p)$) and that s has the same colors of m ($\gamma(m)$):

$$\mu(m) = g(k,p) \cdot \gamma(m)$$

The probability that a set of nodes s has the same colors of the motif is:

$$\gamma(m) = \frac{k!}{\prod_{c\in\mathcal{C}} s(c)!} \prod_{i=1}^{k} f(m_i)$$

where $f(m_i)$ is the probability of a node to have color m_i, and $s(c)$ is the multiplicity of color c in the motif m. And the probability to have a connected

subset of k nodes is recursively defines as:

$$g(k,p) = 1 - \sum_{i=1}^{k-1} \binom{k-1}{i-1} g(i,p)(1-p)^{i(k-i)}$$

the base case is $g(1,p) = 1$.

1.3 Counting Colored Motifs

The conjecture by Lacroix *et al.* [LFS06] says that, considering a topological colored motif the problem of finding an occurrence of the motif in a colored graph is always NP-complete "except when the colors are fixed (each node has a color assigned to it) and no repetition of colors is allowed". Here we prove that the problem is hard even under these restrictions and give a simple extension of the method of Schbath *et al.* [SLS09] for paths, cycles and cycles with chords.

NP-completeness Results

First we give the following two theorems for the NP-completeness of the problem in case of non-induced and induced subgraphs.

Theorem 1.1 *Given a colored topological motif M without repeated colors and a colored undirected graph G, to find an occurrence of M in G is an NP-complete problem.*

Proof: The completeness for NP is shown by a reduction from Satisfiability. Let an instance of SAT be given by a collection of variable $X = \{x_1, \ldots, x_n\}$ and a collection of clauses $C = \{c_1, \ldots, c_m\}$ over X.

Form a motif graph M and a target graph G as follows. Let the graph M be a complete graph K_m and set for each node u_i in M $color(u_i) = c_i$. Let $G = (V_G, E_G)$ be the graph whose nodes consist of pairs:

$$V_G = \{(x,c) | c \in C, x \in c\}$$

and whose edges are:

$$E_G = \{((x,c_i),(y,c_j)) | i \neq j, y \neq \overline{x}\}$$

Graph G has one node corresponding to each occurrence of a literal in a clause of C. Let $color((x,c)) = c$ for all nodes $(x,c) \in V_G$ (see Figure 1.1). We

claim that there is a satisfying truth assignment for C if and only if M is a subgraph of G.

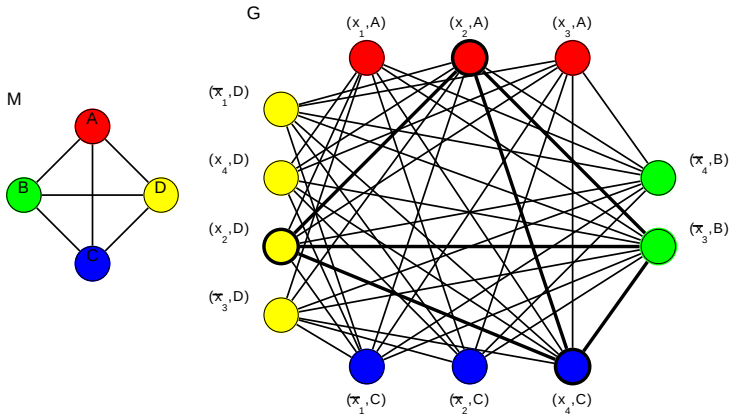


Figure 1.1: Motif M and graph G for clauses $A = \{x_1, x_2, x_3\}$, $B = \{\overline{x}_4, \overline{x}_3\}$, $C = \{\overline{x}_1, \overline{x}_2, x_4\}$ and $D = \{\overline{x}_1, x_4, x_2, \overline{x}_3\}$. The bold nodes in G are the selected nodes for the truth assignment $t = \{x_2, \overline{x}_3, x_4\}$

($\Rightarrow$) Assume that t is a truth assignment for C. Define a mapping h from the nodes of M onto the nodes of G as follows: For all nodes u of M set $h(u) = (x, color(u)) \in V_G$, such that x is *true* under assignment t; such a node can be selected because t satisfied at least one literal of every clause $c \in C$. Mapping h induces a complete graph K_m on G because only a node (x, c_i) is selected for $1 \leq i \leq m$ and each selected node (x, c_i) is connected to the others since no nodes $(\overline{x}, c_j)$ $(j \neq i)$ could be selected ($\overline{x}$ doesn't belong to truth assignment t).

($\Leftarrow$) Assume that h is a mapping of M in G. Set $t(x) = true$ if (x, c_i) $(1 \leq i \leq m)$ is selected otherwise if $(\overline{x}, c_i)$ is selected $(1 \leq i \leq m)$ set $t(x) = false$. It is easy to see that t is a well-defined truth assignment for a subset of variables in C and that t satisfies each clause in C since at least one node (x, c) for each $c \in C$ must be selected.

□

When we deal with the problem of looking for induced occurrence of the motif the problem is NP-complete even if the motif is a simple path. For this

problem we give the following theorem.

Theorem 1.2 *Given a colored topological motif M without repeated colors and a colored undirected graph G, to find an induced occurrence of M in G is an NP-complete problem, even if M is a path.*

Proof: The completeness for NP is shown by a reduction from Satisfiability. Let an instance of SAT be given by a collection of variable $X = \{x_1, \ldots, x_n\}$ and a collection of clauses $C = \{c_1, \ldots, c_m\}$ over X.

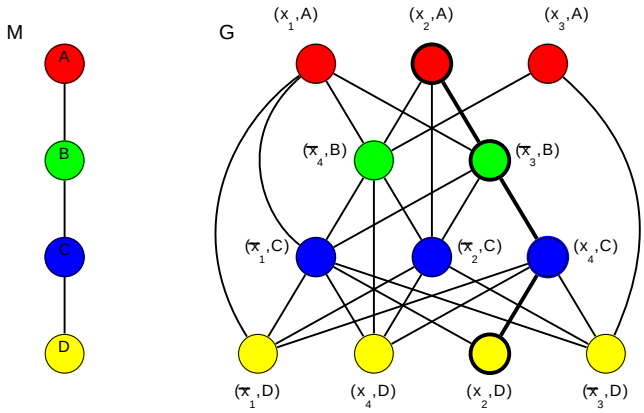


Figure 1.2: Motif M and graph G for clauses $A = \{x_1, x_2, x_3\}$, $B = \{\overline{x}_4, \overline{x}_3\}$, $C = \{\overline{x}_1, \overline{x}_2, x_4\}$ and $D = \{\overline{x}_1, x_4, x_2, \overline{x}_3\}$. The bold nodes in G are the selected nodes for a truth assignment $t = \{x_2, \overline{x}_3, x_4\}$

Form a motif path M and a target graph G as follows. Let the path M be composed by nodes $\{u_1, \ldots, u_m\}$, with $color(u_i) = c_i$, and edges $\{(u_i, u_{i+1}) | 0 \leq i < m\}$. Let $G = (V_G, E_G)$ be the graph whose nodes consist of pairs:

$$V_G = \{(x, c) | c \in C, x \in c\}$$

and whose edges are:

$$\begin{aligned} E_G \quad = \quad & \{((x, c_i), (y, c_j)) | 1 \leq i < m, j = i + 1, y \neq \overline{x}\} \cup \\ & \{((x, c_i), (\overline{x}, c_j)) | 1 \leq i < m, j > i + 1\} \end{aligned}$$

Graph G has one node corresponding to each occurrence of a literal in a clause of C. Let $color((x,c)) = c$ for all nodes $(x,c) \in V_G$ (see Fig. 1.2). We claim that there is a satisfying truth assignment for C if and only if M is an induced subgraph of G.

($\Rightarrow$) Assume that t is a truth assignment for C. Define a mapping h from the nodes of M onto the nodes of G as follows: for all nodes u of M set $h(u) = (x, color(u)) \in V_G$, such that x is $true$ under assignment t; such a node can be selected because t satisfies at least one literal of every clause $c \in C$. Mapping h induce a path on G because all selected nodes (x, c_i) are connected only with (y, c_{i-1}) (if $i > 1$) and (z, c_{i+1}) (if $i < m$) since no nodes $(\overline{x}, c_j)$ $(j \neq i \pm 1)$ could be selected ($\overline{x}$ doesn't belong to truth assignment t).

($\Leftarrow$) Assume that h is a mapping of M in G. Set $t(x) = true$ if (x, c_i) $(1 \leq i \leq m)$ is selected otherwise if $(\overline{x}, c_i)$ is selected $(1 \leq i \leq m)$ set $t(x) = false$. It is easy to see that t is a well-defined truth assignment for a subset of variable in C and that t satisfies each clause in C since at least one node (x, c) for each $c \in C$ must be selected. □

Counting Method

Since we have proved that the problem of finding a topological colored motif is NP-complete even when the repetition of the colors is not allowed, we propose an extension of the idea by Schbath *et al.* [SLS09] for colored topological motifs. Schbath *et al.* deal with colored motifs, where a motif is a multiset of the set of colors that is a set in which each color can appear more than once. A motif has no topological constraints, so an occurrence of a motif is just a connected subgraph with the same colors of the motif.

Instead we define a colored topological motif as a connected subgraph where each color appears exactly once. This problem is a hard problem, so obtaining an estimate of the number of motifs can avoid to wait the running time of exponential time algorithms.

Given a graph $G = (V, E)$ we suppose to know its degree distribution, and from this information we try to give an approximated count of the occurrences of a motif. The motifs under analysis are simple motifs of at most 6 nodes, and they are: a path, a cycle, and a cycle with a chord. The dimension and the topology of these motifs are commonly used in many works (for example [SLS09, HBPS07]).

We suppose to have a colored graph with k colors and a motif of k' nodes $(k' \leq k)$, and we suppose that the distribution of the colors is uniform over the

graph. This last assumption is still used in the color coding [AYZ95] technique for which a non-colored graph is colored assigning each node to a color with the same probability.

Path

Say x_i the event that a node has degree i, and $p(x_i)$ the probability to have such a node. The first node u_1 of the path of color c_1 can be chosen among n/k nodes. Instead, the second node must be a node of color c_2 in the neighborhood of the first node. The probability that a node u_2 is a neighbor of u_1 is:

$$P(u_2) = \left(\frac{1}{n}p(x_1) + \ldots + \frac{n-1}{n}p(x_{n-1})\right)$$

that is the summation of the probabilities of u_1 to be of degree i times the probability of choosing one of the i neighbors of u_1 among the n nodes. The neighbors of u_1 of color c_2 are $P(u_2)n/k$. The probability that a node $u_3 \neq u_1$ is a neighbor of u_2 is:

$$P(u_3) = \left(\frac{2}{n}p(x_2) + \ldots + \frac{n-1}{n}p(x_{n-1})\right)$$

for this probability we do not count the case in which u_2 has only one neighbor since it is the node u_1 and we want a node that is different from u_1. By the same consideration the probability to have a neighbor of u_3 is $p(u_4) = p(u_3)$ and so on ($p(u_{i+1}) = p(u_i)$ for $i \leq k'$).

The total number of paths of length k' starting from node u_1 is:

$$\begin{aligned} Path(u_1) &= \frac{n}{k}p(u_2) \cdot \frac{n}{k}p(u_3) \cdot \ldots \cdot \frac{n}{k}p(u_{k'}) \\ &= \left(\frac{n}{k}\right)^{k'-1}\left(p(u_2)p(u_3)^{k'-2}\right) \end{aligned}$$

We can obtain the total number of paths in the graph multiplying the previous equation for the number of nodes (n/k) of color c_1.

Considering d regular graphs ($p(x_d) = 1$), the total number of paths is:

$$Path(G) = \frac{n}{k}Path(u_1) = \frac{n}{k}\left(\frac{n}{k}\frac{d}{n}\right)^{k'-1} = \frac{nd^{k'-1}}{k^{k'}}$$

In the following, we use k regular graphs as benchmark for the formulas.

Cycle

A colored cycle can be viewed as a colored path in which the last node is connected to the first. The probability that two fixed nodes are connected by an edge, in an undirected graph, is:

$$p(u_1, u_{k'}) = \frac{2e}{n(n-1)} = \frac{\sum_{d=1}^{n-1} n \cdot d \cdot p(x_d)}{n(n-1)} \approx \frac{\sum_{d=1}^{n-1} d \cdot p(x_d)}{n}$$

where n is the number of nodes and e the number of edges.

It follows that the number of cycles in a graph is:

$$\begin{aligned} Cycle(G) &= \left(\frac{n}{k}\right)^{k'} \left(p(u_2)p(u_3)^{k'-2}\right) \frac{\sum_{d=1}^{n-1} d \cdot p(x_d)}{n} \\ &= \frac{n^{k'-1}}{k^{k'}} \left(p(u_2)p(u_3)^{k'-2}\right) \sum_{d=1}^{n-1} d \cdot p(x_d) \end{aligned}$$

In the case of d regular graphs, the number of cycles is:

$$Cycle(G) = \frac{n}{k} \left(\frac{n}{k}\frac{d}{n}\right)^{k'-1} p(u_1, u_{k'}) = \frac{nd^{k'-1}}{k^{k'}} \frac{d-1}{n-2} = \frac{d^{k'-1}(d-1)}{k^{k'}} \frac{n}{n-2}$$

The probability to have the edge in this case is $\frac{d-1}{n-2}$ since we know a neighbor of $u_{k'}$ so we can choose among the $d-1$ neighbors of u_{k-1} in a set of $n-2$ nodes (the whole set minus $u_{k'}$ and $u_{k'-1}$).

Cycle with α Chords

The last motif of our interest is a cycle with α chords. As done for the cycle, we multiply the probability to have a cycle by the probability to have α pairs of nodes of the cycle connected to each other. The probability to have these α chords is:

$$p_\alpha = \left(\frac{\sum_{d=1}^{n-1} d \cdot p(x_d)}{n}\right)^\alpha$$

The number of cycle with α chords is:

$$\begin{aligned} Chord(G) &= \frac{n^{k'-1}}{k^{k'}} \left(p(u_2) p(u_3)^{k'-2} \right) \left(\sum_{d=1}^{n-1} d \cdot p(x_d) \right) p_\alpha \\ &= \frac{n^{k'-1-\alpha}}{k^{k'}} \left(p(u_2) p(u_3)^{k'-2} \right) \left(\sum_{d=1}^{n-1} d \cdot p(x_d) \right)^{\alpha+1} \end{aligned}$$

In the case of d regular graphs, the number of cycle with α chords is:

$$\begin{aligned} Chord(G) &= \frac{d^{k'-1}(d-1)}{k^{k'}} \frac{n}{n-2} p_\alpha \\ &= \frac{d^{k'-1}(d-1)}{k^{k'}} \frac{n}{n-2} \left(\frac{d-2}{n-3} \right)^\alpha \end{aligned}$$

where $p_\alpha = \left(\frac{d-2}{n-3} \right)^\alpha$ since we know 2 neighbors of each node in the cycle so we can choose only among the other $d-2$ neighbors.

The formulas proposed so on are derived from a simulated traversal of the graph, in which at each node we can move on its neighbors according to their color. For this traversal we suppose that the colors are equally distributed over all the neighbors of a node, that is that the number of neighbors of a node of color c_1 is the same that the neighbors of color c_2. However, this condition is not always verified. To introduce the possibility that the colors are not equally distributed, we simulate a traverse of the graph with a probability to visit an edge. Introducing this probability, the traversal to count the occurrences can be viewed as performed by a random walker.

We consider a constant probability p_ω for all the edges and the formulas change as follows:

$$\begin{aligned} Path(G) &= \frac{n d^{k'-1}}{k^{k'}} p_\omega^{k'} \\ Cycle(G) &= \frac{d^{k'-1}(d-1)}{k^{k'}} \frac{n}{n-2} p_\omega^{k'} \\ Chord(G) &= \frac{d^{k'-1}(d-1)}{k^{k'}} \frac{n}{n-2} \left(\frac{d-2}{n-3} \right)^\alpha p_\omega^{k'} \end{aligned}$$

The factor $p_\omega^{k'}$ represent the probability to choose the edges over the traversal to find a path.

Experimental Results

The experiments are conducted on the formulas for the d regular graphs. For each experiment we ran 10 thousand tests and the values presented are the average over the 10 thousand runs. Experiments ran for random graphs of 500 and 1,000 nodes and degrees 10 and 20. The number of colors is fixed to 6. We choose these parameters according to [LFS06]. A d regular random graph G is generated as follows:

1. initialize two graphs G and G', G as an empty graph of N nodes, and G' as a complete graph of N nodes;
2. randomly select an edge (u, v) of G' and move the edge to G if both u and v have a degree less than k, otherwise simply remove (u, v) from G'; and
3. if G' is an empty graph then stop, otherwise start again from point 2.

In Table 1.1 and Table 1.2 are shown the results from random 10-regular graphs and 20-regular graphs respectively.

The result obtained using the probability p_ω to visit an edge (columns F_ω in tables) are quite close to the expected occurrences of a motif, while the results without p_ω (columns F_T), as expected, are far from the expected values. The values of ω are fixed to 0.94 and 0.96 respectively for 10-regular and 20-regular graphs.

1.4 Conclusions and Open Problems

A motif is a small component of a graph that is able to describe some properties and distinguish the graph among others of different type. In this chapter we have recalled some methods used to search and to count motifs in graphs. In particular the counting problem allows to determine what motifs are more important; in fact, not all the motifs are equals. Motifs that are over or under expressed, respect to a random network, are considered to be more important. Since all the algorithms proposed have an exponential computational time, approximate techniques to counting the number of occurrence of a motif achieve a relative importance.

We have shown that the problem of finding a motif in a network is NP-complete even if the motif has fixed and non repeated colors, and also, when we consider induced occurrences, when the motif is a simple path. We have

			Degree 10			
Motif	N	k	$\mathbb{E}$	σ	F_T	F_ω
Path	500	4	312.05	60.75	385.80	288.60
		5	465.99	99.14	643.00	447.33
		6	694.18	154.50	1071.67	693.36
	1000	4	624.66	86.92	771.60	602.43
		5	935.63	139.61	1286.01	943.81
		6	1398.18	219.55	2143.35	1478.63
Cycle	500	4	5.10	2.48	6.97	5.21
		5	7.58	3.31	11.62	8.08
		6	11.23	4.46	19.37	12.53
	1000	4	5.07	2.36	6.96	5.43
		5	7.65	3.07	11.60	8.51
		6	11.44	4.00	19.33	13.34
Cycle with Chord	500	4	0.07	0.27	0.11	0.08
		5	0.11	0.33	0.19	0.13
		6	0.15	0.41	0.31	0.20
	1000	4	0.03	0.19	0.06	0.05
		5	0.05	0.23	0.09	0.07
		6	0.08	0.29	0.16	0.11

Table 1.1: The average number of motifs found in 10 thousands random 10-regular networks compared to the results obtained using the formula. $\mathbb{E}$ and σ are respectively the expected and the variance of the number of motif in the random networks. F_T are the values of the formulas without the visit probability p_w, F_ω are the values considering p_w.

also presented a simple method to count the occurrences of topological colored motifs in networks given their degree distribution; we have shown that considering random regular graphs, this approach leads to a good approximation of the expected value.

The formula of the expected number of motifs simulates a walk on the graph performed by a random walker that counts the occurrences of the motif. At this moment, the probability to move on an edge is manually chosen.

Open Problem 1.1 *Can we set a priori the probability of the random walker?*

In [SLS09] a formula for computing the variance of the number of occurrences of a motif without any topological constraint, is given. Here we have dealt with topological motifs in which each node is assigned to a distinct color.

			Degree 20			
Motif	N	k	$\mathbb{E}$	σ	F_T	F_ω
Path	500	4	2780.43	438.27	3086.42	2621.44
		5	8768.01	1319.90	10288.07	8388.61
		6	27584.80	3548.11	34293.55	26843.54
	1000	4	5562.19	610.10	6172.84	5242.88
		5	17573.40	1858.40	20576.13	16777.21
		6	55576.57	4974.12	68587.11	53687.09
Cycle	500	4	101.20	19.37	117.75	100.01
		5	318.37	54.71	392.52	320.05
		6	1002.19	149.36	1308.39	1024.15
	1000	4	100.82	15.48	117.52	99.82
		5	318.38	40.45	391.73	319.41
		6	1006.72	109.00	1305.77	1022.10
Cycle with Chord	500	4	3.38	2.06	4.26	3.62
		5	10.62	4.45	14.22	11.59
		6	33.02	10.86	47.39	37.09
	1000	4	1.71	1.40	2.12	1.80
		5	5.21	2.66	7.07	5.76
		6	16.70	5.97	23.57	18.45

Table 1.2: The average number of motifs found in 10 thousand random 20-regular networks compared to the results obtained using the formula. $\mathbb{E}$ and σ are respectively the expected and the variance of the number of motifs in the random networks. F_T are the values of the formulas without the visit probability p_w, F_ω are the values considering p_w.

However, this problem is still open.

Open Problem 1.2 *Is it possible to give a formula for the variance of the number of occurrences of colored topological motifs using a random walk?*

Part II

Graph Compression and Storage

Introduction

THANKS to new technologies and increasing storage capability, more and more real graphs are assuming huge dimension in the size of the nodes. For example, the possibility to be connected to each other using social networks produces graphs that are interesting to be analyzed, but whose dimensions make this operation difficult.

This large-scale graphs have too many links to be stored in the main memory which forces several random seeks to a disk. Since disk access is (by five orders of magnitude) slower than main memory access, this leads to unacceptable retrieval times. To mitigate this problem, several compression techniques have been proposed for large graphs, aimed at reducing the number of bits per link required in graph representation. In this part of the thesis we show how redundancies in graphs and strings can be exploited in graph compression and storage.

The problem of graph compression has been approached by several authors over the years, beginning with the paper [Tur84] by Turan. Among the most recent works, Feder and Motwani [FM95] looked at graph compression from the algorithmic point of view, with the goal of carrying out algorithms on compressed versions of a graph. All these works are focused on the exploitation of redundancies in the data: they substitute frequent patterns with indices or compress sequences of the same element with a smaller representation. After one have exploited all the possible redundancies it emerges the problem to store the compressed data on the disk. This task is realized through the compression of a string of characters.

The aim of a compact representation of a stream of characters is to have an expected length of the codeword, assigned to each character, that follows the entropy of the stream. The entropy is the most important measure to know how a stream of characters is redundant and then how compact can be its representation; it is a lower bound on the average number of bits needed to

encode the characters [Sha48].

In this part we focus on the problem of storing in an efficient way huge graphs, with the possibility to access to the information contained without decompressing the whole graph, providing random access to the structure. We also focus on the problem of storing in an convenient way the compressed graph. For this task, the last three chapters of this part deal with the problem of string compression and in particular with the compression of sequences of integers, investigating different aspects of prefix codes.

In the first chapter of this part a new compression technique is introduced; this method is suitable to compress the Web Graph. Compression scores and retrieval time performances are shown, compared to the most important compression schemes presented so far. Moreover, algorithms that need to read in a sequential way the whole graph can take advantage of the compressed structure and achieve better performances. The compression is based on the exploiting of particular subgraphs; however, in contrast with other method proposed in literature, we do not search explicitly for these subgraphs but we order the nodes to induce their exploitation.

The second chapter is focused on social networks. This kind of networks are more difficult to be compressed; however, it is possible to obtain good results and build data structures that allow complex queries. The method proposed for the social networks is an extension of the method for the Web Graph.

In the third chapter we propose a method for efficiently storing a sequences of integers representing a compressed graph. Many works have introduced different techniques, for the problem of encoding a stream of characters. One of the most important is due to Huffman [Huf52] that computes the probability of the characters in a string and builds a tree to assign to each character a codeword. Different techniques of encoding are based on the knowledge of the probability distribution of the characters and achieve comparable results to the Huffman method, without the necessity of secondary structures. Here we use this last strategy and in particular the prefix codes, in which each code is not a prefix of any other code. Prefix codes use few bits to encode the most probable characters and it was proved [Eli75] that it exists a prefix code that achieves always an expected length at most twice the entropy of the stream.introduce a prefix code for power law distributions of the integers. We prove that this new encoding technique is most suitable than the others presented in the literature and we show that these codes follow the entropy of the zeta distribution.

In the fourth chapter we focus on a method to obtain a prefix code that performs well under a probability distributions that is unknown. In fact, we suppose to know only some properties of the distribution that are easily com-

putable at run time while reading the stream. This technique leads to the definition of codes that achieve better results than the best known, or the technique suggests the same code that, in the literature, is shown to be the best.

In the last chapter of this part, we try to give a common description to some prefix codes presented by different authors. In the literature, each author gave the definition of his code in a different way and this renders the comparison of two encoding technique more difficult. To give an unified description of the prefix codes, we introduce an extension of the truncated binary encoding; in this way, a code is always described by a fixed number of parameters (five).

leads to increase the difficult of comparison of two encoding technique and, moreover, to have the same code defined more than once. To give an unified description of the prefix codes, we introduce an extension of the truncated binary encoding, and in this way a code is described by five parameters.

Chapter 2

A New Method to Compress the Web Graph

In recent years, many applications have been developed for retrieving information over the *World Wide Web*, and analyzing the structure of the underlying *Web Graph*, which contains currently more than 1 trillion different URLs[1]. However, its dimensions make difficult to design algorithms to study topological or dynamic properties of the Web Graph. A simple solution is to deal only with small snapshots of the whole graph; nevertheless, the possibility to manage the Web Graph as a whole would allow to go into its properties, and to develop more accurate models of its growth.

This chapter[2] presents a new compression scheme for the Web Graph. It is possible to obtain a compact representation of the Web Graph exploiting the redundancies of its peculiar topology. The compression method proposed here can be properly fitted to allow fast access to the adjacency lists of the nodes or to reduce the space needed for the representations. Exploiting the compact dimension of the graph it is also possible to speed up some algorithms and to introduce queries that do not need the decoding of the compressed graph.

[1] http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

[2] The contents of this chapter are a joint work with Alberto Apostolico appeared in [AD09].

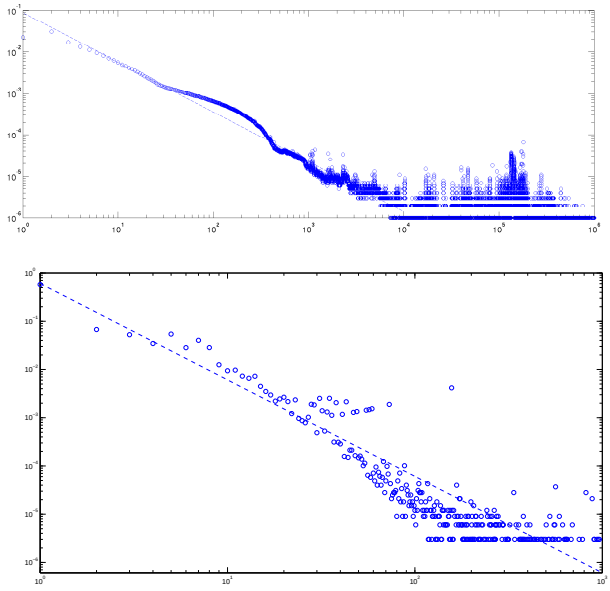


Figure 2.1: The distribution of gaps between neighboring nodes (top) and of node degrees (bottom) in the dataset "`in-2004`" as gathered by Boldi and Vigna [BV04] using UbiCrawler [BCSV04].

2.1 Background

The Web Graph over some subset of URLs or *pages* is a directed graph in which each node u is an URL in the subset and an edge or link is directed from u to v whenever there is a hyperlink from u to v. Formally, the Web Graph is a directed graph $G = (V, E)$, where V is the set of URL *identifiers* or *indices* and E is the set of links between them. For any node $v \in V$, $A_v = \{u_1, \ldots, u_k | u_i \in V\}$ will denote the adjacency list of v. We will assume that the identifiers in each list appear sorted in some linear order.

From the standpoint of compression, one convenient way to assign indices is to sort the URLs lexicographically and then to give each URL its corresponding rank in the ordering. This induces two properties, namely:

- *Locality*: For a node with index i, most of its neighbors may be expected

to have an index close to i; and

- *Similarity*: URLs with a lexicographically close index (for example, pages residing on a same host) may be expected to have many neighbors in common (that is, they are likely to reference each other).

These properties induce that the gap between the index i of a page and the index j of one of its neighbors is typically small. The approach followed here is based on ordering nodes based on the Breadth First Search (BFS) of the graph instead of the lexicographic order, while still retaining these features. In fact, Figure 2.1 (top) shows the corresponding distribution of the gaps between neighbors. This distribution follows a power law similar to the node degree distribution displayed in Figure 2.1 (bottom).

We focus on the efficient storage of and rapid access to compressed graphs. In contrast to other techniques that make use of lexicographic ordering of URLs, and thus are specifically tailored for the Web Graph, however, the scheme presented here does not need to use the URLs and therefore may be applied to graphs of more general nature. The specific aim of this new method is to produce a compressed graph supporting queries of the kind:

- *For two input pages X and Y, does X have a hyperlink to page Y?*
- *For input page X, list the neighbors of X*

In summary, the new method produces a compressed Web Graph featuring high compression ratio, short retrieval time of the adjacency list of a node, and fast testing of whether or not two pages share a hyperlink.

2.2 Related Works

Among the earliest works specifically devoted to web graph compression one finds papers by Adler and Mitzenmacher [AM01], and Suel and Yuan [SY01].

For quite some time the best compression performance was that achieved by the *WebGraph* algorithm by Boldi and Vigna (BV in the following) [BV04], which uses two parameters to compress the Web Graph $G = (V, E)$: the *refer range* W and the *maximum reference count* R. For each node $v_i \in V$, BV represent a modified version of A_{v_i} obtained from an adjacency list A_{v_j} of another node $v_j \in V$ $(i - W \leq j < i)$ called the *reference list* (the value $i - j$ is called *reference number*). The representation is composed of a sequence of bits (*copy list*), that tells if a neighbor of v_j is also a neighbor of v_i, and the

description of the *extra nodes* $A_{v_i} \setminus A_{v_j}$. Extra nodes are encoded using gaps that is, if $A_{v_i} \setminus A_{v_j} = \{a_1, \ldots, a_l\}$ the representation is $\{a_1, a_2 - a_1 - 1, \ldots, a_i - a_{i-1} - 1, \ldots, a_l - a_{l-1} - 1\}$. Table 2.1 reproduces an example from [BV04] of representation using copy list.

Node	Degree	Ref. N.	Copy List	Extra Nodes
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203
16	10	1	011100110	22, 316, 317, 3041
...	...	...	...	...

Table 2.1: The Boldi and Vigna representation using copy lists [BV04].

The parameter R is the maximum size of a reference chain; In fact BV do not consider all the nodes v_j, with $i - W \leq j < i$, to encode v_i, but only those that produce a chain not longer than R. Boldi and Vigna also developed ζ-Codes [BV05], a family of universal codes used to compress power law distributions with small exponent.

Claude and Navarro (CN) [CN07] proposed a modified version of Re-Pair [LM99] to compress the Web Graph. Re-Pair is an algorithm that builds a generative grammar for a string by hierarchically grouping frequent pairs into variables, frequent variable pairs into more variables and so on. Along these lines, CN essentially applies Re-Pair to the string that results from concatenation of the adjacency lists of the nodes. The data plots presented in [CN07] display that their method achieved a compression comparable to BV (at more than 4 bits per link) but the retrieval time of the neighbors of a node is faster.

Buehrer and Chellapilla [BC08] (BC) proposed a compression based on a method presented in [FM95] by Feder and Motwani; They search for dense bipartite graphs (communities) and for each occurrence found they generate a new node, called *virtual node*, that replaces the intra-links of the community (see Figure 2.2). In [KCA09] Karande *et al.* showed that this method has competitive performances over well know algorithms including PageRank [BP98, PBMW99].

In [AMN08], Asano *et al.* obtained better compression result than BV and BC but their technique does not permit a comparably fast access to the neighbors of a node. They compressed the intra-host links (that is, links between pages residing in the same host) by identifying through local indices six different types of blocks in the adjacency matrix, respectively dubbed: isolated 1-element, horizontal block, vertical block, L-shaped block, rectangular block,

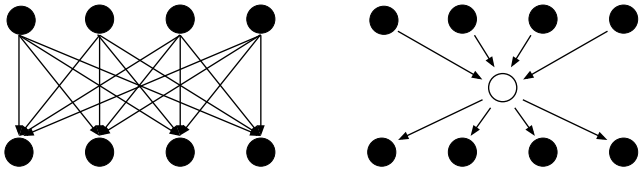


Figure 2.2: The method by Buehrer and Chellapilla [BC08] compresses a complete bipartite graph (left) by introducing a *virtual node* (right).

and diagonal block. Each block is represented by its first element, its type, and its size. The inter-host links are compressed by the same method used for the intra-host links, through resort to ad-hoc "new local indices" (refer to [AMN08] for details).

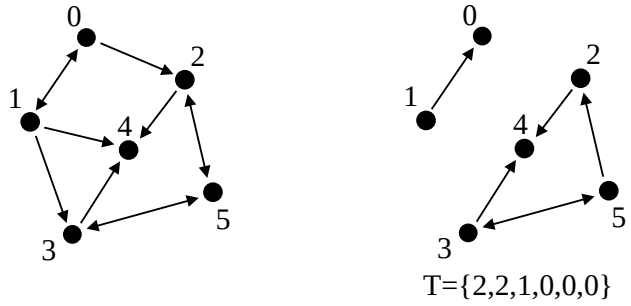
The recent method by Anh and Moffat [AM10] is one of the most simple method proposed in the literature. They partition the adjacency lists in group of h consecutive lists. For each group a *model* is defined as the union of the adjacency lists of the nodes in the group. Sequences of consecutive element in all the models are replaced by new symbols, and each element in a list is substituted by ordinal integer pertaining to the model's alphabet. The representation of a list with respect to previous list in the group (`XOR` operation) is used when it is shorter than the list itself. This method is indicated with $s = 1$, while $s = 2$ indicates that the same compression scheme is applied to the list composed of all the models.

2.3 A New Way to Encode by BFS

The compression method presented here is based on the topological structure of the Web Graph rather than on the underlying URLs. Instead of assigning indices to nodes based on the lexicographical ordering of their URLs, we perform a breadth-first traversal of G and index each node according to the order in which it is expanded. We refer to this process, and the compression it induces, as *Phase 1*. Following this, we compress in *Phase 2* all of the remaining links.

During the traversal of Phase 1, when expanding a node $v_i \in V$, we assign consecutive integer indices to its k_i (not yet expanded) neighbors, and also store the value of k_i. Once the traversal is over, all the links that belong to the breadth-first tree are encoded in the sequence $\{k_1, k_2, \ldots, k_{|V|}\}$, which we call the *traversal list*. In our experiments, the traversal allows to remove almost

$|V|-1$ links from the graph. Figure 2.3 shows an example of Phase 1: the graph with the indices assigned to nodes is displayed in Figure 2.3a, while Figure 2.3b shows the links remaining after the BFS and, below them, the *traversal list*. The compression ratio achieved by the present method is affected by the indices assignment of the BFS. In [CKL+09], Chierichetti *et al.* showed that finding an optimal assignment that minimizes $\sum_{(v_i,v_j)\in E} \log|i-j|$ is NP-hard.



(a) Indices assigned to the nodes.

(b) Remaining links after BFS and the *traversal list* T.

Figure 2.3: Illustrating Phase 1.

We now separately compress consecutive chunks of l nodes, where l is a prudently chosen value for what we call the *compression level*. Each compressed chunk is prefixed with the items of the traversal list that pertain to the nodes in the chunk: that is, assuming that the chunk C consists of the nodes v_i, ..., v_{i+l-1}, then the compressed representation of C is prefixed by the sequence $\{k_i, \ldots, k_{i+l-1}\}$.

It is easy to note that the traversal list is sufficient to decode all the edges belonging to the breadth-first tree in fact, the first node has links with nodes from 1 to k_1 (exclusive), the second with nodes from k_1 to $k_1 + k_2$ (exclusive) and so on.

In Phase 2, we encode the adjacency list A_i of each node $v_i \in V$ in a chunk C in increasing order. Each encoding consists of the integer gap between adjacent elements in the list and a *type indicator* chosen in the set $\{\alpha, \beta, \chi, \phi\}$ needed in decoding. With A_i^j denoting the jth element in A_i, we distinguish three main cases as follows:

1. $A_{i-1}^j \leq A_i^{j-1} < A_i^j$: the code is the string $\phi \cdot (A_i^j - A_i^{j-1} - 1)$

2. $A_i^{j-1} < A_{i-1}^j \leq A_i^j$: the code is the string $\beta \cdot (A_i^j - A_{i-1}^j)$

3. $A_i^{j-1} < A_i^j < A_{i-1}^j$: this splits in two subcases, namely,

   a) if $A_i^j - A_i^{j-1} - 1 \leq A_{i-1}^j - A_i^j - 1$ then the code is the string $\alpha \cdot (A_i^j - A_i^{j-1} - 1)$

   b) otherwise the code is the string $\chi \cdot (A_{i-1}^j - A_i^j - 1)$

The types α and ϕ encode the gap with respect to the previous element in the list (A_i^{j-1}), while β and χ are given with respect to the element in the same position of the adjacency list of the previous node (A_{i-1}^j).

When A_{i-1}^j does not exist it is replaced by A_k^j, where k ($k < i-1$ and $v_k \in C$) is the closest index to i for which the degree of v_k is not smaller than j, or by a ϕ-type code in the event that even such a node does not exist. In the following, we will refer to an encoding by its type. Table 2.3 displays the encoding that results under these conventions for the adjacency list of Table 2.2.

Node	Degree	Links
...	...	...
i	8	13 15 16 17 20 21 23 24
$i+1$	9	13 15 16 17 19 20 25 31 32
$i+2$	0	
$i+3$	2	15 16
...	...	...

Table 2.2: An adjacency list. It is assumed that the node v_i is the first node of a chunk.

As mentioned, the BFS encoding achieves that two nodes connected by a link are likely to be assigned close index values. Moreover, since two adjacent nodes in the Web Graph typically share many neighbors then the adjacency lists will feature similar consecutive lines. This leads to the emergence of four types of "redundancies" the exploitation of which is described, with the help of Table 2.4, as follows.

1. A run of identical lines is encoded by assigning a multiplier to the first line in the sequence;

Node	Degree	Links
...	...	...
i	8	ϕ 13 ϕ 1 ϕ 0 ϕ 0 ϕ 2 ϕ 0 ϕ 1 ϕ 0
$i+1$	9	β 0 β 0 β 0 β 0 χ 0 α 0 β 2 ϕ 5 ϕ 0
$i+2$	0	
$i+3$	2	β 2 α 0
...	...	...

Table 2.3: Encoding of the adjacency list of Table 2.2.

2. Since there are intervals of constant node degrees (such as, for example, the block formed by two consecutive "9" in the table) then the degrees of consecutive nodes are gap-encoded;

3. Whenever for some suitably fixed $\mathcal{L}_{\min}$ there is a sequence of at least $\mathcal{L}_{\min}$ identical elements (such as the block of ϕ 1's in the table), then this sequence is run-length encoded;

4. Finally, a *box* of identical rows (such as the biggest block in the table) exceeding a pre-set threshold size $\mathcal{A}_{\min}$ is run-length encoded.

We exploit the redundancies according to the order in which they are listed above, and if there is more than one box beginning at the same entry we choose the largest one.

In order to signal the third or fourth redundancy to the decoder we introduce a special character Σ, to be followed by a flag Σ_F denoting whether the redundancy starting with this element is of type 3 (Σ_F^3), 4 (Σ_F^4), or both (Σ_F^+).

For the third redundancy in the example we write "ϕ Σ Σ_f^3 1 1", where ϕ identifies a ϕ-type encoding, Σ_f^3 means a redundancy of the third type, the first 1 is the gap, and the second 1 is the number of times that the element appears minus $\mathcal{L}_{\min}$ (2 in this example).

To represent the fourth redundancy both width and height of the *box* need encoding, thus in our example we can write "β Σ Σ_F^4 0 7 5", where β is the code type, Σ_F^4 is the flag for the fourth type of redundancy, 0 is the gap, 7 is the width minus 1, and 5 is the height minus 2 (a box has at least two rows).

When a third and fourth type redundancy originate at the same entry, both are encoded in the format "$[\alpha\beta\gamma\phi]$ Σ Σ_F^+ *gap* $l - \mathcal{L}_{\min}$ $w_b - 1$ $h_b - 2$", where l is the number of identical elements on the same line starting from this element, and w_b and h_b are, respectively, the width and the height of the box.

Degree	Links									
...	...									
0										
9	β 7	ϕ 1	ϕ 1	ϕ 1	ϕ 0	ϕ 1	ϕ 1	ϕ 1	ϕ 1	
9	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 2	
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 1	ϕ 903
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 223	ϕ 900
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 1	α 0
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 1	β 0
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 1	β 0
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 1	β 0
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 1	β 0
10	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	α 76	α 232
9	β 0	β 1	β 0	β 0	β 0	β 0	β 0	β 0	β 0	
...	...									

Table 2.4: Exploiting redundancies in adjacency lists.

Table 2.5 shows the encoding resulting from this treatment.

Lines	Degree	Links			
...	...	...			
0	0				
0	9	β 7	$\phi\ \Sigma\ \Sigma_F^3$ 1 1	ϕ 0	$\phi\ \Sigma\ \Sigma_F^3$ 1 2
0	0	$\beta\ \Sigma\ \Sigma_F^4$ 0 7 5	β 1	$\beta\ \Sigma\ \Sigma_F^3$ 0 4	β 2
0	1	β 1	ϕ 903		
0	0	β 223	ϕ 900		
0	0	β 1	α 0		
3	0	β 1	β 0		
0	0	α 76	α 232		
0	-1	β 0			
...	...	...			

Table 2.5: An example of adjacency list encoding exploiting redundancies.

We observe that we do not need to explicitly write ϕ characters, which are implicit in $A_{i-1}^{j} \leq A_{i}^{j-1}$, a condition easily testable at the decoder. Each chunk is independently encoded as follows. We encode the characters α, β and χ as

well as Σ_F by Huffman-code. Gaps, the special character Σ (Σ is an integer that does not appear as a gap) and other integers are encoded using the ad-hoc π-code described in the Chapter 4; to encode each chunk the best π-code for that chunk is used. When a gap g could be negative (as with degrees), then we encode $2g$ if g is positive, and $2|g| - 1$ when g is negative.

2.4 Experimental Results

Table 2.8 reports sizes expressed in bits per link of compressed graphs. We used datasets[3] collected by Boldi and Vigna [BV04] (salient statistics in Table 2.7). Many of these datasets were gathered using UbiCrawler [BCSV04] by different

	$R = 3$		$R = \infty$	
	BFS		BFS	
arabic-2005	2.58	2.81	1.87	1.99
cnr-2000	3.92	3.56	3.23	2.84
eu-2005	4.83	5.17	4.05	4.38
in-2004	3.05	2.82	2.35	2.17
indochina-2004	2.03	2.06	1.46	1.47
uk-2002	3.28	3.00	2.46	2.22

Table 2.6: The results in bits per link of the BV method using an ordering based on the BFS.

laboratories. As shown in Table 2.8, with a compression level $l = 10^3$ the present method yielded consistently better results than BV [BV04], BC [BC08], Asano *et al.* [AMN08], and AM [AM10].

Table 2.6 displays also the results of the BV method using an ordering of the URLs induced by the BFS; This indicates that BV does not take advantage of an ordering based on BFS. The BV highest compression scores ($R = \infty$) are comparable to those we obtain at level 8, while those for general usage ($R = 3$) are comparable to level 4 as shown in Table 2.9.

[3]Datasets and *WebGraph* can be downloaded from `http://webgraph.dsi.unimi.it/`.

	Nodes	Links	Avg Degree
arabic-2005	22 744 080	639 999 458	28.1
cnr-2000	325 557	3 216 152	9.9
eu-2005	862 664	19 235 140	22.3
in-2004	1 382 908	16 917 053	12.2
indochina-2004	7 414 866	194 109 311	26.2
it-2004	41 291 594	1 150 725 436	27.9
sk-2005	50 636 154	1 949 412 601	38.5
uk-2002	18 520 486	298 113 762	16.1
uk-2005	39 459 925	936 364 282	23.7
uk-2006-05	77 741 046	2 965 197 340	38.1
uk-2006-12	103 098 631	3 768 836 665	36.6
uk-2007-05	105 896 555	3 738 733 648	35.3
uk-union	133 633 040	5 507 679 822	41.2
webbase-2001	118 142 155	1 019 903 190	8.6

Table 2.7: Statistics of datasets used for tests.

	BV	BC	Asano *et al.*	AM	BFS
arabic-2005	1.99	1.81	-	-	1.62
cnr-2000	2.84	-	1.99	-	1.84
eu-2005	4.38	2.90	2.78	3.55	2.70
in-2004	2.17	-	1.71	-	1.45
indochina-2004	1.47	-	-	-	1.01
it-2004	1.99	1.67	-	-	1.54
sk-2005	2.86	2.46	-	2.26	1.97
uk-2002	2.22	1.95	-	-	1.86
uk-2005	1.70	1.42	-	-	1.56
uk-2006-05	2.12	1.95	-	-	1.84
uk-2006-12	2.08	-	-	-	1.37
uk-2007-05	1.95	-	-	-	1.44
uk-union	1.87	-	-	-	1.83
webbase-2001	3.08	3.01	-	2.95	2.87

Table 2.8: Compressed sizes in bits per link. The BV results are obtained with $R = \infty$, AN for $s = 2$, and BFS for $l = 10^3$.

	BV		BFS	
	$R = 3$	$R = \infty$	$l = 4$	$l = 8$
arabic-2005	2.81	1.99	2.85	2.30
cnr-2000	3.56	2.84	3.33	2.68
eu-2005	5.17	4.38	4.20	3.48
in-2004	2.82	2.17	2.85	2.19
indochina-2004	2.06	1.47	2.09	1.57
it-2004	2.80	1.99	2.83	2.20
sk-2005	3.88	2.86	3.24	2.62
uk-2002	3.00	2.22	3.33	2.62
uk-2005	2.36	1.70	2.80	2.19
uk-2006-05	2.90	2.12	2.92	2.39
uk-2006-12	2.78	2.08	2.48	1.94
uk-2007-05	2.70	1.95	2.59	2.03
uk-union	2.64	1.87	2.86	2.38
webbase-2001	3.74	3.08	4.62	3.77

Table 2.9: Compressed sizes in bits per link.

2.5 Performances

As said one of the behavior of the proposed method is a fast access to the adjacency list of the graph. In this section we compare the performances to retrieve information in the compressed graphs, using the method presented here and the BV (WebGraph version 2.4.5); among all the techniques of compression, BV is the only implementation available on the Internet.

To randomly access the graph we need to store the offset of the first element of each chunk, but the results shown here do not account for these offsets. In fact, we do need N/l offsets. BV use $l = 1$ in their tests, which are slowed down by about 50% setting $l = 4$. In BC an offset per node is required. Asano *et al.* and Anh and Moffat do not provide information about offsets. In order to recover the links of the BFS tree, we also need to store for the first node u of each chunk the smallest index of a node v such that (u, v) belongs to the BFS tree. In total, this charges an extra $(b + k)/l$ bits per node, where b bits are charged by the offset and k by the index of the node. With r the bits per link and d the average degree of a graph, b requires at most $2 + \log_2(lrd)$ bits and k at most $2 + \log_2 l$ bits by Elias-Fano encoding [Eli74a, Fan71]. Using the same encoding, BC and BV require $2 + \log_2(rd)$ bits per node to represent

	BV	BFS	
	$R = 3$	$l = 8$	$l = 4$
arabic-2005	2.94 μs	3.69 μs	2.47 μs
cnr-2000	1.40 μs	1.72 μs	1.18 μs
eu-2005	2.85 μs	3.74 μs	2.57 μs
in-2004	1.62 μs	1.85 μs	1.27 μs
indochina-2004	2.50 μs	2.77 μs	1.92 μs
it-2004	3.06 μs	3.51 μs	2.38 μs
sk-2005	4.04 μs	5.36 μs	3.44 μs
uk-2002	2.05 μs	2.58 μs	1.86 μs
uk-2005	2.68 μs	3.03 μs	2.15 μs
uk-2006-12	3.53 μs	4.18 μs	2.73 μs
uk-2007-05	3.51 μs	4.25 μs	2.76 μs
uk-union	4.51 μs	5.64 μs	3.79 μs
webbase-2001	1.49 μs	1.87 μs	1.24 μs

Table 2.10: Average times to retrieve the adjacency list of a node.

	BV $R = \infty$
cnr-2000	0.48 ms
in-2004	0.11 ms
indochina-2004	3.85 ms
it-2004	2.72 ms

Table 2.11: Average times to retrieve the adjacency list of a node, of BV high compression mode.

any offset. Since, for $l > 1$, $(4 + \log_2(l^2 rd))/l < 2 + \log_2(rd)$, then we need less memory to store this information. Currently, in our implementation we use 64 bits per node.

Table 2.10 displays average times to retrieve adjacencies of 10 million random nodes. The tests run on an Intel Core i7-920 2.66 GHz with 6 GB main memory and 8 MB of cache memory under Linux 2.6.26. For the tests, we use the original implementation of BV and an implementation of the BFS method freely available at `http://dia.uniroma3.it/~drovandi/software.php`; both implementations are written in `Java` and run under `java SE` version 1.6.0. Table 2.10 shows the BV general usage version ($R = 3$) performs better than the BFS method with compression level 8. However, setting for $l = 4$ the BFS

Algorithm isNeighbour(v_i,v_j)
$v_f :=$ the first node of chunk C ($v_i \in C$)
$a := \sum_{h \leftarrow 1}^{f-1} k_h + 1$
for $h \leftarrow f$ to $i - 1$
    $a \leftarrow a + k_h$
end for
if $a \leq j < a + k_i$ **then return** *true*
if $j \geq a + k_i$ **then return** *false*
$A_i \leftarrow$ the adjacency list of v_i
if $v_j \in A_i$ **then return** *true*
return *false*

Figure 2.4: Algorithm to check if the directed link (v_i, v_j) exists.

method becomes faster. Table 2.11 shows that the BV high compression mode is much slower than BFS method; in fact, it is in the order of the milliseconds.

By virtue of the underlying BFS, we can implement a fast query to check whether or not a link (v_i, v_j) exists. In fact, we know that a node v_i has k_i links that belong to the BFS tree, say, to v_a, ..., v_{a+k_i-1}. We also know that v_i does not have any link with a node v_b where $b \geq a + k_i$, so we need to generate the adjacency list of v_i only if $j < a$. Figure 2.4 displays the pseudocode of this query. Table 2.12 shows average times to test the connectivity of 10 million pairs of random nodes. The average time is less than 60% of the retrieval time.

2.6 PageRank on Compressed Graphs

The compressed graph generated by our method is suitable also for algorithms that need to scan sequentially the whole graph. Among these, we tested the well known Page-Rank algorithm [BP98, PBMW99]. This algorithm needs to perform more iterations of the graph until it converges to a result. During each iteration the whole graph must be visited; since the order in which the nodes are visited is not important, it is possible to read the compressed graph file sequentially, reducing to the minimum the number of cache-misses and page-faults. In this case a higher compression level gives better performances.

In [KCA09], Karande *et al.* show that some algorithms (ie. PageRank, HITS and SALSA [Kle99]), perform well with graphs compressed using the BC method. They propose two approaches for the PageRank algorithm based

	BFS	
	$l = 8$	$l = 4$
arabic-2005	1.98 μs	1.42 μs
cnr-2000	0.99 μs	0.71 μs
eu-2005	2.08 μs	1.47 μs
in-2004	1.10 μs	0.76 μs
indochina-2004	1.64 μs	1.22 μs
it-2004	2.09 μs	1.47 μs
sk-2005	2.78 μs	1.93 μs
uk-2002	1.49 μs	1.06 μs
uk-2005	1.78 μs	1.28 μs
uk-2006-12	2.31 μs	1.58 μs
uk-2007-05	2.28 μs	1.55 μs
uk-union	2.77 μs	1.83 μs
webbase-2001	1.09 μs	0.80 μs

Table 2.12: Average times to test adjacency between pairs of random nodes.

on Black-box, and Markov chain. Table 2.13 shows the times per iteration of PageRank running on graphs compressed with BC and the method presented here. Unfortunately, the implementation of BC is not available so we use the numerical results presented in [KCA09] only as reference; their experiments ran on a quad-code Intel Xeon at 3.0 GHz with 16 GB of main memory, and the implementations of the algorithms used only one core.

	BC		BFS	
	Black-box	Markov chain	$l = 10^3$	$l = 8$
eu-2005	1.58 s	1.50 s	0.66 s	0.78 s
uk-2005	60.80 s	60.06 s	23.26 s	26.59 s

Table 2.13: PageRank time per iteration.

As said, we cannot compare directly the results. However, considering the ratio between the time per iteration (tpi) of the dataset `uk-2005` and the tpi of `eu-2005`, we can point out that this ratio for BC method is 38 while using the BFS method it is 35. From the numerical results it appears that the PageRank algorithm scales better on a graph compressed using the BFS encoding method.

2.7 Conclusions and Open Problems

We have proposed a new way to compress the Web Graph and other graphs of comparable structure. In fact, we assume no a priori knowledge of the graph, and in contrast with previous works based on lexicographic ordering of URLs we use a traversal to order nodes. The size of the compressed files is smaller than that of Asano *et al.* [AMN08], considered the current state of the art. The average retrieval time is comparable to that of BV [BV04].

The average bits per link that we have shown are obtained using a random node as root of the BFS. Changing the root of the BFS the compression results can vary.

Open Problem 2.1 *What is the best node to be elected as root of the BFS?*

We have introduced a fast query to check whether two nodes are connected, without the necessity to generate an entire adjacency list; however, the following problem is still open:

Open Problem 2.2 *Can we extend the set of primitive queries for compressed graphs?*

The results presented in this chapter shown that two bits per link are sufficient to encode a graph, but we have no formal lower bound. Regarding the minimum number of bits required two problems are of interest:

Open Problem 2.3 *Is it possible to achieve better results with a BFS ordering of the nodes?*

Open Problem 2.4 *What is the minimum number of bits required to encode a link?*

Chapter 3

Extension of the Web Graph Method to Social Networks

Is the compression method by BFS suitable only for the Web Graph or can it be used for more general kind of graphs? To answer this question we investigate the compression problem of different kind of real networks, in particular social networks. As shown in [CKL+09] it is more difficult to compress a social network that the Web Graph; the bits per link required are at least more than three times than those for the Web Graph.

3.1 Background

Social Networks have a different topology with respect to the Web Graph. The two main properties of the Web Graph, locality and similarity, induce a particular subgraph that is exploited to achieve good compression results. This peculiar subgraph is a high connected component in which two nodes have a high probability to share many neighbors. Listing the adjacency lists of the nodes belonging to this subgraph it is possible to represent all the common elements in a succinct way.

Unfortunately applying this technique to other types of networks does not bring to the same good results. Social networks share some properties with the Web Graph, as the degree distribution, but the communities are not so close as in the Web Graph. All the pages of a web site are likely to link to each other, while the friends of a person usually do not know all the other friends but just a subgroup of them, and each friend belongs to many different groups

of friends, said communities.

The main propriety to be exploited in the social network is the *reciprocity* of the links. If node A has a link directed to node B then B has an high probability to have a link directed to A.

3.2 Related Work

The compression of social networks is a recent problem, there just few works on this.

The method by Chierichetti *et al.* [CKL$^+$09] exploits the property of reciprocity of the links of social networks. The method proposed in [CKL$^+$09] is an extension of the method by Boldi and Vigna [BV04] (Section 2.2). Let u be a node, its encoding consists of the following information (as defined in [BV04]).

1. base information - a bit specifying if u has a self loop, and the outdegree of u, minus 1 if u has a self-loop and minus the number of reciprocal edges of u;

2. copying - the node v used as prototype for the links of u. It is encoded the difference $u - v$ since $u \geq v$. The copying list is composed of a bit for each out-neighbor of v to indicate if that out-neighbor is an out-neighbor of u (if $u = v$ no copying list is performed);

3. residual edges - all the links not belonging to the copying list are gap-encoded. Say $u_1, \ldots, u_k$ the out-neighbor of u not present in the copying list, a bit indicate if $u > u_1$ and the encoding is $|u - u_1| - 1$, $u_2 - u_1 - 1$, ..., $u_k - u_{k-1} - 1$;

4. reciprocal edges - for all the out-neighbors v such that $v > u$ a bit is added to the encoding to represent if the edge (v, u) is also present in the network. If this edge belongs to E it is removed from the set since it is encoded in the adjacency list of u.

The reciprocal edges are exploited in the fourth point. This method achieves good results; however it is not suitable to retrieve the adjacency list of a node since to discover reciprocal edges a scan of the whole compressed file is needed.

In [CKL$^+$09] a new ordering of the nodes is introduced. The Boldi and Vigna method is based on a lexicographic ordering of the nodes, instead the technique by Chierichetti *et al.* uses a new heuristic to order the nodes in way called shingle ordering. The problem of finding a permutation of the nodes π :

$V \leftarrow N$ (labeling of the nodes) to minimize the summation $\sum_{(u,v)\in E} \log(|\pi(u) - \pi(v)|)$ is an NP-hard problem.

A different approach to exploit the reciprocity of the links is proposed in [MP10] by Maserrat and Pei. They proposed a method that allow to answer to the query *"What are the nodes in the neighborhood of u?"*. This is an extension of all the previous methods of graph compression that allow only to retrieve the out-neighborhood in sublinear time.

This method is based on the multi-position (MP_k) linearization of a graph. An MP_k of a graph is a list of the nodes of the graph (with possible multi entry for each node) such that if the edge (u, v) exists there is at least a pair of occurrences of u and v that are at distance at most k.

Maserrat and Pei use a data structure called Eulerian Data Structure. A compact representation of a graph using this data structure is composed of an optimal MP_1 linearization L of the graph and, said $v(i)$ the node that appears in position i in L, two information for each position i:

1. local information - two bits specifying if edges $(v(i-1), v(i))$ and $(v(i), v(i-1))$ exist or not;
2. pointer - a pointer to the next occurrence of $v(i)$. If this is the last occurrence then the pointer is set to the first occurrence of $v(i)$.

The Eulerian data structure allows to retrieve both the in-neighbors and out-neighbors of a node u.

3.3 Extended BFS Method

The compression schema presented in the previous chapter does not allow to perform the query for the in-neighbors of a node in sublinear time. Here we extend the BFS encoding method exploiting the reciprocity properties of the social networks, and providing queries for the out-neighbors and in-neighbors of a node.

For this purpose we do not consider the direct graph G but the extended directed graph $\overline{G}$. The extended directed graph is composed of the same set of nodes of G and it has an edge (u, v) if and only if at least one of the edges (u, v) and (v, u) are present in G. We encode $\overline{G}$ using the BFS method instead of G. Note that in $\overline{G}$ the neighborhood of a node u is composed of both the out-neighbor and in-neighbor of u in G.

Once we have compressed $\overline{G}$ we need to mark each edge (u, v) in order to know if it is the representation in G of (u, v), of (v, u) or of both of them. This

labeling must be repeated twice for each edge in G, since if the edge (u, v) exists in G, in $\overline{G}$ we have both (u, v) and (v, u). For these labels we use the Huffman encoding, that needs at most 2 bits; in fact, we need to encode only three symbols: in-neighbor, out-neighbor, or both. When we deal with graphs that do not present reciprocal edges the encoding needs only one bit. The labeling of the edges is trivial for undirected graphs, in which each edge is considered to be reciprocal, so we do not mark each edge but the graph itself as undirected.

3.4 Experimental Results

The data sets used for these tests are from the SNAP project[1] (Stanford Network Analysis Package). Main properties of the data sets are shown in Table 3.1 and the results of the tests are shown in Table 3.2.

We compare the results obtained using the method by Maserrat and Pei (MP) with the extended BFS method (eBFS) and the original BFS method.

Network	Nodes	Edges	AVG Degree	Fre
ca-CondMat	23 133	186 878	8.08	1.00
ca-HepPh	12 006	236 978	19.74	1.00
email-Enron	36 692	367 662	10.02	1.00
email-EuAll	265 009	418 956	1.58	0.26
p2p-Gnutella08	6 301	20 777	3.30	0.00
p2p-Gnutella24	26 518	65 369	2.47	0.00
soc-Slashdot0902	82 168	870 161	10.59	0.84
web-Stanford	281 903	2 312 497	8.20	0.28

Table 3.1: Main statistics of the data sets.

To retrieve both the in-neighbors and out-neighbors, a simple method is to compress both the direct and transposed graph. The transposed graph $G' = (V', E')$ of $G = (V, E)$ has the same set of nodes $V' = V$ and all the edges inverted that is for all $(u, v) \in E$ the edge (v, u) belongs to E'. To study how this kind of compression perform, we also run tests on the transposed graphs. With BFSd+t we indicate the summation of the results obtained from of the BFS compression scheme applied to the direct and transposed graphs.

[1] Datasets are available at `http://snap.stanford.edu/data/`.

We do not consider the method by Chierichetti *et al.* since no information on the source of their dataset is given in the article and because their method does not allow the retrieval of the in-neighborhood.

The properties *Fre* in Table 3.1 represents the fraction of reciprocal edges that is $Fre(G) = \frac{\text{number of reciprocal edges in G}}{\text{edges in G}}$.

Network	ca-CondMat	ca-HepPh	email-Enron	email-EuAll	p2p-Gnutella08	p2p-Gnutella24	soc-Slashdot0902	web-Stanford
MP	**6.77**	4.53	**7.26**	22.55	21.63	24.33	**12.14**	9.88
eBFS	7.45	**3.61**	8.04	**9.21**	**21.27**	**23.60**	14.31	11.13
BFSd+t	14.85	7.21	15.78	17.84	21.57	24.91	22.40	**8.39**
BFS	7.45	3.61	8.04	13.21	10.89	11.91	11.55	3.58

Table 3.2: The results in bits per link.

The results in Table 3.2 show that the eBFS method does not always perform better than MP, however it achieves comparable results. For the `email-EuAll` data set, eBFS outperforms MP halving its compression result. It is interesting to note that the BFS method achieves comparable results to eBFS, and that for three data sets, `p2p-Gnutella08`, `p2p-Gnutella24` and `web-Stanford`, the BFSd+t achieves comparable or better results than MP and eBFS.

3.5 Conclusions and Open Problems

The compression of social networks seems to be harder than the compression of the Web Graph. The most important peculiarity of this networks is the reciprocity of the links. This characteristic of the social networks can be exploited to achieve good compression results and it is possible to obtain a compression schema to perform in sublinear time the query *"What is the neighborhood of a node?"*.

To deal with these kind of graphs, in this chapter an extension of the method of compression by BFS was introduced. It allows to perform queries on the

out-neighbors and in-neighbors. This new method achieves compression results that are competitive with the results presented in the literature.

The compression results achieved with the new method are comparable, or sometimes better, with respect to the original method of compression by BFS when the fraction of reciprocal edges is high. When this fraction is low, it is interesting to note that the ad hoc techniques for social networks achieve the same or worst results that the BFS method applied to the original network and its transpose.

To encode a social network we need more bits that to encode the Web Graph, so the following open problem is one of the most important:

Open Problem 3.1 *What is the minimum number of bits per link required to encode a social network?*

Chapter 4

Prefix Codes for Power Law Distributions

The need for efficient integer compression arises in a variety of data processing tasks. In this thesis, we are interested in the compression of a sequence of integers representing a compressed graph. This kind of sequences has the property that the integers have a power law distribution; this suggests to adopt prefix codes for efficient compression. The emphasis on power law distributions is also motivated by the fact that many other graph parameters follow this kind of distribution: for instance the in-degrees of the Web Graph [BKM+00], or the betweenness centrality distribution in a scale free network [KHP+07].

In this chapter a new family of prefix codes is introduced. This family of code performs well under a distribution of the integers that follows a power law, in particular distributions described by a zeta function. Moreover, we investigate the more general problem of compression of sequences of integers whose probability distribution is unknown; such problem will be the topic of chapters 5 and 6.

4.1 Preliminaries

We assume familiarity with some basic concept about prefix codes. A source (M, P) is composed of a countable set M and a probability distribution P of each message such that $P : M \to (0, 1]$. A code for the source (M, P) is a function ρ that assigns to each message in M a distinct element of a codeword

set $C \subseteq B^*$ of size $|M|$. In the following we consider only binary codes[1], so $B = \{0, 1\}$. With $\beta(n)$ we indicate the binary encoding of an integer n. A code ρ is said to be uniquely decipherable if the extension $\rho^* : M^* \to C^*$ is a one-to-one function, and is said to be a prefix code if no codeword of C is the prefix of another. We indicate with $E_P(L_\rho)$ the expected length of a code ρ under a distribution P of the integers (in a more compact form we can use $E_P(\rho)$), and with H_P the entropy of a source (M, P). The entropy is defined as $H_P = -\sum_{i=1}^{|M|} P(m_i) \log(P(m_i))$. According to [Eli75], a code ρ is universal iff the ratio $E_P(L_\rho)/\max\{1, H_P\}$ is bounded above by a constant with $0 < H_P < \infty$; moreover, a code is said to be asymptotically optimal iff the previous ratio is bounded above by a function of which the limit for $H_P \to \infty$ is equal to 1.

The power law distribution of our interest is the zeta distribution. This is a discrete probability distribution defined as follows:

$$Z_\alpha(x) = \frac{1}{\zeta(\alpha)x^\alpha}$$

where x is a positive integer and $\zeta(\alpha)$ is the Riemann zeta function:

$$\zeta(\alpha) = \sum_{x=1}^{\infty} n^{-\alpha}$$

The Riemann zeta function assumes finite values for $\alpha > 1$. In this chapter, we focus on values of α in the range $(1; 2]$; in fact, in this range belongs many of the parameters of real distributions [AJB99, BA99].

The information entropy of the zeta distribution is:

$$H_\alpha = \sum_{x=1}^{\infty} \frac{\log\left(\zeta(\alpha)x^\alpha\right)}{\zeta(\alpha)x^\alpha}$$

To estimate the expected length of the code, we need to solve some summations, so we recall the Euler-Maclaurin summation formula [GKP94]:

$$\sum_{x=a}^{b} f(x) = \int_a^b f(x)dx + \frac{1}{2}[f(a) + f(b)] + \sum_{t=1}^{\infty} \frac{B_{2t}}{(2t)!} f^{(2t-1)}(x)\Big|_a^b$$

[1] In the following all logarithms are binary.

where B_i is the ith Bernoulli number ($B_2 = \frac{1}{6}$, $B_4 = -\frac{1}{30}$, $B_6 = \frac{1}{42}$,...). Subtracting $f(b)$ from both left and right we obtain:

$$\sum_{x=a}^{b-1} f(x) = \int_a^b f(x)dx - \frac{1}{2}f(x)\Big|_a^b + \sum_{t=1}^{\infty} \frac{B_{2t}}{(2t)!} f^{(2t-1)}(x)\Big|_a^b$$

that is for $f(x) = x^{-\alpha}$:

$$\sum_{x=a}^{b-1} x^{-\alpha} = \frac{1}{1-\alpha}x^{1-\alpha}\Big|_a^b - \frac{1}{2}x^{-\alpha}\Big|_a^b - \frac{\alpha}{12}x^{-\alpha-1}\Big|_a^b \tag{4.1}$$

with an error of:

$$\frac{\alpha(\alpha+1)(\alpha+2)}{720}x^{-\alpha-3}\Big|_a^b$$

From Equation 4.1, we can provide a framework to solve summations:

$$\begin{aligned}
&\sum_{j=0}^{\infty}\sum_{x=ac^j}^{abc^j-1}\frac{j}{\zeta(\alpha)x^{\alpha}}\\
&=\frac{1}{\zeta(\alpha)}\sum_{j=0}^{\infty}\Bigg[\frac{a^{1-\alpha}(b^{1-\alpha}-1)}{1-\alpha}j(c^{1-\alpha})^j - \frac{a^{-\alpha}(b^{-\alpha}-1)}{2}j(c^{-\alpha})^j\\
&\quad - \frac{\alpha a^{-1-\alpha}(b^{-1-\alpha}-1)}{12}j(c^{-1-\alpha})^j\Bigg]\\
&=\frac{1}{\zeta(\alpha)}\Bigg[\frac{a^{1-\alpha}(b^{1-\alpha}-1)}{1-\alpha}\frac{c^{1-\alpha}}{(1-c^{1-\alpha})^2} - \frac{a^{-\alpha}(b^{-\alpha}-1)}{2}\frac{c^{-\alpha}}{(1-c^{-\alpha})^2}\\
&\quad - \frac{\alpha}{12}(a^{-1-\alpha}(b^{-1-\alpha}-1))\frac{c^{-1-\alpha}}{(1-c^{-1-\alpha})^2}\Bigg]
\end{aligned}$$

If $b = c$:

$$\sum_{j=0}^{\infty}\sum_{x=ac^j}^{ac^{j+1}-1}\frac{j}{\zeta(\alpha)x^{\alpha}} = \frac{1}{\zeta(\alpha)}\Bigg[\frac{a^{1-\alpha}}{(\alpha-1)(c^{\alpha-1}-1)} + \frac{a^{-\alpha}}{2(c^{\alpha}-1)} + \frac{\alpha}{12}\frac{a^{-1-\alpha}}{c^{1+\alpha}-1}\Bigg]$$

In the same way:

$$\begin{aligned}
&\sum_{j=0}^{\infty}\sum_{x=ac^j}^{abc^j-1}\frac{1}{\zeta(\alpha)x^\alpha}\\
&=\frac{1}{\zeta(\alpha)}\sum_{j=0}^{\infty}\left[\frac{a^{1-\alpha}(b^{1-\alpha}-1)}{1-\alpha}(c^{1-\alpha})^j-\frac{a^{-\alpha}(b^{-\alpha}-1)}{2}(c^{-\alpha})^j\right.\\
&\quad\left.-\frac{\alpha a^{-1-\alpha}(b^{-1-\alpha}-1)}{12}(c^{-1-\alpha})^j\right]\\
&=\frac{1}{\zeta(\alpha)}\left[\frac{a^{1-\alpha}(b^{1-\alpha}-1)}{(1-\alpha)(1-c^{1-\alpha})}-\frac{a^{-\alpha}(b^{-\alpha}-1)}{2(1-c^{-\alpha})}-\frac{\alpha}{12}\frac{a^{-1-\alpha}(b^{-1-\alpha}-1)}{1-c^{-1-\alpha}}\right]
\end{aligned}$$

In the following we use the approximation $H_P \approx E_P(\log(n))$ presented by Wyner in [Wyn72]; it is possible to use this approximation since the entropy H_P and $E_P(\log(n))$ converge or diverge together. In [Eli75] the result by Wyner was strengthen by the inequality:

$$E_P(\log(n)) \leq H_P \leq 1 + k + \left(1+\frac{1}{k}\right) E_P(\log(n))$$

where k is a value in $\mathbb{N}^+$.

4.2 Related Work

Entropy encodings has been developed extensively in lossless data compression schemes. Among these codes, the most commons are the Huffman code [Huf52] and the arithmetic coding [Ris76], that are suitable for the encoding of a source of unknown entropy. Both these techniques require a finite alphabet and a dictionary is needed for the decoding. However, if an approximation of the source entropy is known, an universal code (introduced by Elias in [Eli74b]) may perform better since there is no necessity of a dictionary for decoding and one can throw away any limitation on the dimension of the alphabet.

Many prefix encoding techniques have been proposed for different source distributions. For power law distributions of the integers, the most important codes are γ- and δ-Code by Elias [Eli75], and Boldi and Vigna ζ-Code [BV05].

The definitions of these codes are based on two simple encoding methods: unary code and truncated binary encoding. The unary representation of an integer $n \geq 1$ is composed of $n-1$ zeros followed by 1. This representation

performs well for the encoding of small integers (for example, for an exponential distribution). The second important code is the truncated binary encoding; this technique is useful to encode an integer in an interval $[0, N)$. When N is a power of 2, one can use $\log(N)$ bits to encode each integer. When N is not a power of 2, the usage of $\lceil \log(N) \rceil$ bits to encode the integers is not the best possible choice in fact, the last $2^{\lceil \log(N) \rceil} - N$ codewords are not exploited. Using the truncated binary encoding, integers in the interval $[0, M)$ are encoded using $\lceil \log(N) \rceil - 1$ bits, where $M = 2^{\lceil \log(N) \rceil - 1}$, and integers in the interval $[M, N)$ using $\lceil \log(N) \rceil$ bits.

The basic idea of Elias γ-Code [Eli75] is to prefix an integer n with the unary encoding of its bit-length, for instance the integer 5 ($\beta(5) = 101$, the bit-length is 3) is encoded as 001 01, since its bit-length unary code is 001. In the δ-Code [Eli75] the unary representation of the length is substituted by its γ encoding, so 5 will become 011 01. Note that the most significant bit is not encoded; in fact, it is always 1 since we deal only with positive integers. It is easy to see that the expected length of γ is $E_P(L_\gamma) < 2E_P(\log(n)) + 1 = 2H_P + 1$, since the length is $L_\gamma(n) = 2\lfloor \log(n) \rfloor + 1 \leq 2\log(n) + 1$. This code is universal but not asymptotically optimal. In fact, $\lim_{H_P \to \infty} E_P(L_\gamma)/H_P = 2$. The δ-Code works better for large H_P and one can quickly verify that it is both universal and asymptotically optimal.

The most important code introduced with the aim of graph compression is the ζ-Code, presented by Boldi and Vigna in [BV05]. The ζ-Code was introduced to fit better a power law distribution with smaller exponent (α close to 1) than the δ-Code. Given a shrinking factor $k \in \mathbb{N}^+$, they divide the set of natural numbers in intervals of the kind $I_h = [2^{hk}, 2^{(h+1)k})$. An integer n, belonging to interval I_h, is ζ_k-coded by writing $h + 1$ in unary followed by the truncated binary encoding of $n - 2^{hk}$ in the interval $[0, 2^{(h+1)k} - 2^{hk})$.

As described in [BV05] the expected length of a ζ_k-Code is:

$$\frac{1}{\zeta(\alpha)} \left[\frac{k + 2^{1-\alpha}}{(\alpha - 1)(1 - 2^{k(1-\alpha)})} + \frac{k + 2^{-\alpha}}{2(1 - 2^{-k\alpha})} + \frac{\alpha(k + 2^{-\alpha-1})}{12(1 - 2^{k(-\alpha-1)})} \right] \tag{4.2}$$

with an error of 9% when $\alpha \leq 1.6$.

4.3 The Family of π-Codes

In this section we introduce π-Codes, a new family of universal codes for the integers. These codes perform well under a zeta distribution of the source, and, under this condition, they are better suited than δ- and ζ- codes.

n	$\gamma = \zeta_1$	ζ_2	ζ_3	δ
1	1	10	111	1
2	010	110	1010	0100
3	011	111	1011	0101
4	00100	01000	1100	01100
5	00101	01001	1101	01101
6	00110	01010	1110	01110
7	00111	01011	1111	01111
8	0001000	011000	0100000	00100000

Table 4.1: The first codewords of ζ_k ($1 \leq k \leq 3$) and δ. Note that $\gamma = \zeta_1$.

Let n be a positive integer, $\beta(n)$ its binary representation, and $h = 1 + \lfloor \log(n) \rfloor$. Having fixed a positive integer k, we represent n using $k+h+\lceil \frac{h}{2^k} \rceil - 1$ bits. More in detail, say $h = 2^k l - c$ ($l > 0$ and $0 \leq c < 2^k$), then the π_k-encoding of n is produced by writing the unary representation of l, which is followed by the k bits needed to encode c, and finally by the rightmost $h-1$ bits of $\beta(n)$.

For instance, the π_2-encoding of 21 is 01 11 0101 since $\beta(21)$ is 10101 and we can write $h = 2^2 \cdot 2 - 3 = 5$; the prefix of the encoding is the unary representation of $l = 2$, followed by the 2 bits needed to encode the value of $c = 3$, the suffix is formed by the $h-1$ least significant digits of $\beta(21)$.

Table 4.2 shows the first codewords of the γ-, δ- and π- codes. Note that the π_0-Code is equal to the γ-Code and π_1-Code is equivalent to the ζ_2-Code.

An alternative definition of π-Code (that we use in the following) is:

Definition 4.1 *Given two integers $k \geq 0$ and $n \geq 1$, define $\beta(n)$ its binary representation and $h = \lfloor \log(n) \rfloor$. The π_k-encoding of n is obtained writing a 1 in position i ($0 \leq i < k$) if $\lceil \frac{h}{2^i} \rceil$ is even, otherwise 0; then writing the rightmost h bits of $\beta(n)$. To this partial encoding we add as prefix the unary representation of $1 + \lfloor \frac{h}{2^k} \rfloor$.*

It is also possible to define π-Codes in a recursive way (note that $\pi_0 = \gamma$):

Definition 4.2 *Given two positive integers $k \geq 1$ and $n \geq 1$, say $\beta(n)$ the binary representation of n and $h = \lfloor \log(n) \rfloor$. The π_k-encoding of n is obtained writing 1 if h is odd, 0 otherwise, followed by the π_{k-1}-encoding of $\lfloor n/2^{\lfloor h/2 \rfloor} \rfloor$, and then writing the rightmost remaining $\lfloor \frac{h}{2} \rfloor$ bits of b.*

n	$\pi_0 = \zeta_1 = \gamma$	$\pi_1 \equiv \zeta_2$	π_2	π_3	δ
1	1	11	111	1111	1
2	010	100	1100	11100	0100
3	011	101	1101	11101	0101
4	00100	01100	10100	110100	01100
5	00101	01101	10101	110101	01101
6	00110	01110	10110	110110	01110
7	00111	01111	10111	110111	01111
8	0001000	010000	100000	1100000	00100000
9	0001001	010001	100001	1100001	00100001
10	0001010	010010	100010	1100010	00100010
11	0001011	010011	100011	1100011	00100011
12	0001100	010100	100100	1100100	00100100
13	0001101	010101	100101	1100101	00100101
14	0001110	010110	100110	1100110	00100110
15	0001111	010111	100111	1100111	00100111
16	000010000	00110000	01110000	10110000	001010000

Table 4.2: The initial segment of π_k-codes ($0 \leq k \leq 3$) versus δ. Note that $\pi_0 = \gamma = \zeta_1$ and $\pi_1 \equiv \zeta_2$.

In the context of Web Graph compression we have used a modified version of π-codes in which 0 is encoded by 1 and any other positive integer n is encoded with a 0 followed by the π-encoding of n. However, here we use the original definition of π-codes to show its properties.

Expected Length of π-Codes

Let n be a positive integer and $h = \lfloor \log(n) \rfloor$. Fixed a positive integer k ($k \geq 0$), we represent n using $k + 1 + h + \lfloor \frac{h}{2^k} \rfloor$ bits, as described in Definition 4.1.

First, we observe that an integer x in the interval $[2^h, 2^{h+1})$, where $h = 2^k j + i$ ($j \geq 0$ and $0 \leq i < 2^k$), is π_k-coded using $h + j + k + 1$ bits then, we can write the expected length as[2]:

$$E(\pi_k) = \sum_{j=0}^{\infty} \sum_{i=0}^{2^k-1} \sum_{x=2^h}^{2^{h+1}-1} (h + j + k + 1) \frac{1}{\zeta(\alpha) x^{\alpha}}$$

[2]We omit the P in the notation $E_P(\pi_k)$, assuming that in this context the probability P is a zeta distribution.

That is:

$$\frac{1}{\zeta(\alpha)}\left[(1+k)\sum_{x=1}^{\infty}\frac{1}{x^\alpha}+(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k j}}^{2^{2^k(j+1)}-1}\frac{j}{x^\alpha}+\sum_{i=1}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{i}{x^\alpha}\right]$$
$$=1+k+\frac{1}{\zeta(\alpha)}\left[(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k j}}^{2^{2^k(j+1)}-1}\frac{j}{x^\alpha}+\sum_{i=1}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{i}{x^\alpha}\right] \quad (4.3)$$

The first summation can be easily solved using the framework defined in Section 4.1 (say $a=1$ and $b=c=2^{2^k}$):

$$\frac{1+2^k}{\zeta(\alpha)}\sum_{j=0}^{\infty}\sum_{x=2^{2^k j}}^{2^{2^k(j+1)}-1}\frac{j}{x^\alpha}$$
$$=\frac{1+2^k}{\zeta(\alpha)}\left[\frac{1}{(\alpha-1)(2^{2^k(\alpha-1)}-1)}+\frac{1}{2(2^{2^k\alpha}-1)}+\frac{\alpha}{12(2^{2^k(1+\alpha)}-1)}\right]$$

We can then calculate the second summation in the same way (say $a=2^i$, $b=2$ and $c=2^{2^k}$):

$$\frac{1}{\zeta(\alpha)}\sum_{i=1}^{2^k-1} i\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1} x^{-\alpha}$$
$$=\frac{1}{\zeta(\alpha)}\sum_{i=1}^{2^k-1} i\left[\frac{2^{1-\alpha}-1}{(1-\alpha)(1-2^{2^k(1-\alpha)})}(2^{1-\alpha})^i-\frac{2^{-\alpha}-1}{2(1-2^{-2^k\alpha})}(2^{-\alpha})^i\right.$$
$$\left.-\frac{\alpha}{12}\frac{2^{-1-\alpha}-1}{1-2^{2^k(-1-\alpha)}}(2^{-1-\alpha})^i\right]$$
$$=\frac{1}{\zeta(\alpha)}\left[\frac{2^{1-\alpha}-1}{(1-\alpha)(1-2^{2^k(1-\alpha)})}\left(2^{1-\alpha}\frac{1-2^{(1-\alpha)(2^k-1)}}{(1-2^{1-\alpha})^2}-\frac{(2^k-1)2^{2^k(1-\alpha)}}{1-2^{1-\alpha}}\right)\right.$$
$$-\frac{2^{-\alpha}-1}{2(1-2^{-2^k\alpha})}\left(2^{-\alpha}\frac{1-2^{-\alpha(2^k-1)}}{(1-2^{-\alpha})^2}-\frac{(2^k-1)2^{-2^k\alpha}}{1-2^{-\alpha}}\right)$$
$$\left.-\frac{\alpha}{12}\frac{2^{-1-\alpha}-1}{1-2^{2^k(-1-\alpha)}}\left(2^{-1-\alpha}\frac{1-2^{(-1-\alpha)(2^k-1)}}{(1-2^{-1-\alpha})^2}-\frac{(2^k-1)2^{2^k(-1-\alpha)}}{1-2^{-1-\alpha}}\right)\right]$$

Putting together the two summations we obtain the expected length of a π_k-Code:

$$\begin{aligned}E(\pi_k) =& \frac{1+2^k}{\zeta(\alpha)}\left[\frac{1}{(\alpha-1)(2^{2^k(\alpha-1)}-1)}+\frac{1}{2(2^{2^k\alpha}-1)}+\frac{\alpha}{12(2^{2^k(1+\alpha)}-1)}\right]\\ &+\frac{1}{\zeta(\alpha)}\left[\frac{2^{1-\alpha}-1}{(1-\alpha)(1-2^{2^k(1-\alpha)})}\left(2^{1-\alpha}\frac{1-2^{(1-\alpha)(2^k-1)}}{(1-2^{1-\alpha})^2}-\frac{(2^k-1)2^{2^k(1-\alpha)}}{1-2^{1-\alpha}}\right)\right.\\ &-\frac{2^{-\alpha}-1}{2(1-2^{-2^k\alpha})}\left(2^{-\alpha}\frac{1-2^{-\alpha(2^k-1)}}{(1-2^{-\alpha})^2}-\frac{(2^k-1)2^{-2^k\alpha}}{1-2^{-\alpha}}\right)\\ &\left.-\frac{\alpha}{12}\frac{2^{-1-\alpha}-1}{1-2^{2^k(-1-\alpha)}}\left(2^{-1-\alpha}\frac{1-2^{(-1-\alpha)(2^k-1)}}{(1-2^{-1-\alpha})^2}-\frac{(2^k-1)2^{2^k(-1-\alpha)}}{1-2^{-1-\alpha}}\right)\right]\\ &+k+1\end{aligned} \tag{4.4}$$

The estimated error is:

$$\frac{\alpha(\alpha+1)(\alpha+2)}{720\zeta(\alpha)}\left[\frac{1+2^k}{2^{2^k(\alpha+3)}-1}+\frac{1-2^{-\alpha-3}}{1-2^{2^k(-\alpha-3)}}\sum_{i=0}^{2^k-1} i(2^{-\alpha-3})^i\right] < 1.4\cdot 10^{-3}$$

Kullback-Leibler Distance

To obtain a measure of the goodness of π-Codes, we show that the expected length of the code is close to the entropy. The distance D between two discrete distributions P and Q, called Kullback-Leibler distance (or relative entropy), is defined as:

$$\begin{aligned}D(P,Q) &= \sum_k P_k \log\left(\frac{P_k}{Q_k}\right)\\ &= \sum_k P_k \log\left(\frac{1}{Q_k}\right) - \sum_k P_k \log\left(\frac{1}{P_k}\right)\\ &= \sum_k P_k \log\left(\frac{1}{Q_k}\right) - H_P\end{aligned}$$

Defining as Q the probability induced by a π_k-Code, the distance is:

$$D(P,Q) = E(\pi_k) - H_P$$

The entropy H_α of the zeta distribution of probability $1/(x^\alpha \zeta(\alpha))$ is:

$$\begin{aligned} H_\alpha &= -\sum_{x=1}^{\infty} \log\left(\frac{1}{\zeta(\alpha)x^\alpha}\right)\frac{1}{\zeta(\alpha)x^\alpha} \\ &= \log\zeta(\alpha) - \frac{\alpha}{\ln 2}\frac{\zeta'(\alpha)}{\zeta(\alpha)} \end{aligned}$$

In order to make easier the following calculations, we recall Equation 4.3:

$$E(\pi_k) = 1 + k + \frac{1}{\zeta(\alpha)}\left[(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k j}}^{2^{2^k(j+1)}-1}\frac{j}{x^\alpha} + \sum_{i=1}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{i}{x^\alpha}\right]$$

The first element of the entropy ($\log\zeta(\alpha)$) is approximated, for $1 < \alpha \leq 2$, with $1+k$; in fact, always exists an integer $k \geq 0$ such that $0 \leq k+1-\log\zeta(\alpha) < 1$. We can set k to $\left\lceil \log\frac{\zeta(\alpha)}{2} \right\rceil$.

The second element of H_α is approximated with (the second part of Equation 4.3):

$$\frac{1}{\zeta(\alpha)}\left[(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k j}}^{2^{2^k(j+1)}-1}\frac{j}{x^\alpha} + \sum_{i=0}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{i}{x^\alpha}\right]$$

in fact:

$$\begin{aligned}
&\frac{1}{\zeta(\alpha)}\left[(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k}j}^{2^{2^k}(j+1)-1}\frac{j}{x^\alpha}+\sum_{i=0}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{i}{x^\alpha}\right]\\
&<\frac{1}{\zeta(\alpha)}\left[(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k}j}^{2^{2^k}(j+1)-1}\frac{\log x}{2^k}\frac{1}{x^\alpha}+\sum_{i=0}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{\log x-2^k j}{x^\alpha}\right]\\
&=\frac{1}{\zeta(\alpha)}\left[\left(1+\frac{1}{2^k}\right)\sum_{x=1}^{\infty}\frac{\log x}{x^\alpha}+\sum_{x=1}^{\infty}\frac{\log x}{x^\alpha}-2^k\sum_{j=0}^{\infty}\sum_{x=2^{2^k}j}^{2^{2^k}(j+1)-1}\frac{j}{x^\alpha}\right]\\
&=\frac{1}{\zeta(\alpha)}\left[\left(1+\frac{1}{2^k}\right)\sum_{x=1}^{\infty}\frac{\log x}{x^\alpha}+\sum_{x=1}^{\infty}\frac{\log x}{x^\alpha}-\sum_{x=1}^{\infty}\frac{\log(x)}{x^\alpha}\right]\\
&=\frac{1}{\ln(2)\zeta(\alpha)}\left[\left(1+\frac{1}{2^k}\right)\sum_{x=1}^{\infty}\frac{\ln x}{x^\alpha}\right]\\
&=\frac{-1}{\ln(2)\zeta(\alpha)}\left[\left(1+\frac{1}{2^k}\right)\zeta'(\alpha)\right]
\end{aligned}$$

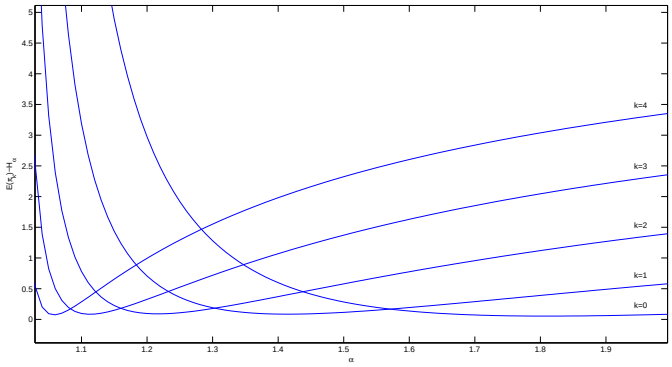
As said before, setting k to $\lceil\log\frac{\zeta(\alpha)}{2}\rceil$ (we throw the ceil away):

$$\frac{-\zeta'(\alpha)}{\ln(2)\zeta(\alpha)}\left[\frac{1}{2^k}+1\right]=\frac{-\zeta'(\alpha)}{\ln(2)\zeta(\alpha)}\left[\frac{2}{\zeta(\alpha)}+1\right]<\frac{-\alpha}{\ln(2)}\frac{\zeta'(\alpha)}{\zeta(\alpha)}+\frac{1}{\ln(2)}$$

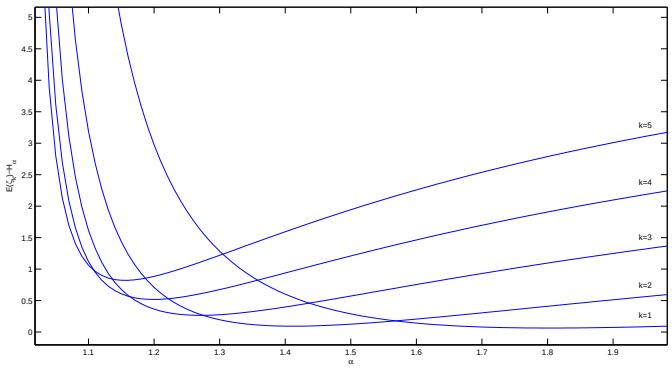
and the distance between the zeta distribution and a π_k-Code is:

$$\begin{aligned}
E(\pi_k)-H_\alpha =&1+k+\frac{1}{\zeta(\alpha)}\left[(1+2^k)\sum_{j=0}^{\infty}\sum_{x=2^{2^k}j}^{2^{2^k}(j+1)-1}\frac{j}{x^\alpha}+\sum_{i=1}^{2^k-1}\sum_{j=0}^{\infty}\sum_{x=2^h}^{2^{h+1}-1}\frac{i}{x^\alpha}\right]\\
&-\log\zeta(\alpha)+\frac{\alpha}{\ln 2}\frac{\zeta'(\alpha)}{\zeta(\alpha)}\leq 1+\frac{1}{\ln(2)}
\end{aligned}$$

Figure 4.1.(a) shows the redundancy of π_k-Code ($0\leq k\leq 4$) when $1<\alpha<2$. As expected, there always exists a $k\geq 0$ for which the expected length of π_k is almost equal to the entropy H_α. In the bottom part of Figure 4.1 it is shown the redundancy of ζ-Codes. The figure depicts the behavior of these codes to increase the distance from the entropy of the zeta distribution when α is close to 1.



(a)



(b)

Figure 4.1: The difference between the expected length and H_α of π_k-Code $(0 \leq k \leq 4)$ (a) and of ζ_k-Code $(1 \leq k \leq 5)$ (b).

Properties of π-Codes

The expected length of the π_k-encoding is:

$$\begin{aligned} E_P(\pi_k) &\leq E_P\left(1 + k + \left(1 + \frac{1}{2^k}\right)\log(n)\right) \\ &\leq 1 + k + \left(1 + \frac{1}{2^k}\right) E_P(\log(n)) \\ &\approx 1 + k + \left(1 + \frac{1}{2^k}\right) H_P \end{aligned}$$

It follows that each π-Code is a universal code since $(0 < H_P < \infty)$:

$$\frac{E_P(\pi_k)}{\max\{1, H_P\}} \leq 2 + k + \frac{1}{2^k}$$

and, for $k \to \infty$, we have an asymptotically optimal code:

$$\lim_{H_P \to \infty} \frac{E_P(\pi_k)}{\max\{1, H_P\}} = 1 + \frac{1}{2^k}\bigg|_{k \to \infty} = 1$$

4.4 Codes Comparison

In Table 4.3 is shown the comparison between ζ- and π- codes. The estimated expected length is computed using equations 4.2 and 4.4; since the estimate given for π-Code (Equation 4.4) has an error lower than the estimate for ζ-Code (Equation 4.2) and code π_1 is equivalent to ζ_2, we use Equation 4.4 to compute the expected length of ζ_2-Code. The results show that the expected length of π-Codes is closer to the entropy than the expected length of ζ-Codes; this behavior is more evident when the value of α approaches 1.

α	H_α	ζ_k			π_k		
		$E(\zeta_k)$	k	$D(\alpha, \zeta_k)$	$E(\pi_k)$	k	$D(\alpha, \pi_k)$
1.05	33.80	36.15	7	6.95%	33.90	4	0.27%
1.10	18.37	19.45	5	5.77%	18.48	3	0.52%
1.15	13.01	13.61	4	4.66%	13.16	3	1.14%
1.20	10.20	10.57	3	3.56%	10.30	2	0.94%
1.25	8.45	8.72	3	3.21%	8.56	2	1.28%
1.30	7.24	7.43	2	2.62%	7.42	2	2.44%
1.35	7.35	6.46	2	1.77%	6.46	1	1.77%

Table 4.3: The expected length of ζ- and π- codes. The distance $D(\alpha, \pi_k)$ ($D(\alpha, \zeta_k)$) is the distance in percentage of the expected length of π-Code (ζ-Code) from the entropy H_α of the zeta distribution.

4.5 Conclusions and Open Problems

In this chapter a new family of prefix codes was introduced, the π-Codes. This family performs well under a distribution of the integers that follows a power

law distribution, in particular a zeta distribution. The focus on this distribution is motivated by some properties of real graphs.

We have shown that π-Codes are universal codes and asymptotically optimal for $k \to \infty$. The characteristics of the π-Codes outperform the ζ-Code [BV05], and the δ-Code [Eli75] for a distribution of the integers that follows a zeta distribution. However, this problem is still open:

Open Problem 4.1 *Are the π-Codes suitable for different kind of distributions?*

Chapter 5

Prefix Codes for Arbitrary Distributions

Many prefix codes, presented in the literature, are proposed for the compression of sequences of integers that follow a particular probability distribution (for example the π-Codes introduced in Chapter 4). Other times, authors give the construction of the code and then they show that the proposed code could fit well some probability distributions of the integers (as the Yokoo code [Yok88]).

In this chapter the problem of creating a prefix code for a stream of integers of unknown probability distribution is investigated. To achieve this goal we use only those properties, of the unknown distribution, that can be discovered while reading the stream of integers. We also introduce a general prefix code, whose parameters can be fitted to obtain codes suitable for different probability distributions.

5.1 Background and Related Works

A famous technique to create a prefix code for a given distribution was presented in [Pig01]. In this paper Pigeon introduced the *Start/Stop* code as an extension of the *Start/Step/Stop* code proposed by Fiala and Greene in [FG89]. The Start/Stop code is constructed starting from a finite sequence of M non-negative integers m_i as follows: an integer n that belongs to the jth interval $[2^b, 2^{b+m_j})$ (of 2^{m_j} elements), where $b = \sum_{i=0}^{j-1} m_i$, is encoded with the unary encode of $j+1$ followed by the binary representation of the position of

n in its interval. The main advantage of the Start/Stop code with respect to the Start/Step/Stop code is that it is possible to select the parameters of the Start/Stop code in a way that the resulting average length code is shorter than the average length of a Start/Step/Stop code. Pigeon also proposed a method to properly set up the parameters of the Start/Stop code, from the probability distribution of the integers.

In [Gol95], Golin introduced the Simple Variable-Length Code that have an average length typically within 1% of the Huffman code. Golin provided two different algorithms to build from a given probability distribution a simple variable-length code and an optimal code. The first algorithm has time complexity $O(K)$ and the second $O(K \log(K))$, where K is the dimension of the alphabet. A simple variable-length code is equivalent to a Start/Stop code; in fact, it is composed by a prefix of variable length, that indicates the interval to which the integer belongs to, and a suffix of fixed length, that is, in the Start/Stop code the parameter m.

In this chapter, we focus on two families of probability distributions: geometric and power laws. We choose this two probabilities since the first is the most studied distribution (it was used also by Golin to validate his results), and the second distribution, as said in the chapter before, is a common distribution to describe real data.

n	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$
0	0	00	00	000	000
1	10	01	010	001	001
2	110	100	011	010	010
3	1110	101	100	011	0110
4	11110	1100	1010	1000	0111
5	111110	1101	1011	1001	1000

Table 5.1: The first codewords of the Golomb-Rice codes.

For the geometric distribution the most important codes are those by Golomb [Gol66] and Rice [Ric79], commonly referred to as Golomb-Rice. The Golomb-Rice code depends on the choice of a base $h \in \mathbb{N}^+$ and is computed as follows. The codeword of an integer $n \geq 0$ is composed of the unary representation of $i = \left\lfloor \frac{n}{h} \right\rfloor$ followed by the position of $n - ih$ in the interval $[0, h)$ using the truncated binary encoding. More in detail the Golomb code is a particular case of this code when h is a power of 2, so the code uses $\log(h)$ bits to encode the position in the interval instead of the truncated binary representation. In Table 5.1

are shown the first six codewords of the Golomb-Rice code for $1 \leq h \leq 5$.

The power law distributions of interest in this chapter are those for which the mean is finite. In Chapter 4, it is shown that the π-Code follows the entropy of a zeta distribution of infinite mean. When we deal with power law distributions of finite mean, the best known code is the K-Code [Bae08]. This code is based on one parameter $K \in \mathbb{Z}$. When $K = 0$ we have the base case of the code, and the definition of a codeword c_0 for an integer $n \geq 1$ is:

$$c_0(n) = \begin{cases} 0\ t(n-1,3) & n < 4 \\ 1\ c_0\left(\frac{n-2}{2}\right)\ 0 & n = 4 + 2k, \quad k \geq 0 \\ 1\ c_0\left(\frac{n-3}{2}\right)\ 1 & n = 5 + 2k, \quad k \geq 0 \end{cases}$$

where $t(n,k)$ denotes the truncated binary encoding of n in the interval $[0,k)$. When $K < 0$ the codewords of the first $|K|$ integers are the unary representation of n, and the codewords of an integer $n > |K|$ are composed of $|K|$ ones followed by the codeword $c_0(n-|K|)$. When $K > 0$ the codeword c_K of an integer n is:

$$c_K(n) = c_o\left(1 + \left\lfloor \frac{n-1}{2^K} \right\rfloor\right) t((n-1) \mod 2^K, 2^K)$$

For power law distribution of finite mean, only the K-Codes for $K < 0$ are important. In Table 5.2 are shown the first six codewords of K-Codes for $-3 \leq K \leq 1$.

n	$K=-3$	$K=-2$	$K=-1$	$K=0$	$K=1$
1	0	0	0	00	000
2	10	10	100	010	001
3	110	1100	1010	011	0100
4	11100	11010	1011	1000	0101
5	111010	11011	11000	1001	0110
6	111011	111000	11001	10100	0111

Table 5.2: The first codewords of K-Code.

n	$\Sigma = \{1,1,2,2\}$	$\Sigma = \{2,1,3\}$	$\Sigma = \{1,2,3\}$
1	1	10	1
2	01	11	010
3	0010	01	011
4	0011	0010	0010
5	00010	00110	00110
6	00011	00111	00111

Table 5.3: Examples of Σ-Codes. Only the first elements of the sequences are shown.

5.2 The Σ-Code

In the previous section we have presented two methods to build prefix codes from a given probability distribution. In this section we introduce a general code, called Σ-Code, and we study its properties with the aim of give a method to obtain a code that is suitable for a probability distributions of the integers that is unknown.

For any given infinite sequence $\Sigma = \{\sigma_k \in \mathbb{N}^+\}$, we define a Σ-Code for positive integers. First we divide the set $\mathbb{N}^+$ in subsets: the first subset with the first σ_0 integers, the second of σ_1 integers, and so on. The codeword associated with $n \in \mathbb{N}^+$, belonging to the kth subset, is composed of the unary description of $k+1$ followed by the truncated binary encoding of $n - n_k$ in the interval $[0; \sigma_k)$, where n_k is the smallest integer belonging to the kth subset. The truncated binary encoding of an integer i in the interval $[0; z)$ has length $\lceil \log(z) \rceil - 1$ if $i \in [0; 2^{\lceil \log(z) \rceil} - z)$ or $\lceil \log(z) \rceil$ otherwise. In Table 5.3 are presented an example of Σ-Codes.

To study some properties of a general Σ-Code, we define $\Sigma = \{\sigma_k = \max(1, \lceil ab^{f(k)} \rceil)\}$, where $a > 0$, $b \geq 1$ and $f(k) \geq 0 \ \forall \ k \geq 0$. Say I_k the interval of size σ_k equal to $(\sum_{i=0}^{k-1} \sigma_i; \sum_{i=0}^{k} \sigma_i]$. We call $L_k = \sum_{i=0}^{k-1} \sigma_i + 1$ and $U_k = \sum_{i=0}^{k} \sigma_i$ respectively the lower and upper bound of interval I_k. The codeword associated with $n \in I_k$ has length $l(n)$:

1. if $n - L_k \in [0; (2^{\epsilon_k^1} - 1)\sigma_k)$ then $l(n) = k + \lceil \log(\sigma_k) \rceil = k + \log(\sigma_k) + \epsilon_k^1 = k + f(k)\log(b) + \log(a) + \epsilon_k^2$;
2. otherwise $l(n) = k + 1 + \lceil \log(\sigma_k) \rceil$.

where $\epsilon_k^1 = \lceil \log(\sigma_k) \rceil - \log(\sigma_k)$ and $\epsilon_k^2 = \lceil \log(\sigma_k) \rceil - \log(ab^{f(k)})$.

We start showing that any Σ-Code achieves Kraft's inequality ($\Sigma_n 2^{-l(n)} \leq 1$) with equality. This is a necessary condition to build optimal infinite codes [LTZ97]. For Σ-Code we have:

$$\begin{aligned}\sum_{n=1}^{\infty} 2^{-l(n)} &= \sum_{k=0}^{\infty}\left(\frac{\sigma_k(2^{\epsilon_k^1}-1)}{2^{k+\log(\sigma_k)+\epsilon_k^1}} + \frac{\sigma_k(2-2^{\epsilon_k^1})}{2^{k+1+\log(\sigma_k)+\epsilon_k^1}}\right)\\ &= \sum_{k=0}^{\infty}\frac{2(2^{\epsilon_k^1}-1)+2-2^{\epsilon_k^1}}{2^{k+1+\epsilon_k^1}}\\ &= \sum_{k=0}^{\infty}\frac{1}{2^{k+1}} = 1\end{aligned}$$

To study the properties of a Σ-Code, in the following we use the implied distribution itself; in fact, we suppose to know only some features of a source (entropy, mean and variance), ignoring its distribution function. The implied distribution $P(\Sigma)$ of a Σ-Code is:

$$\begin{aligned}P(\Sigma) &= 2^{-(k+1+\lceil\log(\sigma_k)\rceil)}\\ &= \frac{1}{2^{k+1+\epsilon_k^1+\log(\sigma_k)}}\\ &= \frac{1}{ab^{f(k)}2^{k+1+\epsilon_k^2}}\end{aligned}$$

For each interval σ_k we use, as said, $k+\lceil\log(\sigma_k)\rceil$ bits to encode the first $\sigma'_k = (2^{\epsilon_k^1}-1)\sigma_k$ elements and $k+1+\lceil\log(\sigma_k)\rceil$ bits to encode the others. Under this condition the entropy of a Σ-Code is:

$$\begin{aligned}H(\Sigma) &= -\sum_{k=0}^{\infty}\sum_{n=L_k}^{L_k+\sigma'_k-1}\frac{1}{2^{k+\log(\sigma_k)+\epsilon_k^1}}\log\left(\frac{1}{2^{l(n)}}\right) +\\ &\quad \sum_{n=L_k+\sigma'_k}^{U_k}\frac{1}{2^{k+1+\log(\sigma_k)+\epsilon_k^1}}\log\left(\frac{1}{2^{l(n)}}\right)\\ &= \sum_{k=0}^{\infty}\frac{\sigma_k(2^{\epsilon_k^1}-1)}{2^{k+\log(\sigma_k)+\epsilon_k^1}}\log\left(2^{k+\log(ab^{f(k)})+\epsilon_k^2}\right) +\\ &\quad \frac{\sigma_k(2-2^{\epsilon_k^1})}{2^{k+1+\log(\sigma_k)+\epsilon_k^1}}\log\left(2^{k+1+\log(ab^{f(k)})+\epsilon_k^2}\right)\end{aligned}$$

$$
\begin{aligned}
&= \sum_{k=0}^{\infty} \frac{1}{2^{k+1}} \log\left(2^{k+\log(ab^{f(k)})+\epsilon_k^2}\right) + \frac{1-2^{\epsilon_k^1-1}}{2^{k+\epsilon_k^1}} \\
&= \sum_{k=0}^{\infty} \frac{2^{\epsilon_k^1}-1}{2^{k+\epsilon_k^1}} \log\left(2^{k+\log(ab^{f(k)})+\epsilon_k^2}\right) + \\
&\quad \frac{1-2^{\epsilon_k^1-1}}{2^{k+\epsilon_k^1}} \log\left(2^{k+\log(ab^{f(k)})+\epsilon_k^2}\right) + \frac{1-2^{\epsilon_k^1-1}}{2^{k+\epsilon_k^1}} \\
&= \sum_{k=0}^{\infty} \frac{k+\log(ab^{f(k)})+\epsilon_k^2}{2^{k+1}} + \frac{1-2^{\epsilon_k^1-1}}{2^{k+\epsilon_k^1}}\Bigg|_{\epsilon_k^1=\epsilon_k^2=\epsilon} \\
&= \sum_{k=0}^{\infty} \frac{k}{2^{k+1}} + \sum_{k=0}^{\infty} \frac{\log(a)+\epsilon}{2^{k+1}} \\
&\quad + \sum_{k=0}^{\infty} \frac{f(k)\log(b)}{2^{k+1}} + \sum_{k=0}^{\infty} \frac{1-2^{\epsilon-1}}{2^{k+\epsilon}} \\
&= \epsilon + 2^{1-\epsilon} + \log(a) + \frac{\log(b)}{2} \sum_{k=0}^{\infty} \frac{f(k)}{2^k} \\
&\leq 2 + \log(a) + \frac{\log(b)}{2} \sum_{k=0}^{\infty} \frac{f(k)}{2^k}
\end{aligned}
$$

We assume that $\epsilon_k^1 = \epsilon_k^2 = 0$ (it means that $ab^{f(k)}$ is always a power of 2). Under the same assumption, the mean of a Σ-Code, with respect to the implied distribution, is:

$$
\begin{aligned}
E(\Sigma) &= \sum_{n=1}^{\infty} n\ p(n) = \sum_{k=0}^{\infty} \sum_{n=L_k}^{U_k} \frac{n}{\sigma_k 2^{k+1}} \\
&= \sum_{k=0}^{\infty} \frac{L_k+U_k}{2^{k+2}} = \sum_{k=0}^{\infty} \frac{2\sum_{i=0}^{k-1}\sigma_i + 1 + \sigma_k}{2^{k+2}} \\
&= \sum_{k=1}^{\infty} \frac{\sum_{i=0}^{k-1}\sigma_i}{2^{k+1}} + \sum_{k=0}^{\infty} \frac{1}{2^{k+2}} + \sum_{k=0}^{\infty} \frac{\sigma_k}{2^{k+2}} \\
&\leq \frac{a}{4} \sum_{k=1}^{\infty} \frac{kb^{f(k-1)}}{2^{k-1}} + \frac{1}{2} + \sum_{k=0}^{\infty} \frac{ab^{f(k)}+1}{2^{k+2}}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{a}{4}\sum_{k=0}^{\infty}\frac{(k+1)b^{f(k)}}{2^k} + \frac{a}{4}\sum_{k=0}^{\infty}\frac{b^{f(k)}}{2^k} + 1 \\
&= \frac{a}{4}\sum_{k=0}^{\infty}\frac{b^{f(k)}(k+2)}{2^k} + 1
\end{aligned}
$$

And its variance:

$$
\begin{aligned}
V(\Sigma) &= \sum_{n=1}^{\infty} n^2 p(n) - E(\Sigma)^2 = \sum_{k=0}^{\infty}\sum_{n=L_k}^{U_k}\frac{n^2}{\sigma_k 2^{k+1}} - E(\Sigma)^2 \\
&= \sum_{k=0}^{\infty}\frac{1}{\sigma_k 2^{k+1}}\left(\frac{(U_k(U_k+1)(2U_k+1)}{6}\right. \\
&\quad \left. -\frac{L_k(L_k-1)(2L_k-1)}{6}\right) - E(\Sigma)^2 \\
&= \sum_{k=0}^{\infty}\frac{2(U_k^3-L_k^3)+3(U_k^2+L_k^2)+\sigma_k-1}{6\sigma_k 2^{k+1}} - E(\Sigma)^2 \\
&= \frac{1}{6}\left(1+\sum_{k=0}^{\infty}\frac{6k^2\sigma_k^2+6k\sigma_k^2+6k\sigma_k+2\sigma_k^2+3\sigma_k}{2^{k+1}}\right) - E(\Sigma)^2 \\
&= \frac{1}{6}+\frac{1}{2}\sum_{k=0}^{\infty}\frac{k^2\sigma_k^2+k\sigma_k^2+k\sigma_k}{2^k} + \frac{1}{6}\sum_{k=0}^{\infty}\frac{\sigma_k^2}{2^k} + \frac{1}{4}\sum_{k=0}^{\infty}\frac{\sigma_k}{2^k} - E(\Sigma)^2
\end{aligned}
$$

Since we can set, without loss of generality, $\sigma_k = \max(1, \lceil a2^{g(k)}\rceil)$, where $g(k) = f(k)\log(b)$, we set $b = 2$; the above properties become:

$$H(\Sigma) \leq 2 + \log(a) + \sum_{k=0}^{\infty}\frac{f(k)}{2^{k+1}} \tag{5.1}$$

$$E(\Sigma) \leq \frac{a}{4}\sum_{k=0}^{\infty}\frac{k+2}{2^{k-f(k)}} + 1 \tag{5.2}$$

$$V(\Sigma) \leq \frac{1}{6} + \frac{a^2}{6}\sum_{k=0}^{\infty}\frac{6k^2+6k+1}{2^{k+1-2f(k)}} + \frac{a}{4}\sum_{k=0}^{\infty}\frac{2k+1}{2^{k-f(k)}} - E(\Sigma)^2 \tag{5.3}$$

With reference to these properties it is easy to see how fast must the intervals grow to obtain codes of finite entropy, mean or variance: to have a finite

mean we need that $f(k) < k$ for k greater than a fixed constant (under the assumption that $f'(k)$ is a monotonic function for sufficiently large k). Note also that for k greater than a constant we have $f(k) < k < n$ since $n > k\sigma_0$. Given a distribution P and a Σ-Code (of implied probability distribution $P(\Sigma)$), we refer to the differences $H_{P(\Sigma)}(\Sigma) - H_P$, $E_{P(\Sigma)}(\Sigma) - E_P$, $V_{P(\Sigma)}(\Sigma) - V_P$ respectively with d_H, d_E and d_V; under this assumption the Kullback-Leibler distance (defined in Section 4.3) is:

$$\begin{aligned} D(P,\Sigma) &= \sum_{n=1}^{\infty} \left(l(n) - \log\left(\frac{1}{p(n)}\right)\right) p(n) \\ &= d_H + \sum_{n=1}^{\infty} l(n) \left(p(n) - \frac{1}{2^{l(n)}} \right) \\ &= d_H + \sum_{k=0}^{\infty} \sum_{n=L_k}^{U_k} (k+1+\log(\sigma_k)) \left(p(n) - \frac{1}{2^{l(n)}} \right) \\ &< d_H + \left| \sum_{k=0}^{\infty} \sum_{n=L_k}^{U_k} (n + (n-E)^2 + c) \left(p(n) - \frac{1}{2^{l(n)}} \right) \right| \\ &\approx d_H + |d_E + d_V| \end{aligned} \tag{5.4}$$

If the variance of both P and the distribution induced by the Σ-Code are infinite, we consider their distance d_V equal to 0.

In the following we refer to a Σ-Code with intervals $\sigma_i = \max(1, \lceil a 2^{f(k)} \rceil)$ with the name $\Sigma(a, f(k))$ and with the name $\Sigma(f(k))$ when $a = 1$. As we show in the next chapter, with this compact notation we can describe many codes presented in the literature, given to all of them a uniform description.

5.3 Codes for Well-Known Distributions

In this section we show how to create a prefix code that is suitable for some well-known probability distributions. We consider only probability distributions with finite entropy and mean, and such that $p(n) \geq p(n+1)$ for $n \in \mathbb{N}^+$; the distributions under analysis (main properties in Table 5.4) are the geometric distribution and two power law distributions: the Yule-Simon and the Zeta.

To deal with these probabilities we study $\Sigma(a, f(k))$ for $f(k)$ equals to 0, $\log(k+1)$, $k/4$, $k/2$ and $2k/3$; In Table 5.5 we present entropy, mean and variance of the distributions induced by these codes (using equations 5.1, 5.2 and 5.3).

	Geometric	Yule-Simon ($\rho > 1$)	Zeta ($\alpha > 2$)
PDF	$(1-\theta)\theta^{n-1}$	$\rho B(n, \rho+1)$	$\frac{1}{\zeta(\alpha)n^{\alpha}}$
Entropy	$\log\left(\frac{\theta}{1-\theta}\right) - \frac{\log(\theta)}{1-\theta}$	-	$\log(\zeta(\alpha)) - \frac{\alpha}{\ln(2)}\frac{\zeta'(\alpha)}{\zeta(\alpha)}$
Mean	$\frac{1}{1-\theta}$	$\frac{\rho}{\rho-1}$	$\frac{\zeta(\alpha-1)}{\zeta(\alpha)}$
Variance	$\frac{\theta}{(1-\theta)^2}$	$\begin{cases}\frac{\rho^2}{(\rho-2)(\rho-1)^2}\\ \infty\end{cases}$	$\begin{cases}\frac{\zeta(\alpha-2)}{\zeta(\alpha)} - \left(\frac{\zeta(\alpha-1)}{\zeta(\alpha)}\right)^2\\ \infty\end{cases}$

Table 5.4: Properties of geometric and power law distributions. The variances of the Yule-Simon and Zeta distribution are infinite for, respectively, $\rho \leq 2$ and $\alpha \leq 3$.

For a distribution P we choose, among the codes in Table 5.5, the one that minimize the distance $D(P, \Sigma)$; to compute this distance we set the parameter a of each code to $\max\{a_H, a_E, a_V\}$, where a_H, a_E and a_V are respectively the values of a computed using the formula of entropy, mean and variance shown in Table 5.5. For codes with infinite variance we set a to $\max\{a_H, a_E\}$. The choice to select the maximum value between a_H, a_E, and a_V is guided by the inequalities 5.1, 5.2 and 5.3.

$f(k)$	Entropy	Mean	Variance
0	$\log(a) + 2.0$	$1.5a + 1$	$2.1a^2 - 1.5a - 0.8$
$\log(k+1)$	$\log(a) + 2.7$	$4.0a + 1$	$105a^2 - 4a + 2.2$
$k/4$	$\log(a) + 2.3$	$2.1a + 1$	$23.9a^2 - 1.8a - 0.8$
$k/2$	$\log(a) + 2.5$	$3.8a + 1$	∞
$2k/3$	$\log(a) + 2.7$	$7.1a + 1$	∞

Table 5.5: Properties of some $\Sigma(a, f(k))$ codes.

Geometric Distribution

The geometric distribution is one of the most studied probability distribution and the best known code for this distribution is the Golomb-Rice code that is an optimal code [GvV75, YQ06] for this kind of distribution. For the geometric distribution when $\theta \geq 0.23$ the selected code is $\Sigma(a, 0)$ that is the Golomb code. For $\theta < 0.23$ the selected code is not the Golomb but it achieves the same

performance. This is because the first four integers have a total probability greater than 99%.

In [GvV75], Gallager and van Voorhis proposed a selection rule for the best level of the Golomb code under a geometric distribution, they set the level to:

$$l_\theta = \left\lceil \log_\theta \left(\frac{1}{1+\theta} \right) \right\rceil$$

Using Σ-Code the level of the Golomb code is set to the value obtained from the formula of the variance:

$$l_\theta^\Sigma = \left\lceil \frac{1}{25} \left(9 + \sqrt{331 + \frac{300\theta}{(1-\theta)^2}} \right) \right\rceil$$

The distance between l_θ^Σ and l_θ is at most one.

Power-Law Distribution

As far as we know the best code, called K-Code, for power-law distribution was presented in [Bae08] by Baer. We compare our results with the numerical results presented in that article. In this case we consider two different probability functions: Yule-Simon and Zeta.

		Entropy	K-Code	$f(k)$	$\Sigma(a, f(k))$	
Yule-Simon	$\rho = 1.5$	2.1707	**2.2308**	$k/2$	2.2573	(2.2256)
	$\rho = 2$	1.7469	1.8485	$k/2$	**1.8453**	(1.8351)
	$\rho = 2.5$	1.4763	1.6267	$k/4$	**1.6231**	(1.6231)
	$\rho = 3$	1.2867	1.4882	$k/4$	**1.4858**	(1.4857)
Zeta	$\alpha = 2.5$	1.4653	**1.6580**	$k/2$	1.6583	(1.6583)
	$\alpha = 3$	0.9789	1.3367	$k/2$	**1.3348**	(1.3348)
	$\alpha = 3.5$	0.6791	**1.1863**	$k/4$	1.1865	(1.1859)
	$\alpha = 4$	0.4816	**1.1100**	$k/4$	**1.1100**	(1.1099)

Table 5.6: Compression results in bits per symbol. Between parentheses there are the results of Σ-Codes manually setting the best a value.

As shown in Table 5.6, Σ-Codes not always outperform the best K-Code however it achieves comparable results. In both the cases of geometric and power law distributions, manually setting the best a value of the selected Σ-Code leads to obtain better (or equal) results that the one in the literature.

From this observation, we can conclude that the code selected using this method performs better than the best known code, but the selection of the parameter a is not optimal.

5.4 Real World Codes

Since encoding and decoding a Σ-Code require many operations, the codes proposed in Table 5.5 are too slow to be used in real application. However, in in this section we create codes from those presented in Table 5.5 that can be used in practice.

$\Sigma(a, \lfloor k/m \rfloor)$-Code

The codes $\Sigma(k/2)$ and $\Sigma(k/4)$ perform well for power law distributions, so we create a more general code, the $\Sigma(a, \lfloor k/m \rfloor)$, where $a, m \in \mathbb{N}^+$. This code encodes a natural number n into a codeword (using truncated binary encoding) of length $l(n) = k + \lceil \log(a) \rceil + k'$ bits if $n - am(2^{k'} - 1) - 1 < 2^{k'}(2^{\lceil \log(a) \rceil} - a)$, otherwise $l(n) = k + 1 + \lceil \log(a) \rceil + k'$, where k' is:

$$k' = \left\lceil \log\left(\frac{n}{ma} + 1\right) - 1 \right\rceil$$

and k is:

$$k = m(k' - 1) + \left\lfloor \frac{n + ma - 1}{a2^{k'}} \right\rfloor$$

Σ-Codes, that have a value of a less than 1, have a slow start of the size of the intervals, that is the first intervals have only one element. To use this slow start feature, we do not use directly the code as defined before but the code $\Sigma(a, \lfloor k/m \rfloor - c)$. This code has the first $m(c - \log(a))$ intervals of size 1.

$\Sigma(a, \log(k+1))$-Code

Another code we have used before is the $\Sigma(a, \log(k+1))$. A natural number n is encoded into a codeword of length $l(n) = k + \lceil \log(a(k+1)) \rceil$ if $n - a\frac{k^2+k}{2} - 1 < 2^{\lceil \log(a(k+1)) \rceil} - a(k+1)$, otherwise $l(n) = k + 1 + \lceil \log(a(k+1)) \rceil$, where k is:

$$k = \left\lceil \frac{1}{2}\sqrt{1 + \frac{8n}{a}} - \frac{3}{2} \right\rceil$$

5.5 Experimental Results

In Table 5.7, the performances of the codes $\Sigma(1, \lfloor k/2 \rfloor - c)$, $\Sigma(1, \lfloor k/4 \rfloor - c)$ and $\Sigma(1, \log(k + 1 - c))$ compared to the best K-Code are shown. The expected length is computed for integers in the interval 1 to 10^6; for the probability distributions under investigation, an integer greater than 10^6 has a probability less than 10^{-15} to appear. The K-Code outperform Σ-Codes only for the zeta distribution with $\alpha = 2.5$, in all the other cases the $\Sigma(\log(k + 1 - c))$-Code performs better. From Table 5.7 results that the best code to deal with power law distributions is the $\Sigma(1, \log(k + 1 - c))$.

		K-Code	$\Sigma(\lfloor k/2 \rfloor - c)$		$\Sigma(\lfloor k/4 \rfloor - c)$		$\Sigma(\log(k + 1 - c))$	
Yule-Simon	$\rho = 1.5$	2.2308	**2.2258**	(0)	2.2971	(0)	2.2289	(1)
	$\rho = 2$	1.8485	1.8402	(1)	1.8524	(0)	**1.8390**	(2)
	$\rho = 2.5$	1.6267	1.6220	(2)	1.6227	(0)	**1.6213**	(3)
	$\rho = 3$	1.4882	1.4859	(3)	**1.4857**	(0)	**1.4857**	(4)
Zeta	$\alpha = 2.5$	**1.6580**	1.6582	(1)	1.6771	(0)	1.6593	(2)
	$\alpha = 3$	1.3367	1.3350	(2)	1.3361	(0)	**1.3348**	(3)
	$\alpha = 3.5$	1.1864	1.1858	(3)	1.1858	(1)	**1.1857**	(4)
	$\alpha = 4$	1.1100	**1.1099**	(4)	**1.1099**	(1)	**1.1099**	(4)

Table 5.7: Comparison between K-Code and Σ-Codes. The number between parentheses is the values of the parameter c.

It is interesting to note that the values shown in Table 5.7 (obtained using the version of the codes that can be used in practice) are close to the results presented in Table 5.6 (obtained using general Σ-Codes). This is an evidence that it is not necessary to study exactly a code to know how it performs, but it is sufficient to know its general properties.

5.6 Conclusions and Open Problems

In this chapter we introduced Σ-Code, a prefix code built from a sequence of integers greater than zero. After the presentation of its properties, we showed how to obtain a Σ-Code for a probability distribution P of the integers knowing only entropy, mean and variance of P; in fact, we suppose to not have any information on the probability function of P. We have shown that the Σ-Code, obtained for the distribution P, is as good as - or it outperforms - the best

code proposed in the literature for P, but we have not yet proved in general how good it is.

Since it is difficult to use a general Σ-Code in real applications, we have introduced two new codes directly from the general definition of Σ-Code. These codes perform as we expected from the analysis of the Σ-Code, so studying Σ-Codes for a probability distribution seems to give enough information to create a good prefix code for that distribution, avoiding the necessity of an ad-hoc analysis for each particular distribution.

In this chapter we considered probability distributions of finite entropy and mean, so the following problem is still open:

Open Problem 5.1 *Can we extend the method to probability distribution of infinite mean?*

We choose a Σ-Code among the others when it minimize the distance calculated with Formula 5.4. However, this formula gives an upper bound of the distance:

Open Problem 5.2 *What is the distance of a Σ-Code from a probability distribution?*

A last open problem regards the parameter a of Σ-Codes. The technique to select this parameter that we use does not always give the best value for a:

Open Problem 5.3 *How can we set the parameter a of a Σ-Code to best fit the entropy of a probability distribution?*

Chapter 6

Unification of Prefix Codes

In the previous chapters we have presented some prefix codes that have been proposed in the last decades. Any prefix code has its own construction method based, for example, on a recursive formulation (as K-Code [Bae08]) or on a transformation of its binary representation (as γ-Code [Eli75]). Moreover, each code has its own properties that can differ from those of other codes. In this chapter, we show that using the Σ-Code (defined in Section 5.2), it is possible to give an unified description to many codes presented in the literature.

6.1 Background

In this section we recall the codes that we can describe using a Σ-Code.

The Golomb-Rice code [Gol66, Ric79] (GR in the following) divides the set of the integers in groups of constant size h and the codeword of an integer n is given by the unary representation of $i = \lfloor \frac{n}{h} \rfloor$ followed by the truncated binary encoding of $n - ih$ in the interval $[0, k)$.

Teuhola [Teu78] proposed the exponential Golomb code. In this code the number of codewords of a fixed length grows exponentially with the length of the codeword itself. Fixed a parameter h, an integer n in the ith interval $[2^h(2^i - 1), 2^h(2^{i+1} - 1))$ is encoded with the unary code of i followed by the truncated binary encoding of the position of $n - 2^h(2^i - 1)$ in the interval $[0, 2^{h+i})$.

In [Eli75], Elias proposed three codes: γ, δ and ω. In the γ-Code the length in unary encoding of the binary representation $\beta(n)$ of an integer n is added

as prefix, and the suffix is $\beta(n)$ without the first bit that is always a 1 (for $n \geq 1$). In the δ-Code the prefix is formed of the γ encoding of the length of $\beta(n)$. The ω-Code is more complicated, it encodes 1 with a 0 and any other integer greater than 1 in the following way. The partial encoding of $n > 1$ is composed of $\beta(n)$ followed by 0. To this partial encoding it is prefixed the length of $\beta(n)$ minus 1 in binary. Say $l = |\beta(n)| - 1$, the operation of adding a prefix is repeated considering l instead of n until $l \geq 2$.

The K-Code introduced by Baer in [Bae08] is defined using a parameter $K \in \mathbb{Z}$. Each codeword $c_0(n)$ of the K-Code for $K = 0$ is defined as:

$$c_0(n) = \begin{cases} 0\ t(n-1,3) & n < 4 \\ 1\ c_0\left(\frac{n-2}{2}\right)\ 0 & n = 4 + 2k, \quad k \geq 0 \\ 1\ c_0\left(\frac{n-3}{2}\right)\ 1 & n = 5 + 2k, \quad k \geq 0 \end{cases}$$

where $t(n,k)$ denotes the truncated binary encoding of n in the interval $[0,k)$. When $K < 0$, the codewords of the first K integers are the unary representation of n, and the codeword of an integer $n > |K|$ is composed of $|K|$ ones followed by the codeword $c_0(n-|K|)$. When $K > 0$ the codeword c_K of an integer n is:

$$c_K(n) = c_o\left(1 + \left\lfloor \frac{n-1}{2^K} \right\rfloor\right) t((n-1) \mod 2^K, 2^K)$$

Yokoo [Yok88] proposed a modified γ-Code. The prefix of the code is the same of the γ-Code, that is the unary representation of the length of $\beta(n)$. The suffix part of a codeword is composed of a single bit $\beta_0(n)$ and a variable-length part $\beta'(n)$. For an integer such that $|\beta(n)| = k$, let:

$$\mu(k) = \frac{1}{3}(2^{k-1} - (-1)^{k-1})$$

The first $\mu(k)$ integers of binary representation of length k have the bit 0 as $\beta_0(n)$, the other $2^{k-1} - \mu(k)$ have $\beta_0(n) = 1$. For the integers with $\beta_0(n) = 1$, the $\beta'(n)$ part is the truncated binary encoding of $n - 2^{k-1} - \mu(k)$ in the interval $[0, 2^{k-1} - \mu(k))$. For the integers with $\beta_0(n) = 0$, the $\beta'(n)$ part is the same of $n - 2^{k-2} - \mu(k-1)$ (an integer in the previous interval).

The Start-Step-Stop code, introduced by Fiala and Greene in [FG89], is based on three parameters that give the name to the code: *start*, *step* and *stop*. An integer belonging to the kth interval is encoded with the unary description of k and its position in the interval using $start + k \cdot step$ bits. The kth interval is bounded by $\left[start + step\frac{k(k+1)}{2}, start + step\frac{(k+1)(k+2)}{2}\right)$. The *stop* parameter

is just an upper bound on the set of integers to be encoded; to encode the whole set of natural numbers *stop* must be set to ∞.

An extension of the Start-Step-Stop, called Start-Stop, was proposed by Pigeon [Pig01]. Instead of using $start + k \cdot step$ bits to encode an integer in the kth interval, the Start-Stop code uses m_k bits. This code is defined by the set of parameters m_k. So the first 2^{m_0} integers are encoded with 0 and m_0 bits, integers in the interval $[2^{m_0}, 2^{m_0} + 2^{m_1})$ are encoded with 10 and m_1 bits, and so on.

The last code of our interest is the Levenstein code [Lev68]. It encodes zero with a single 0. For an integer greater than zero, to obtain the encoding we need to perform the following steps:

1. initialize the partial encoding of n to the empty string and the counter *count* to 1;
2. add $\beta(n)$ without the leading 1 to the partial encoding as prefix;
3. if $|\beta(n)| - 1 > 0$, increment *count* by one and go to step 2 considering $|\beta(n) - 1|$ instead of n;
4. if $|\beta(n)| - 1 = 0$, prefix *count* in unary encoding to the partial encoding and stop.

The Levenstein code of an integer n greater than zero is always a bit longer than the Elias ω-Code.

In Table 6.1 are reported the first codewords of some of the codes described in this section.

6.2 Extended Truncated Binary Encoding

As shown in the previous section, many prefix codes encode integers in an interval with the same prefix and with a suffix that can depend from the integer in different way. The Σ-Code is suitable to divide integers in interval assigning to each block the same prefix. However, to unify the prefix codes under a common description, we need a more powerful and adaptable code to describe the suffix part.

To encode an integer in the interval $[0, N)$ usually it is used the truncated binary encoding. When N is a power of 2 it encodes each integer using $\log(N)$ bits however, to not waste space, when N is not a power of 2 it encodes an integer in the interval $[0, M)$ using $\lceil \log(N) \rceil - 1$ bits, where $M = 2^{\lceil \log(N) \rceil - 1}$,

n	GR_h	EG_h	δ	ω	Yokoo	Levenstein
0	1 00	1 0	-	-	-	1
1	1 01	1 1	1	0	1	01
2	1 10	01 00	010 0	10 0	01 0	001 0
3	1 11	01 01	010 1	11 0	01 1	001 1
4	01 00	01 10	011 00	10 100 0	001 0	0001 0 00
5	01 01	01 11	011 01	10 101 0	001 10	0001 0 01
6	01 10	001 000	011 10	10 110 0	001 110	0001 0 10
7	01 11	001 001	011 11	10 111 0	001 111	0001 0 11
8	001 000	001 010	00100 000	11 1000 0	0001 00	0001 1 000

Table 6.1: First codewords of some of the codes recalled in the background. The Golomb-Rice is shown for $h = 4$ and the Exponential-Golomb for $h = 1$.

and integers in the interval $[M, N)$ using $\lceil \log(N) \rceil$ bits. This encoding is the standard method to encode the suffix part of a Σ-Code.

In this section we extend the truncated binary encoding to allow different possibilities to encode integers in the same interval $[0, N)$ using at most $\lceil \log(N) \rceil + c$ bits (where $c \geq 0$ is a constant). Fixed a non negative integer ρ, let $c' = \lceil \log(N) \rceil + c - \rho$ the number of bits used to encode the first elements of the interval: we encode the first chunk c_0 of $2^{c'} - q_1$ elements using c' bits, the following chunks c_i of $2q_i - q_{i+1}$ $(1 \leq i < \rho)$ elements using $c' + i$ bits, and the last chunk c_ρ of $2q_\rho$ elements using $\lceil \log(N) \rceil + c$ bits.

The first codeword of this code is composed of c' zeros, the last is composed of $\lceil \log(N) \rceil + c$ ones. The first codeword of chunk c_i $(i > 0)$ is composed of the last codeword of c_{i-1} plus one followed by a zero, and any other codeword is equal to the previous codeword plus one. An example of extended truncated binary encoding is shown in Table 6.2.

As shown in Table 6.2 setting the values of ρ and c does not guarantee an unique code. To have this feature the rule to set the values of q_i must be known to the encoder.

The q_i cannot be fitted to arbitrary values, we have the following constraints:

$$\begin{cases} N = q_0 + q_1 + \ldots + q_\rho \\ q_0 = 2^{c'} \\ q_0 \geq q_1 \\ 2q_i \geq q_{i+1} \end{cases} \tag{6.1}$$

n	$\rho = 1$ $c = 0$	$\rho = 2$ $c = 1$	$\rho = 3$ $c = 1$	$\rho = 3$ $c = 1$	$\rho = 4$ $c = 2$
0	00	00	0	0	0
1	01	01	100	10	10
2	100	10	101	1100	110
3	101	110	110	1101	1110
4	110	1110	1110	1110	11110
5	111	1111	1111	1111	11111

Table 6.2: An example of truncated binary encoding using different parameters for integers in the interval $[0, 6)$.

Note that $\rho + \log(N) \geq \lceil \log(N) \rceil + c$ since $q_0 \leq N$. If each chunk must contain at least one element we simple substitute the $\geq$ signs with $>$ in the previous system of equations.

6.3 Common Description for Some Prefix Codes

Each prefix code proposed has its own definition that can differ from the other and the same code could have more possible definitions. In this section we introduce a convenient way to describe prefix codes that uses just few parameters: the parameters of a Σ-Code, and those of the extended truncated binary encoding.

Suppose that we want to use at most $\lceil \log(\sigma_k) \rceil + c$ to encode the σ_k elements of an interval of a $\Sigma(a, f(k))$ code. To avoid the codeword of n to be longer than the one of $n + 1$, we need that $\lceil \log(\sigma_{k+1}) \rceil + c - \rho + 1 \geq \lceil \log(\sigma_k) \rceil + c$ (the first codeword of interval $k + 1$ must be longer or equal to the last of the interval k), that is, $\rho \leq f(k + 1) - f(k) + 1$ (we throw the ceil away).

For example, for $\Sigma(a, k)$ one can choose ρ in $\{0, 1, 2\}$. When $\rho = 0$ we have the γ-Code for $a = 1$ and the Exponential-Golomb for $a = 2^h$. When $\rho = 1$ we have, for $a = 3 \cdot 2^K$, the K-Code.

Using one bit more than $\lceil \log(\sigma_k) \rceil$ ($c = 1$) we can set ρ to 2. In this case many systems of equations have more solutions, for example the system for $N = 8$, $c = 1$ and $\rho = 2$ has two solutions; to avoid this ambiguity one can impose q_ρ to be 1 obtaining the code proposed by Yokoo [Yok88] (for $a = 1$).

In Table 6.3 are shown how some well known codes can be expressed using Σ-Codes.

Code [Reference]	a	$f(k)$	c	ρ	q_i
Unary	1	0	0	0	
GR_h [Gol66, Ric79]	h	0	0	1	
γ-Code [Eli75]	1	k	0	0	
EG_h [Teu78]	1	$k+h$	0	0	
K-Code [Bae08]	3	$k+K$	0	1	
Yokoo [Yok88]	1	k	1	2	$q_2 = 1$
Start-Step-Stop [FG89]	2^{start}	$k \cdot step$	0	1	
Start-Stop [Pig01]	1	$\sum_{i=0}^{k} m_i$	0	0	
π_h-Code [Chapter 4]	$2^{2^h}-1$	$2^h k$	0	2^h-1	$\begin{cases} q_0 = 2^{(2^h k+h)} \\ q_i = 2^{(2^h k+i-1)}(2^h-i) \end{cases}$
δ-Code [Eli75]	1	$2^k - 1 + \log(2^{2^k}-1)$	$k-1$	2^k-1	$\begin{cases} q_0 = 2^{(2^k+k-1)} \\ q_i = 2^{(2^k-2+i)}(2^k-i) \end{cases}$

Table 6.3: Parameters to obtain well known codes.

The codewords obtained using Σ-Codes, shown in Table 6.3, can be different from those of the original code, but for each integer the length of the codeword is the same for both the codes.

Using Σ-Codes instead of particular definitions leads to two main advantage:

1. using this notation it is easy to note similarities between codes and their properties; and
2. the codewords of a Σ-Code are lexicographically ordered.

6.4 Unified Decoder

Another advantage of a common description for prefix codes is the possibility to reuse the same software. Here we present a common decoder for Σ-Codes shown in Figure 6.1.

```
Algorithm - Σ-Decoder
 1:  k ←readZeros()
 2:  n ←readBits(⌈log(σ_k)⌉ + c − ρ)
 3:  if ρ > 0 then
 4:    compute q_0 and q_1
 5:    i ← 1
 6:    s ← 0
 7:    m ← 0
 8:    d ← q_0 − q_1
 9:    while n ≥ m + d
10:      n ← 2n+readBits(1)
11:      s ← s + d
12:      m ← 2(m + d)
13:      compute q_{i+1}
14:      d ← 2q_i − q_{i+1}
15:      i ← i + 1
16:    end while
17:    n ← n − m + s
18:  end if
19:  n ← n + L_k
```

Figure 6.1: A decoder for Σ-Codes. It can be used to decode many well-know codes such those in Table 6.3.

We suppose to have two operations to read a sequence of bits:

- *readZeros()* - reads a sequence of 0s and return the number of bits read. The next element to be read will be the bit at the right of the first 1 found; and
- *readBits(l)* - reads l consecutive bits.

The computation of the parameters q_i of the code (line 4 and 13) is the most complex operation in the decoder. It can be accomplished solving the system of equations 6.1 (although this does not guarantee a unique solution), or using some specific methods for particular codes. For example, it is easy to solve the systems for the codes in Table 6.3 (as the Yokoo code) since a simple equation for each q_i is given.

6.5 Conclusions and Open Problems

In this chapter we have dealt with the problem of giving a common description to the prefix codes.

For this problem we have introduced an extension of the truncated binary encoding, that allows to include under the same description more prefix codes. Using this extension the description of a prefix code is composed of the parameters of a Σ-Code and the parameters of the extended truncated binary encoding. Unifying the prefix codes permits to note the similarity among different codes in an easy way. Moreover, codes defined with the proposed model can use the same decoder.

We have shown that we can use Σ-Codes with extended binary truncated encoding to describe many other codes. However, the following problem is still open:

Open Problem 6.1 *Can we describe any prefix code using a Σ-Code?*

Part III

String Mining

Chapter 7

Classification and Pattern Recognition

String mining is concerned with finding statistically relevant patterns in a set of strings. Finding patterns in strings is a challenging problem in many domains where data are represented as sequences. In medical domain, a set of characteristics is associated to each individual with a certain disease. In biological domain, we know the DNA sequence of a huge number of species or the gene expression level for many patients having the same disease.

In most cases finding patterns is motivated by the necessity to extract knowledge from these sequences; the pattern itself results to be interesting to characterize the sequence. Other times, the patterns are a mean to support classification tasks.

In this chapter, we examine some methods to detect patterns in strings and to classify them. Then we focus on a method specific to extract pattern in form of logic formulas and describe its application to relevant problems in DNA analysis.

7.1 Related Work

We now give a very brief, schematic description, of some methods to classify sets of strings that are used in biological contexts; some of them are based on the detection of patterns. We point out that in the following with the term "pattern" we indicate a subset of positions - not necessarily sequential - of the string.

Neighbor Joining

The Neighbor Joining algorithm [SN87] is a bottom-up clustering method used for the construction of phylogenetic trees based on sequence distance. Computational efficiency being its main advantage, Neighbor joining is the most widely used method for classifying DNA data. The underlying assumption is that the DNA sequences of distinct species form discrete clusters in the phylogenetic tree, because genetic variation within species is smaller than that between species.

This method iteratively joins the two elements of V that are at the minimum distance. The distance is computed with the Q-matrix, an $N \times N$ matrix (where N is the number of elements) that contains in position (u, v) the distance between element u and v defined as follows:

$$q_{uv} = d_{uv}(N-2) - \sum_{w \in V} d_{uw} - \sum_{w \in V} d_{vw}$$

where d_{uv} is the distance from u to v in a distance matrix D. The neighbor joining algorithm performs the following steps:

1. compute the Q-matrix of the set of elements V from the distance matrix D;

2. add a node w to the tree joining the two element $(u, v \in V)$ with the lowest value in the Q-matrix. Nodes u and v are removed from V and w is inserted;

3. calculate the new distances D of the nodes from the new node w. The new distances from w are defined as $d_{wz} = \frac{1}{2}(d_{uz} + d_{vz} - d_{uv})$; and

4. start the algorithm again with the new V set and D matrix.

Sequences of unknown specimens can subsequently be included in the tree to see in which cluster they appear.

Best Match

Best match is considered to be the least stringent method for DNA sequence classification. It simply assigns a query sequence to the same species membership as its closest sequence in a reference library, based on distances between sequences [MSV06].

Decision Tree

A decision tree (sometimes also called classification tree) is a binary tree whose leaves are labeled by the classes, and whose internal nodes are associated to binary predicates related to the features defining the objects. At each node, one of the outgoing branches corresponds to objects whose features satisfy the predicate, while the other corresponds to objects which do not satisfy the predicate. The path from the root to each node corresponds therefore to a set of conditions which must be met by all objects associated with the node.

Decision trees are built (learned) recursively from the training data via some standard procedures. One such rule (entropy rule) is to find at each tree node a predicate which optimizes an entropy function, or information gain, of the partition it induces (intuitively, a predicate which splits a group in roughly two halves has a good information content). Popular tree decision software packages, widely used in bioinformatics applications, such as C4.5 [Qui93, Qui96], are based on entropy rules.

To predict the class label of a new input object, the predicates are applied to the input starting at the tree root, one at a time. The responses to the predicates define a path from the root to one leaf. The object is then classified with the class labeling the leaf.

Logic Formulas for Classification

The learning of propositional formulas able to execute a correct classification task is performed by several methods presented in the literature. For instance, the already discussed decision trees may be viewed as propositional formulas whose clauses are organized in a hierarchy, and indeed one of their first implementation was designed to deal with data express in binary or qualitative form. A more recent alternative is the more sophisticate Logical Analysis of Data, originally proposed by in [BIK$^+$96]. In this category belongs also Lsquare, described in [FT01, FT06, Tru04]. The basic idea of this method is that the rules are determined using a particular problem formulation that turns out to be a well know and hard combinatorial optimization problem, the minimum cost satisfiability problem, or MINSAT. The disjunctive normal form formulas identified have the property of being created by conjunctive clauses that are searched for following the order with which they cover the training set. Therefore, they are formed by few clauses with large coverage (the interpretation of the trends present in the data) and several clauses with smaller coverage (the interpretation of the outliers).

The original method proposed in [FT02] and then extended in [BFFL08] has been implemented with some improvements and extensions that lead to the system called DMB, Data Mining in Big. The system is available at `http://dmb.iasi.cnr.it/`, in an online version. It can deal both with strings of characters and with sequences of real numbers. In the second case a binarization step is performed, since strings of characters are needed to perform the mining process.

7.2 Testing DMB on Biological Problems

In this section we describe some experimental results obtained by applying the logic miner DMB to some problems arising in biological context.

Barcode Analysis

This method was applied to mitochondrial DNA sequences, called barcode, in [BFW09] to classify species. The generated formulas, that are able to distinguish among the different species, recognize patterns in the DNA sequences that are specific for each species. An example of logic formulas for class C could be `Position 1=A AND Position 4=C`. This formula leads to a pattern that characterizes all the string belonging to C; if we have 5 features the pattern, with respect to the example formula, would be `AxxCx`, where `x` is a wildcard that means any other character.

In this section, we report some interesting experimental result derived from the comparison of this method with other methods that classify species through barcode. Here we briefly report on some ongoing research activity in this field conducted in collaboration with the Biosystematics Group of the Wageningen University[1].

The experiments compare different data mining methods (Neighbor Joining, Best Match, and Logic Data Mining) applied to DNA strings and show that effective classification can be obtained by suitable string analysis algorithms designed with the purpose of distinguishing strings of different classes based on the substrings they contain.

For the test we used simulated DNA barcode datasets, that are more difficult to be classified and so, they are more appropriate to show the real performances of the methods.

[1]We thank in particularly Robin van Velzen for proving some of the results in Table 7.1 and the for the production of the simulated data described in the following section

Data Simulation

Realistic DNA barcode datasets were simulated using Mesquite (version 2.72 build 528) [MM09]. The simulation took place along two axes: time of species divergence and effective population size. Ultrametric gene trees T, with 20 samples per species, were simulated on a random ultrametric species tree for 50 species. Gene trees were simulated using effective population sizes of 1, 10 and 50 thousands individuals. The nucleotide frequencies are of 0.30 for A, 0.15 for C, 0.10 for G, 0.45 for T. Sequence length was 650 base pairs, approximating the length of the standard DNA barcode for animals. Simulated datasets were converted to fasta and then were divided over a train set with 15 samples per species and a test set with 5 samples per species. The train sets were considered as DNA barcode reference libraries containing samples of a priori known species membership, the samples in the test sets were considered unknown query DNA barcodes.

Methods Comparison

As logic data miner we use DMB (Data Mining in Big). The neighbor joining and best match method are implemented in R, using the functions of the package APE (version 2.53) [PCS04].

For the neighbor joining, to calculate descriptive power we tested if conspecific sequences appear as distinct (monophyletic) clusters in trees based on the train sets. To calculate predictive power we tested if the two clusters nearest to each query sequence in the test sets consist exclusively of the same species membership.

For the best match method, to test descriptive power, we tested if closest matches of all sequences in the train sets are conspecific. To test predictive power we tested if query sequences in the test sets have a closest match with a train sequence of the same species membership.

The comparative study shows that success scores generally decrease with increasing effective population size. The datasets that were simulated according to the lowest population size (1,000) obtained an average success score of 91%, instead datasets simulated according to the highest population size (50,000) were the most challenging in terms of species identification since they obtained only a 78% of average success score.

Table 7.1 shows that neighbor joining obtained the lowest scores both in train and test sets. Best Match obtained good results on the test while DMB

performs better on the train. Taking both (train and test) measures together, DMB (90%) outperformed best match (88%) by two percent.

Population	Train			Test		
Size	NJ	DMB	BM	NJ	DMB	BM
1000	89.34	91.90	92.50	89.94	90.36	92.52
10000	81.56	89.54	88.44	88.60	91.43	92.27
50000	42.84	87.00	73.26	83.10	89.05	91.27

Table 7.1: Success scores in percentage of the compared methods.

Classification of Polyoma Viruses

The DMB system was also tested to discover patterns and to classify sequences of DNA taken from different families of well known viruses called polyoma[2].

The dataset is composed of the five families of polyoma viruses: KIV, MCV, WUV, BKV, and JCV. For each individual of a family, five DNA sequences are given, each corresponding to one gene sequence: LT, ST, VP1, VP2, and VP3. Each DNA sequence has a number of nucleotides in the range between 80 and 100, and each family has from 60 to 1200 individuals.

In order to find patterns, we have applied DMB to the sequences of each gene separately. For each gene we have considered all the families of virus and found positions in the DNA that distinguish each family from all the others. The method detects that for each family and each gene, one position in the sequence of DNA is enough to correctly classify each individual.

These positions are now analyzed by experts to discover biological meaning and the role played by these nucleotides. Another experiment was performed over the entire sets of all the genes. This experiment was devoted to find a model capable to correctly classify a sequence among the genes. In this case the logic formulas are more complex than the previous, but it is highly probable that these patterns have no biological meaning. In both cases, the formulas cover the whole set, so that it is possible to distinguish exactly among the different viruses.

[2]The results of these experiments are reported in [BFW+10].

Gene Expression Analysis in Alzheimer Disease

The identification of early and stage-specific biomarkers for Alzheimer's Disease (AD) is critical, as the development of disease-modification therapies may depend on the discovery and validation of such markers. The identification of early reliable biomarkers depends on the development of new diagnostic algorithms to computationally exploit the information in large biological datasets.

We used the DMB system to identify potential biomarkers from mRNA expression profile data[3]. The analysis was performed on a large microarray gene expression dataset from the anti-NGF AD11 transgenic mouse model. In order to describe specific gene expression patterns characterizing the early and late stage of the neurodegenerative progression in the AD11 model, a microarray mRNA expression analysis was performed at different time points (1, 3, 6 and 15 months). In particular, we address the question as to whether a subset of differentially expressed genes in the brain of AD11 mice might be identified, to represent an expression signature of the neurodegeneration process, that might discriminate diseased from control mice, thus representing potential new biomarkers.

The DMB system was adopted to discover genes whose expression or co-expression strongly characterizes the AD11 models with respect to the corresponding controls. The method identifies combinations of genes whose expression level determines an effective separation between diseased and control mice. The whole datasets was divided into two groups of early (1-3 months) and late (6-15 months) stages samples. Each of these two groups was composed by 60 samples, including the mRNAs from all the analyzed brain regions (basal forebrain, hippocampus and cortex), with over 20 thousands normalized genes for each sample. Each of the two groups was analyzed by the DMB system, with the aim of identifying the genes that can discriminate between the early and late stages.

In order to reduce the number of features we apply a clustering method. The binarization step of DMB assigns each real value of a feature to an interval proportional to the entropy of the values of the feature itself. Once we have assigned all the values of the experiments to intervals, we join together all the features that have assigned the same interval to the values of the same experiment; these features create a cluster. The application of this discrete clustering method shrinks the whole gene set down to 3,656 for 1-3 months and to 3,615 for 6-15 months. After the clustering, the feature selection step was applied, requiring to select a small set of genes. We impose to the system

[3]The results of this section are a joint work to appear in [ADB+10].

	1-3 months
AD11	(Nudt19 $<$ 0.76) OR (Arl16 $\geq$ 1.31) OR (Aph1b $\geq$ 0.47) OR (Slc15a2 $\geq$ 0.55) OR (Agpat5 $\geq$ 0.73) OR (Sox2ot $<$ 0.58 OR Sox2ot $\geq$ 1.53) OR (2210015D19Rik $\geq$ 0.86) OR (Wdfy1 $\geq$ 1.37)
Control	(Nudt19 $\geq$ 0.76) OR (Arl16 $<$ 1.31) OR (Aph1b $<$ 0.47) OR (Slc15a2 $<$ 0.55) OR (Agpat5 $<$ 0.73) OR (0.58 $\geq$ Sox2ot AND Sox2ot $<$ 1.53) OR (2210015D19Rik $<$ 0.86) OR (Wdfy1 $<$ 1.37)

Table 7.2: Logic formulas to distinguish among Alzheimer diseased and healthy mice in the range 1-3 months.

to select at most a single gene only if it is able to discriminate between diseased and control samples. After a gene is chosen and the logic formulas are obtained, we remove the selected gene from the data, reiterating this procedure until no separation with only one gene is possible. The adoption of this simple iterative approach results in the identification of 8 genes in the 1-3 months and of 12 genes in the 6-15 months dataset, each gene is able to separate the AD11 from the control mice. It is interesting to point out that these genes, hereafter called "core genes", are singletons; that is, they do not belong to any of the previously identified clusters.

For each mouse/age group a boolean formula was found shown in Table 7.2 and Table 7.3. Each clause of a formula refers to only one gene and has the same capacity as the others to separate AD11 and control samples: each one of these core genes can be considered a marker on its own, with the same status. It is important to underline that the numbers in the formulas are not fold change ratios, but just Lowess normalized two-color expression values.

These single-gene formulas correspond to two small sets of 8 and 12 core genes Table 7.4 out of the sample groups, called $[core]_{1-3m}$ and $[core]_{6-15m}$,

6-15 months	
AD11	(Slc15a2 $\geq$ 0.62) OR (Agpat5 $<$ 0.26 OR Agpat5 $\geq$ 0.55) OR (Sox2ot $\geq$ 1.78) OR (2210015D19Rik $\geq$ 0.82) OR (Wdfy1 $<$ 0.75 OR Wdfy1 $\geq$ 1.29) OR (D14Ertd449e $<$ 0.33 OR D14Ertd449e $\geq$ 0.52) OR (Tia1 $<$ 0.17 OR Tia1 $\geq$ 0.49) OR (Txnl4 $<$ 0.74) OR (1810014B01Rik $<$ 0.71 OR 1810014B01Rik $\geq$ 1.17) OR (Snhg3 $<$ 0.16 OR Snhg3 $\geq$ 0.35) OR [(1.12 $\geq$ Actl6a AND Actl6a $<$ 1.42) OR Actl6a $\geq$ 1.48] OR (Rnf25 $<$ 0.67 OR Rnf25 $\geq$ 1.26)
Control	(Slc15a2 $<$ 0.62) OR (0.26 $\geq$ Agpat5 AND Agpat5 $<$ 0.55) OR (Sox2ot $<$ 1.78) OR (2210015D19Rik $<$ 0.82) OR (0.75 $\geq$ Wdfy1 AND Wdfy1 $<$ 1.29) OR (0.33 $\geq$ D14Ertd449e AND D14Ertd449e $<$ 0.52) OR (0.17 $\geq$ Tia1 AND Tia1 $<$ 0.49) OR (Txnl4 $\geq$ 0.74) OR (0.71 $\geq$ 1810014B01Rik AND 1810014B01Rik $<$ 1.17) OR (0.16 $\geq$ Snhg3 AND Snhg3 $<$ 0.35) OR [(0.81 $<$ Actl6a AND Actl6a $<$ 1.12) OR (1.42 $<$ Actl6a AND Actl6a $<$ 1.48)] OR (0.67 $\geq$ Rnf25 AND Rnf25 $<$ 1.26)

Table 7.3: Logic formulas to distinguish among Alzheimer diseased and healthy mice in the range 6-15 months.

respectively. Five genes are in common between the two groups, though inserted in slightly different formulas. We tentatively identify these core genes as potential biomarkers for the neurodegenerative disease in the AD11 model,

	Average Fold change ratio AD11 / Control											
	HP		CTX		BF		HP		CTX		BF	
Gene	1m	3m	1m	3m	1m	3m	6m	15m	6m	15m	6m	15m
Nudt19	0.71	0.71	0.72	0.67	0.74	0.66						
Arl16	1.80	1.93	1.93	1.52	2.13	1.78						
Aph1b	2.47	2.09	2.11	1.95	2.29	2.11						
Slc15a2	3.84	3.55	4.11	3.39	5.05	3.45	3.44	3.67	3.30	3.28	4.24	2.82
Agpat5	3.05	1.75	2.90	1.67	3.40	1.84	2.48	2.32	2.68	1.55	2.59	1.48
Sox2ot	3.41	2.59	3.08	3.41	3.03	3.21	3.24	2.79	2.89	2.73	2.47	2.63
2210015D19Rik	1.62	1.81	1.77	1.76	1.88	1.71	1.69	1.79	1.73	1.78	1.54	1.62
Wdfy1	2.21	1.86	2.08	2.19	2.02	1.95	1.93	2.02	1.97	1.91	1.93	1.90
D14Ertd449e							1.58	1.51	1.61	1.60	1.87	1.40
Tia1							3.34	4.47	3.53	2.58	4.74	2.95
Txnl4							0.65	0.66	0.64	0.67	0.59	0.57
1810014B01Rik							1.99	1.78	1.70	1.73	1.82	1.68
Snhg3							2.63	2.60	2.50	2.43	2.41	1.99
Actl6a							1.92	1.82	2.10	1.42	1.94	1.43
Rnf25							2.47	2.63	2.32	1.77	2.60	2.06

Table 7.4: Genes that discriminate AD11 and control samples in experimental data. Average fold change, in the linear scale, of core genes that discriminate AD11 and control samples. Each gene corresponds to a formula in (a) extracted by Logic Mining. Average fold change ratios AD11/Control are shown in the three analyzed tissues at 1, 3, 6, 15 months of age. The direction of fold change is constant: Nudt19 and Txnl4 genes are always down-regulated while the others are always up-regulated.

which are able to discriminate the diseased model from the control. Three genes specific of the early stage of $[core]_{1-3m}$ set, 5 genes common to both groups and 7 genes specific of the late $[core]_{6-15m}$ set were identified (Table 7.4), able to predict the correct assignment of any sample either to the AD11 or control group with 100% probability.

7.3 Conclusion

In this chapter we have presented a sample of methods for the classification of strings. Some of these methods give logic formulas that can be interpreted as peculiar patterns of the elements under analysis.

In particular, we have tested the DMB system. This system is a logic data miner that gives logic formulas of classification. First we have shown that DMB performs better than neighbor joining and best match on synthetic datasets of DNA sequences. Then we have applied DMB to real biological datasets: polyoma viruses and microarray of Alzheimer diseased mice. For the first dataset we found logic formulas composed of only one element; this means that the polyoma viruses are strongly characterized by single nucleotides in their sequences. For the microarray analysis we introduced a clustering method to reduce the huge number of features (genes). With this reduction, we were able to find some genes that are correlated with the Alzheimer disease; some of these genes are known in literature to be related to this disease, while the others are now under examination.

The DMB system performs well on many different datasets. It gives logic formulas that are suitable for classification. However, two correlated problems are still open:

Open Problem 7.1 *Given training and test data, how many logic formulas with the same error rate on test can be extracted from the training data?*

and

Open Problem 7.2 *It is possible to efficiently extract all the logic formulas defined in Open Problem 7.1?*

Bibliography

Bibliography

[AD09] Alberto Apostolico and Guido Drovandi. Graph compression by bfs. *Algorithms*, 2(3):1031–1044, 2009.

[ADB+10] I. Arisi, M. D'Onofrio, R. Brandi, A. Felsani, S. Capsoni, G. Drovandi, G. Felici, E. Weitschek, P. Bertolazzi, and A. Cattaneo. Gene expression biomarkers in the brain of a mouse model for alzheimer's disease: mining of microarray data by logic classification and feature selection. *Journal of Alzheimer's Disease*, 24(4), 2010. In press.

[ADH+08] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24:i241–i249, July 2008.

[AG10] Noga Alon and Shai Gutner. Balanced families of perfect hash functions and their applications. *ACM Trans. Algorithms*, 6:54:1–54:12, July 2010.

[AJB99] Réka Albert, Hawoong Jeong, and Albert-László Barabási. The diameter of the world wide web. *CoRR*, cond-mat/9907038, 1999.

[AM01] Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *In Proc. of the IEEE Data Compression Conference (DCC)*, pages 203–212, 2001.

[AM10] Vo Ngoc Anh and Alistair Moffat. Local modeling for webgraph compression. *Data Compression Conference*, 0:519, 2010.

[AMN08] Yasuhito Asano, Yuya Miyawaki, and Takao Nishizeki. Efficient compression of web graphs. In *COCOON '08: Proceedings of the*

14th annual international conference on Computing and Combinatorics, pages 1–11, Berlin, Heidelberg, 2008. Springer-Verlag.

[AR02] V. Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, ISAAC '02, pages 453–464, London, UK, 2002. Springer-Verlag.

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4), 1995.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[BA99] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[Bae08] M.B. Baer. Prefix codes for power laws. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 2464–2468, July 2008.

[BC08] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 95–106, New York, NY, USA, 2008. ACM.

[BCSV04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[BFFL08] Paola Bertolazzi, Giovanni Felici, Paola Festa, and Giuseppe Lancia. Logic classification and feature selection for biomedical data. *Comput. Math. Appl.*, 55:889–899, March 2008.

[BFW09] Paola Bertolazzi, Giovanni Felici, and Emanuel Weitschek. Learning to classify species with barcodes. *BMC Bioinformatics*, 10(S-14):7, 2009.

[BFW+10] P. Bertolazzi, G. Felici, E. Weitschek, G. Drovandi, A. Lo Presti, M. Ciccozzi, and M. Ciotti. Human polyomaviruses genome analysis by logic mining techniques. Technical Report 10-23, IASI-CNR, December 2010.

[BHKK08] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The fast intersection transform with applications to counting paths. *CoRR*, abs/0809.2489, 2008.

[BIK+96] E. Boros, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik. An implementation of logical analysis of data. Technical Report 29-96, Rutgers University, NJ, 1996.

[BKM+00] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33(1):309–320, June 2000.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, pages 107–117, 1998.

[BV04] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[BV05] Paolo Boldi and Sebastiano Vigna. Codes for the world wide web. *Internet Mathematics*, 2(4), 2005.

[CKL+09] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228, New York, NY, USA, 2009. ACM.

[CN07] F. Claude and G. Navarro. A fast and compact Web graph representation. In *Proc. 14th International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS 4726, pages 105–116. Springer, 2007.

[DS+07] Banu Dost, Tomer Shlomi, Nitin Gupta 0002, Eytan Ruppin, Vineet Bafna, and Roded Sharan. Qnet: A tool for querying protein interaction networks. In *RECOMB*, pages 1–15, 2007.

[Eli74a] Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974.

[Eli74b] Peter Elias. Minimum times and memories needed to compute the values of a function. *J. Comput. Syst. Sci.*, 9(2):196–212, 1974.

[Eli75] Peter Elias. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on*, 21(2):194–203, 1975.

[Epp95] David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, SODA '95, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[ER60] P. Erdös and A. Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[Fan71] R. M. Fano. On the number of bits required to implement an associative memory. *Memorandum 61, Computer Structures Group, Project MAC*, 1971.

[FG89] E. R. Fiala and D. H. Greene. Data compression with finite windows. *Commun. ACM*, 32(4):490–505, 1989.

[FM95] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, 1995.

[FT01] G. Felici and K. Truemper. A minsat approach for learning in logic domains. *INFORMS Journal on Computing*, 13(3):1–17, 2001.

[FT02] G. Felici and K. Truemper. A minsat approach for learning in logic domains. *Informs Journal on Computing*, 14:20–36, 2002.

[FT06] G. Felici and K. Truemper. *J. Wang (ed.), Encyclopedia of Data Warehousing and Mining*, volume 2, chapter The Lsquare System for Mining Logic Data, pages 693–697. Idea Group Inc., 2006.

[GK07] Joshua Grochow and Manolis Kellis. Network Motif Discovery Using Subgraph Enumeration and Symmetry-Breaking. *Research in Computational Molecular Biology*, pages 92–106, 2007.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[Gol66] Solomon W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[Gol95] Stuart J. Golin. A simple variable-length code. *Signal Processing*, 45:23–35, 1995.

[GvV75] R. G. Gallager and D. C. van Voorhis. Optimal source codes for geometrically distributed integer alphabets. *Information Theory, IEEE Transactions on*, 21(2):228–230, 1975.

[HBPS07] Fereydoun Hormozdiari, Petra Berenbrink, Nataša Pržulj, and S. Cenk Sahinalp. Not All Scale-Free Networks Are Born Equal: The Role of the Seed Graph in PPI Network Evolution. *PLoS Comput Biol*, 3(7):e118+, July 2007.

[Huf52] D. Huffman. A method for construction of minimum-redundancy codes. *Proc. of IRE*, 40(9):1098–1101, September 1952.

[KCA09] Chinmay Karande, Kumar Chellapilla, and Reid Andersen. Speeding up algorithms on compressed web graphs. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 272–281, New York, NY, USA, 2009. ACM.

[KHP+07] Maksim Kitsak, Shlomo Havlin, Gerald Paul, Massimo Riccaboni, Fabio Pammolli, and H. Eugene Stanley. Betweenness centrality of fractal and nonfractal scale-free model networks and tests on real networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 75(5):056115, 2007.

[KIMA04] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20:1746–1758, July 2004.

[Kle99] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[Lev68] V. I. Levenstein. On the redundancy and delay. `http://www.compression.ru/download/articles/int/`, 1968.

[LFS06] Vincent Lacroix, Cristina G. Fernandes, and Marie-France F. Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 3(4):360–368, October 2006.

[LM99] N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. *Data Compression Conference*, 0:296, 1999.

[LTZ97] T. Linder, V. Tarokh, and K. Zeger. Existence of optimal codes for infinite source alphabets. *IEEE Transactions on Information Theory*, 43(6):2026–2028, 1997 1997.

[MM09] W.P. Maddison and D.R. Maddison. Mesquite: a modular system for evolutionary analysis. http://mesquiteproject.org/, 2009.

[MP10] Hossein Maserrat and Jian Pei. Neighbor query friendly compression of social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 533–542, New York, NY, USA, 2010. ACM.

[MSV06] R. Meier, K. Shiyang, and G. Vaidya. Dna barcoding and taxonomy in diptera: a tale of high intraspecific variability and low identification success. *Systematic Biology*, 55:715–728, 2006.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[PCJ04] N. Pržulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20:3508–3515, December 2004.

[PCS04] E. Paradis, J. Claude, and K. Strimmer. Ape: analyses of phylogenetics and evolution in r language. *Bioinformatics*, 20:289290, 2004.

[Pig01] S. Pigeon. Start/stop codes. *Data Compression Conference*, 0:0511, 2001.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1 edition, January 1993.

[Qui96] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[Ric79] Robert F. Rice. Some practical universal noiseless coding techniques. Technical Report JPL-79-22, Jet Propulsion Laboratory, Pasadena, CA, March 1979.

[Ris76] Jorma Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3):198–203, 1976.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[SLS09] Sophie Schbath, Vincent Lacroix, and Marie-France Sagot. Assessing the exceptionality of coloured motifs in networks. *EURASIP J. Bioinformatics Syst. Biol.*, 2009:3:1–3:9, January 2009.

[SN87] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 1987.

[SY01] Torsten Suel and Jun Yuan. Compressing the graph structure of the web. In *Proceedings of the IEEE Data Compression Conference (DCC*, pages 213–222, 2001.

[Teu78] J. Teuhola. A compression method for clustered bit-vectors. *Information Processing Letters*, 7:308–311, October 1978.

[Tru04] K. Truemper. *Design of Logic-Based Intelligent Systems*. Wiley-Interscience, 2004.

[Tur84] G. Turan. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8:289–294, 1984.

[Wer06] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3:347–359, October 2006.

[Wyn72] A.D. Wyner. An upper bound on the entropy series. *Information and Control*, 20(2):176 – 181, 1972.

[Yok88] H. Yokoo. An efficient representation of the integers for the distribution of partial quotients over the continued fractions. *J. Inform. Processing*, 1988.

[YQ06] S. Yang and P. Qiu. Efficient integer coding for arbitrary probability distributions. *IEEE Transactions on Information Theory*, 52(8):3764–3772, 2006.