UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Tesi di Laurea Triennale

SUBGRAPH SIMILARITY IN COMPLEX NETWORKS

SIMILARITÀ DI SOTTOGRAFI NELLE RETI COMPLESSE

Relatori:
Prof. *Roberto Grossi*
Prof. *Andrea Marino*

Candidato:
Gaspare Ferraro

ANNO ACCADEMICO 2016-2017

Contents

Chapter 1

Introduction

With the spread of the Internet and more importantly of the social networks, efficient data analysis on graphs becomes increasingly important.

Graphs are a powerful data structure that natural model the interactions between objects.

1.1 Basic definitions

Definition 1.1. A **graph** is a pair of sets $G = (V, E)$, where V is the set of vertices (or nodes) and $E \subseteq V \times V$ is the set of edges.

If two vertices $u, v \in V$ are connected by an edge they are called the extremes of the edge, in this case we denote the edge with the pair $(u, v) \in E$.

If $(u, v) \in E \Leftrightarrow (v, u) \in E$ the graph is called undirected; wherever not specified we will only deal with undirected graphs.

A sequence of nodes $v_1, v_2, \ldots, v_k$ is called path if $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots k-1$; a path is called simple if $v_i \neq v_j$ for all i, j such that $1 \leq i < j \leq k$. A cycle is a path where $(v_k, v_1) \in E$.

We denote by $N(u) = \{v : (u, v) \in E\}$ the set of neighbors of the vertex u. The cardinality of this set is called degree of u (i.e. $deg\ u = |N(u)|$).

With $N^{<k}(u)$ we indicate the set of vertices connected to u by a simple path of length less than k (note that $N(u) = N^{<2}(u)$).

Definition 1.2. A graph $G' = (V', E')$ is called **subgraph** of $G = (V, E)$ if $V' \subset V$ and $E' \subset E$. A subgraph is called **induced subgraph** if $E' = (V' \times V') \cap E$.

We use $G' \subset G$ to indicate that the graph G' is a subgraph of G and $G' < G$ to indicate that the graph G' is an induced subgraph of G.

Note that an induced subgraph $G' = (V', E')$ can be uniquely identified by the set of its vertices V'.

Definition 1.3. A **labeled graph** is a triple (V, E, L) where (V, E) is a graph and $L : V \to \Sigma$ is a function that assigns a symbol of the alphabet Σ to every node v. We call $L(u) \in \Sigma$ the label of the node u.

Given a path $\pi = v_1, v_2, \ldots, v_k$ we extend the function L and define the label sequence of π the string $L(\pi) = L(v_1)L(v_2)\ldots L(v_k) \in \Sigma^k$ obtained by the concatenation of the labels of the nodes in the path.

In this thesis, we mainly focus on analyzing complex networks: special graphs with a non-trivial topology, different from random graphs. Complex networks occurs in graphs that model real system like social networks or computer networks, and are characterized by a specific structural feature:

Definition 1.4. We define as **power-law degree distribution** a network where the degree of the nodes u follow, for some γ (usually $2 < \gamma < 3$), the probability distribution:

$$P(deg(u) = k) \sim k^{-\gamma} \tag{1.1}$$

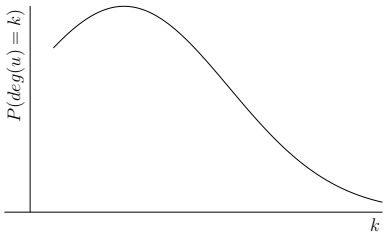


Figure 1.1: Degree distribution of a random network

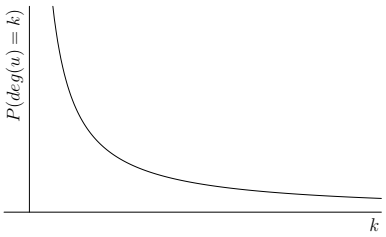


Figure 1.2: Degree distribution of a complex network

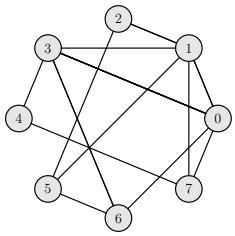


Figure 1.3: Random network

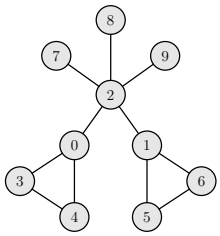


Figure 1.4: Complex network

1.2 The problem

Problem 1.5. Given an undirected labeled graph $G = (V, E, L)$ over an alphabet Σ, an integer q and two set of nodes $V_1, V_2 \subset V$, we want to estimate the similarity between the two induced subgraphs $V_1, V_2 < G$, based on the labels frequency of simple paths with nodes in $V_1 \cup N^{<q}(V_1)$ and $V_2 \cup N^{<q}(V_2)$ that ends in V_1 and V_2.

We will talk about a more formal and rigorous definition of subgraphs similarity in chapter 2.

In the definition we use $V_1 \cup N^{<q}(V_1)$ and $V_2 \cup N^{<q}(V_2)$ instead of simply V_1 and V_2 because, in a complex graph, we also want to keep in mind of the interaction between each subgraph and the rest of the graph.

The difficulty we must face is that, in a complex network, the labels can exponentially explode for increasing values of q and $|\Sigma|$, to $|\Sigma|^q \gg |V|$. Even worse, the number of simple paths can exponentially explode to $|V|^q$. The simple reason is that in complex networks the average separation is very low (the famous idea of *six degrees of separation*).

In this thesis, we exploit the problem using randomized techniques and parallelization, which makes the problem suitable even for large networks.

1.3 Practical applications

The problem presented can be applied to a lot of contexts. We illustrate some examples of practical application by referring to famous websites, in order to better understand. In all the examples we assume to work on a *friendship graph* where the nodes are the registered users and the edges are the friendship relations between two users.

Netflix In order to produce and translate the right television series, we want to estimate the similarity between two geographical groups of viewers, where each viewer is labeled with its favorite film genre.

Amazon In order to improve advertising campaigns, we want to estimate the similarity between two sets of clients of the same age group, where each client is labeled with the category of objects he buys more frequently.

Facebook In order to suggest new friends, we want to estimate the similarity between two users, where each user is labeled with its favorite musical genres.

1.4 Thesis organization

The thesis is divided into five chapters and one appendix, where the topics covered are respectively the following ones.

- Chapter 1: **Introduction**, where we present some basic definitions in order to introduce the problem we are analyzing.
- Chapter 2: **Basic tools**, where we present the definition of some similarity indices already existing in the literature, along with two methods we will use.
- Chapter 3: **Computation of subgraph similarity**, where we give a description of different approaches to compute the subgraph similarity.
- Chapter 4: **Project development**, where we perform implementation choices and steps of the project development with some experimental results.
- Chapter 5: **Conclusion and future works**, where we draw some conclusions from theoretical and practical analysis of the presented approaches, with some hypothesis of future works.
- Appendix A: **Code snippets**, where we give some important code snippets of the project used for the experimental results.

The results in this thesis are joint work with Alessio Conte, Roberto Grossi, Andrea Marino, Kunihiko Sadakane and Takeaki Uno [10].

Chapter 2

Basic tools

In this chapter, we introduce some notion of similarity already existing in the literature and then extending them to define similarity among subgraphs in labeled graphs.

In the last sections we present two different techniques we will use afterwards to estimate such similarity.

2.1 Similarity indices

Definition 2.1. Given two sets A and B we define the **Jaccard** index as the size of the intersection divided by the size of the union between the two sets

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

Definition 2.2. Given two sets A and B we define the **Bray-Curtis** index as

$$BC(A,B) = \frac{2 \times |A \cap B|}{|A| + |B|} \tag{2.2}$$

Example 2.3. Given $A = \{1,3,4,5,7,8\}$ and $B = \{1,2,4,6,8\}$ we have

$$J(A,B) = \frac{|\{1,4,8\}|}{|\{1,2,3,4,5,6,7,8\}|} = \frac{3}{8}$$

$$BC(A,B) = \frac{2 \times |\{1,4,8\}|}{|\{1,3,4,5,7,8\}| + |\{1,2,4,6,8\}|} = \frac{6}{11}$$

Note that when $A = B$ we have $J(A,B) = BC(A,B) = 1$ and when $A \cap B = \emptyset$ we have $J(A,B) = BC(A,B) = 0$.

Using sets may be limiting as we consider only once the repeated values. Thus we can extend the two previous definitions to the multisets.

Definition 2.4. A **multiset** is a generalization of set that allows multiple instances of elements.

To avoid confusion afterwards we use square brackets [ ] to indicate multisets and curly brackets { } to indicate sets.

A multiset can also be seen as an array of frequencies of its objects: for instance, we can indicate with $A = (2, 0, 3)$ the multiset with 2 elements of the first type, 0 elements of the second type and 3 elements of the third type, where this notation is equivalent to write $A = [1, 1, 3, 3, 3]$.

Given two multisets $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ we define the following operations.

- intersection $C = A \cap B = (c_1, \ldots, c_n)$ where $c_i = \min(a_i, b_i)$
- union $C = A \cup B = (c_1, \ldots, c_n)$ where $c_i = \max(a_i, b_i)$
- disjoint union $C = A \uplus B = (c_1, \ldots, c_n)$ where $c_i = a_i + b_i$

Definition 2.5. Given two multisets $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ we define the **Bray-Curtis** index as

$$BC(A, B) = \frac{2 \times \sum_{i=1}^{n} min(a_i, b_i)}{\sum_{i=1}^{n} a_i + b_i} \tag{2.3}$$

As a side note, Bray-Curtis is a relevant index for multisets, and is also known as Steinhaus similarity, Pielou's Similarity, Sorensen's quantitative, and Czekanowski's similarity [16].

Example 2.6. Given $A = (0, 2, 3, 1, 0, 3)$ and $B = (2, 0, 1, 3, 1, 2)$ we have

$$J(A, B) = \frac{0 + 0 + 1 + 1 + 0 + 2}{2 + 2 + 3 + 3 + 1 + 2} = \frac{4}{13} \tag{2.4}$$

$$BC(A, B) = \frac{2 \times (0 + 0 + 1 + 1 + 0 + 2)}{2 + 2 + 4 + 4 + 1 + 4} = \frac{8}{17} \tag{2.5}$$

Both indices are widely used in practical application, Bray-Curtis index gives a larger weight to the intersection, on the other side the Jaccard Index is a metric and may be preferred to Bray-Curtis index since the latter is only a semi-metric (as it does not satisfy the triangle inequality).

Definition 2.7. A function $f : A \times A \to \mathbb{R}^+$ is called **metric** (or simply **distance**) if it satisfies, for all $x, y, z \in A$, the following conditions.

- Non-negativity: $f(x, y) \geq 0$ and $f(x, y) = 0$ iff $x = y$
- Symmetry: $f(x, y) = f(y, x)$
- Triangle inequality: $f(x, z) \leq f(x, y) + f(y, z)$

2.2 Documents similarity

Document similarity is a hot topic in Information Retrieval, as it can be seen as the problem of duplicates detection [6] or, from another point of view, plagiarisms detection.

To define document similarity we need the notion of *q-gram*:

Definition 2.8. A **q-gram** is a contiguous subsequence of q items from a sequence.

In this case the sequence is a document and the items can be words, characters or even syllables. If the elements used are words, q-grams can even be called shingles.

Example 2.9. Given the document *"I live and study in Pisa"* all the possible 3-grams, where items are words, are the following ones: *"I live and"*, *"live and study"*, *"and study in"* and *"study in Pisa"*.

Note that in a document with n words the possible q-grams are exactly $n - q + 1$.

It is easy to see if we use the set, or multiset, of the all possible q-grams of two documents we can use it to calculate their similarity based on the Jaccard or Bray-Curtis index.

Considering that the number of q-grams in a document is linear in its number of words, document similarity is not an hard problem as we have only to perform union and intersection between sets, or multisets.

Example 2.10. Given the documents

$$A = \textit{I live, work and study in Pisa}$$

$$B = \textit{You work and study in Livorno}$$

The set of their 2-grams are:

$$S_A = (\textit{I live}, \textit{live work}, \textit{work and}, \textit{and study}, \textit{study in}, \textit{in Pisa})$$

$$S_B = (\textit{You work}, \textit{work and}, \textit{and study}, \textit{study in}, \textit{in Livorno})$$

The similarity using both Jaccard and Bray-Curtis are

$$J(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} = \frac{3}{8}$$

$$BC(S_A, S_B) = \frac{2 \times |S_A \cap S_B|}{|S_A| + |S_B|} = \frac{6}{11}$$

2.3 Graphs similarity

The definition of similarity between graphs is more complex, as we have to introduce the concept of graph isomorphism.

Definition 2.11. A **graph isomorphism** between two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is a one-to-one function from vertices of G to vertices of H that preserve the edge structure, i.e.

$$f : V_G \to V_H \text{ s.t. } (v, u) \in E_G \implies (f(v), f(u)) \in E_H \tag{2.6}$$

If such isomorphism exists, the graphs are called isomorphic and we denote it with $G \simeq H$.

With the notion of graph similarity we can define a similarity between two graphs [8].

Definition 2.12. The similarity between two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is the size of the largest graph isomorphism between a subgraph $G' \subseteq G$ and a subgraph $H' \subseteq H$ (seen as number of vertex of G').

Unfortunately subgraphs isomorphism is an NP-complete problem [13], so the only way to solve it is by using heuristic or approximated methods.

Example 2.13. The two graphs are isomorphic with the function $f : V_G \to V_H$ s.t. $f(0) = 3$, $f(1) = 4$, $f(2) = 2$, $f(3) = 0$, $f(4) = 5$ and $f(5) = 1$.

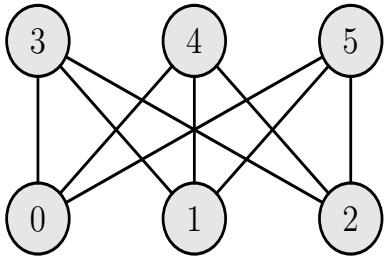


Figure 2.1: Graph $G = (V_G, E_G)$

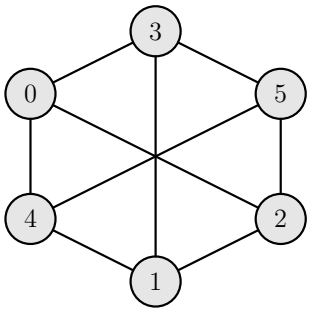


Figure 2.2: Graph $H = (V_H, E_H)$

2.4 Subgraphs similarity

After discussing the already existing notions of similarity, we are ready to extend them to define similarity in a labeled complex network.

Consider a labeled graph $G = (V, E, L)$ over an alphabet Σ where $L \to \Sigma$ is the node labeling, so that each node $u \in V$ has a label $L(u) \in \Sigma$[1], we are interested in analyzing G using the sequences of labels found along its paths.

For a fixed integer $q > 0$, consider an arbitrary simple path $\pi = u_1, \ldots, u_q$. We call the orientation $u_1 \to \ldots \to u_q$ of π a q-path leading to u_q and $L(\pi) = L(u_i) \ldots L(u_q) \in \Sigma^q$ its q-gram, obtained by concatenating the labels of its nodes.[2].

For a set of nodes $A \subseteq V$, we define $L(A)$ as the corresponding multiset of q-grams for all q-path π leading to a node $u \in A$.

$$L(A) = [x \in \Sigma^q : \exists\ q\text{-path } \pi \text{ leading to } u \in A \text{ with } L(\pi) = x] \tag{2.7}$$

In this way, for each q-path $\pi = u_1, \ldots, u_q$ leading to $u_q \in A$, we have that $u_i \in A \cup N^{<q}(A)\ \forall\ 1 \leq i < q$.

This is a good definition because, as it was mentioned before, we take into account both the internal structure of A and its neighborhood $N^{<q}(A)$.

Note that we explicitly exclude all the q-paths that are beginning and starting outside A, as we not considering them influential to define the similarity.

[1]Alternatively we can labeling edges in E instead of nodes in V without making too many changes in the following definitions, but for the sake of simplicity we consider the graph labeled on its nodes.

[2]Note that in an undirected graph we have, for a single simple path, two possible q-paths, one for each orientation: one leading to u_q from u_1 and one leading to u_1 from u_q.

Given a single q-gram x we are interested in its frequency within the multiset $L(A)$ so we define

$$f_A[x] = |\{\pi : \pi \text{ is a } q\text{-path leading to } u \in A \text{ and } L(\pi) = x\}| \tag{2.8}$$

Note the property that $f_A[x] = \sum_{u \in A} f_{\{u\}}[x]$.

Definition 2.14. Given an undirected labeled graph $G = (V, E, L)$ over an alphabet Σ and an integer $q > 0$, the Bray-Curtis similarity index between two set of nodes $A, B \subset V$ is:

$$BC(A, B) = \frac{2 \times \Sigma_{x \in \Sigma^q} \min(f_A[x], f_B[x])}{\Sigma_{x \in \Sigma^q} f_A[x] + f_B[x]} \tag{2.9}$$

Definition 2.15. Given an undirected labeled graph $G = (V, E, L)$ over an alphabet Σ and an integer $q > 0$, the Frequency Jaccard similarity index between two set of nodes $A, B \subset V$ is:

$$FJ(A, B) = \frac{\Sigma_{x \in \Sigma^q} \min(f_A[x], f_B[x])}{\Sigma_{x \in \Sigma^q} f_{A \cup B}[x]} \tag{2.10}$$

Let $\mathcal{L} = \{x \in \Sigma^q : x \in L(V)\} \subseteq \Sigma^q$ be the set of all distinct q-grams found in the q-paths of G.

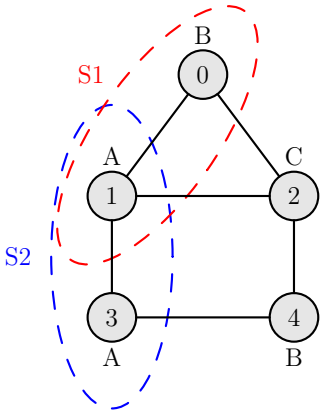
Note that ranging x over $\mathcal{L}$, instead of Σ^q, is sufficient in both the formulas for any A and B.

In general we have that $BC(A, B) \geq FJ(A, B)$. When $A \cap B = \emptyset$ we have that $f_{A \cup B}[x] = f_A[x] + f_B[x]$ and $BC(A, B) = 2 \times FJ(A, B)$

Now we present a little example to better understand our notions.

Example 2.16.

We want to calculate the similarity between the two set $S_1 = \{0, 1\}$ and $S_2 = \{1, 3\}$ using their 3-grams of this graph:



$$L(S_1) = [aab, acb, baa, bca, bca, bcb, cab, cba]$$

$$L(S_2) = [baa, baa, bca, bca, caa, cba, cba]$$

So we have that:

$$FJ(S_1, S_2) = \frac{4}{11} \text{ and } BC(S_1, S_2) = \frac{8}{15}$$

2.5 Sketches

We have seen that computing the exact similarity between two documents is an easy problem as we have only to compute the union and the intersection between the two sets of shingles.

This task becomes more difficult to handle when we have to consider thousands or millions of documents (e.g. the set of Internet web pages), each one of them has thousands of shingles.

To solve this problem, it is no longer possible to handle all the shingles for all the documents, instead we can, for each of them, keep a relatively small, fixed size *sketch* [6].

The computation of the sketches is linear in the size of the documents and can be used to calculate the similarity in linear time in the size of the sketches only.

In the next chapter, we will use the sketches applied to the set of q-grams of a labeled graph to fast compute the similarity between two subgraphs.

Usually sketches are used with explicit documents, we will use it to avoid to generate all the q-grams.

Now we present two different existing techniques to compute the sketches of a document, for simplicity we assume that our document is composed of all numbers between 1 and n, where in practice we use a ranking function to define an ordering of the items in the documents.

Min-wise permutation

The first approach to compute a sketch of a document is the min-wise permutation [7].

Given a document $A = \{1, \ldots, n\}$, to calculate its sketch S_A of size k we choose k random independent permutations $\pi_i : \{1, \ldots, n\} \to \{1, \ldots, n\}$ and define the sketch S_A of A as:

$$S_A = \{\min(\pi_1(A)), \ldots, \min(\pi_k(A))\} \tag{2.11}$$

Note that using the min-wise permutation we can take multiple times the same number, as the permutations are independently and uniformly chosen.

Bottom-k sketches

Another approach that works best in terms of performance is the bottom-k sketches [9].

Instead of using k random permutation $\pi_1, \ldots, \pi_k$ and taking the minimum from each of them, as we did before in the min-wise permutation, we choose only one random permutation π and take the bottom k elements in the permutation, where $\min_x$ indicates the xth smallest element in the permutation.

$$S_A = \{\min_1(\pi(A)), \ldots, \min_k(\pi(A))\} \tag{2.12}$$

Note that, unlike the min-wise permutation, in a bottom-k sketch we do not have repeated numbers as we take the numbers from a single permutation.

Example 2.17. Consider the document $A = \{1, \ldots, 10\}$ and the following 4 permutation ($k = 4$)

$$\pi_1 = \{3, 5, 8, 2, 4, 9, 1, 10, 7, 6\}$$
$$\pi_2 = \{7, 10, 2, 1, 8, 5, 9, 6, 4, 3\}$$
$$\pi_3 = \{3, 4, 6, 2, 8, 5, 1, 10, 7, 9\}$$
$$\pi_4 = \{9, 1, 3, 5, 4, 10, 7, 8, 2, 6\}$$

With the min-wise permutation approach we have that

$$S_A = \{\min(\pi_1(A)), \min(\pi_2(A)), \min(\pi_3(A)), \min(\pi_4(A))\} = \{3, 7, 3, 9\}$$

Instead with the bottom-k sketches using π_4 as permutation

$$S_A = \{\min_1(\pi_4(A)), \min_2(\pi_4(A)), \min_3(\pi_4(A)), \min_4(\pi_4(A))\} = \{9, 1, 3, 5\}$$

2.6 Color Coding

The color coding is a method proposed in 1994 by Alon et al. [2] that efficiently finds simple path, cycles or many other small subgraphs using a probabilistic and parameterized algorithm. We will focus only in finding q-simple paths.

The idea behind this method, which gives it the name, is to randomly coloring each node of V with one of the q possible colors.

We restrict our attention to $q = O(\log |V|)$ and denote with $\chi : V \to [q]$[3] the coloring function, where each node $u \in V$ has a color $\chi(u) \in [q]$.

[3]Here $[q] \equiv [1, \ldots, q]$ is the set of all possible colors

After assigning a color to each node, we will focus on finding only the colorful q-paths. We say that a q-path $u_1, \ldots, u_q$ is colorful iff $\chi(u_i) \neq \chi(u_j)$ for $1 \leq i < j \leq q$ (i.e. all the q colors appear in the q-path).

The main advantage of this method is that it reduces the number of q-paths by roughly a factor of $q!/q^q \geq 1/e^q$, as a colorful q-path can use $q!$ colorings of its nodes out of q^q possible ones. So the number of q-paths exponentially decrease as the value of q increases (e.g. when $q = 3$ we look only for the $\sim 22\%$ colorful q-paths and only for $\sim 4\%$ when $q = 5$).

As we will show in the next chapter, all the colorful q-paths can be easily found using a dynamic programming approach.

An interesting fact is that the method of color coding can be derandomized using a q-perfect hash family [2], that enumerates all the possible q-colorings of V that are relevant to the computation (as otherwise there would be n^q possible q-colorings). This makes the method exponential in the value of q, as if we set $q = |V|$ the problem became to find an Hamiltonian path, which is NP-complete [13].

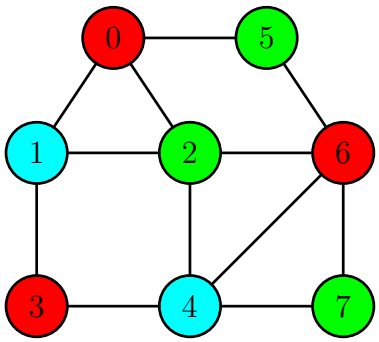
To improve the precision we can repeat the random coloring of nodes for e^q t times, so that the success probability becomes $\geq 1 - e^{-t}$. Another way is to randomly coloring nodes using a larger number of colors q' and then, instead of looking for the colorful q'-paths, we search the color-diversified q-paths (i.e. q-paths without color repetition in $[q']$) [11].

In practice, choosing a simple random coloring is working pretty well on complex networks, and we do not use these optimizations thus avoiding this overhead for our algorithms.

Example 2.18. In this 3-colored graph out of 6 simple 3-paths starting from 0

(0-1-2 0-1-3 0-2-1 0-2-4 0-2-6 0-5-6)

only 3 are colorful (0-1-2 0-2-1 0-2-4).



Chapter 3

Computation of subgraph similarity

In this chapter, we present four different theoretical algorithms to compute subgraphs similarity as previously defined: an exhaustive enumeration, two similar randomized approaches using the tools described in the previous chapter, and a naive randomized approach as a baseline for comparison.

In the following algorithms, we will make use of parallel instructions, postponing the specific programming choices and the comparison among the different approaches to the next chapter.

3.1 Indices computation

Now we illustrate the algorithms to calculate the Frequency Jaccard and Bray-Curtis indices, as they are independent from the next algorithms we will present.

As previously seen, instead of iterating over all the strings in Σ^q we can restrict to $\mathcal{L} \subseteq \Sigma^q$, which is the set of all possible q-grams found in the q-paths of G.

An additional improvement can be made as follows. If we want to compute the similarity between two set $A, B \subset V$, it is enough to ranging, instead over $\mathcal{L}$, over $\mathcal{W} = \{x \in \Sigma^q : x \in L(A) \text{ or } x \in L(B)\} \subseteq \Sigma^q$, as we can easily see that for any $x \in (\Sigma^q \setminus \mathcal{W})$ both $f_A[x]$ and $f_B[x]$ are equal to zero.

Also, we can observe that in the Frequency Jaccard index we do not have to explicitly calculate $f_{A \cup B}[x]$ and its sketch, as the exact value of $R = \Sigma_{x \in \mathcal{W}} f_{A \cup B}[x]$ can be easily calculated from $f_A[x]$ and $f_B[x]$.

So we define the following procedures based on (2.9) and (2.10).

Algorithm 1: BRAY-CURTIS

Input : $\mathcal{W}$ = dictionary of q-grams
$f_A[x]$ = frequency of each $x \in \mathcal{W}$ in A
$f_B[x]$ = frequency of each $x \in \mathcal{W}$ in B
Output: $BC(A, B)$ = the similarity between A and B according to Bray-Curtis index

```
num ← 0
den ← 0
foreach x ∈ W do
    num ← num + 2 × min(f_A[x], f_B[x])
    dem ← den + f_A[x] + f_B[x]
BC ← num/den
return BC
```

Algorithm 2: FREQUENCY-JACCARD

Input : $\mathcal{W}$ = dictionary of q-grams
$f_A[x]$ = frequency of each $x \in \mathcal{W}$ in A
$f_B[x]$ = frequency of each $x \in \mathcal{W}$ in B
R = summation of all frequency
Output: $FJ(A, B)$ = the similarity between A and B according to Frequency Jaccard index

```
num ← 0
foreach x ∈ W do
    num ← num + min(f_A[x], f_B[x])
FJ ← num/R
return FJ
```

Algorithm 1 and 2 compute the values of $BC(A, B)$ and $FJ(A, B)$, as previously defined, by ranging over the given dictionary of q-grams $\mathcal{W}$.

Lemma 3.1. *The execution of* BRAY-CURTIS *or* FREQUENCY-JACCARD *requires* $O(|W|)$ *time and* $O(1)$ *space.*

In the next algorithms, we will only focus to compute the values of $\mathcal{W}$, f_A, f_B and R.

3.2 Naive approach

The naive approach consists in enumerating all the possible q-grams of simple q-paths leading to $u \in A \cup B$. This can be done by starting an exhaustive search for each $u \in A \cup B$.

Algorithm 3: BRUTE-FORCE

Input : q = length of the paths
A, B = set of nodes to compare
Output: $\mathcal{W}$ = dictionary of q-grams
$f_A[x]$ = frequency of each $x \in \mathcal{W}$ in A
$f_B[x]$ = frequency of each $x \in \mathcal{W}$ in B
R = summation of all frequency

```
1  R ← 0
2  W ← ∅
3  f_{A∪B} ← ∅
4  f_A ← ∅
5  f_B ← ∅
6  parallel foreach u ∈ A ∪ B do
7      ⟨W_u, f_u⟩ ← ExhaustiveSearch(⟨u⟩, q)
8      W ← W ∪ W_u
9      f_{A∪B} ← f_{A∪B} ∪ f_u
10 foreach ⟨u, x⟩ ∈ f_{A∪B} do
11     R ← R + f_{A∪B}[⟨u, x⟩]
12     if u ∈ A then
13         f_A[x] ← f_A[x] + f_{A∪B}[⟨u, x⟩]
14     if u ∈ B then
15         f_B[x] ← f_B[x] + f_{A∪B}[⟨u, x⟩]
16 return ⟨W, f_A, f_B, R⟩
```

Here we define $\mathcal{W}_u$ and f_u as, respectively, the dictionary and the frequency of q-grams of the q-paths leading to the node $u \in A \cup B$.

Thus we compute $\mathcal{W}$ with the property $\mathcal{W} = \Sigma_{u \in A \cup B}\ \mathcal{W}_u$ and, in the same way, $f_{A \cup B}$ with the property $f_{A \cup B} = \Sigma_{u \in A \cup B}\ f_u$.

The value of R is computed as described at the beginning of the chapter, namely, $R = \Sigma_{x \in \mathcal{W}}\ f_{A \cup B}[x]$.

At last, we can compute the value of f_A and f_B from $f_{A \cup B}$ by looking at the leading nodes and separate the frequencies, depending if it belongs to A, B or both.

Note that, as we have to separate the frequencies between f_A and f_B, the type of $f_{A\cup B}$ is not a mapping $\Sigma^q \to \mathbb{N}$ but instead is a mapping $V \times \Sigma^q \to \mathbb{N}$, where the element in V is the leading node of the q-path associated with the q-gram.

The values of $FJ(A, B)$ and $BC(A, B)$ computed using this method are exact, and we will use it only to evaluate the precision of the other approaches, as it requires to examine all the possible $O(|\Sigma|^q)$ q-gram with a complexity of $O(|V|^q)$.

For completeness, we also illustrate the ExhaustiveSearch algorithm that keeps track of the current q-path and its relative q-gram.

Algorithm 4: ExhaustiveSearch

Input : $\pi = \langle u_1, \ldots, u_{|\pi|}\rangle$ current traversing path of length $\leq q$
q = length of the paths
Output: $\mathcal{W}$ = dictionary of q-grams of q-path having π as suffix
$f_u[\langle u_q, x\rangle]$ = frequency of each $x \in \mathcal{W}$ leading to u_q

```
1  W ← ∅
2  f_u ← ∅
3  if |π| = q then
4      W ← {L(π)}
5      f_u[⟨u_q, L(π)⟩] ← 1
6  else
7      foreach v ∈ N(u_1) \ π do
8          ⟨W_v, f_v⟩ ← ExhaustiveSearch(⟨v⟩ · π, q)
9          W ← W ∪ W_v
10         f_u ← f_u ∪ f_v
11 return ⟨W, f_u⟩
```

Here the symbol $\cdot$ is the concatenation of the paths. Note that we put the node v before the path π as we are interested to find all the q-paths leading to the node u.

In Algorithm 4, when π is a q-path, we have the base case of the recursion that simply returns $\mathcal{W} = \{L(\pi)\}$, which is the dictionary composed only by the label of π, and the frequency $f_u[\langle u_q, L(\pi)\rangle] = 1$ as we have only one path.

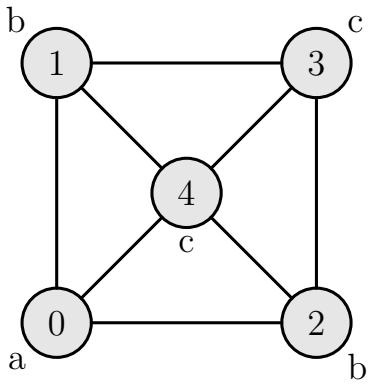
When the path π is shorted than q, we recursively visit all its neighbors, with the new path obtained by prepending the node v to the current path π.

Finally, using $N(u_1) \setminus \pi$ we avoid to traverse again the nodes already in the path π. In this way we restrict the searching only to the simple q-paths.

Lemma 3.2. *For any two sets of nodes $A, B \subseteq V$, the running time of* BRUTEFORCE *requires $O(|V|^q)$ time and $O(\mathcal{L}) = O(|\Sigma|^q)$ space.*

Now we present a little example to better understand our ideas.

Example 3.3. We want to compute the similarity between the two nodes 4 and 3 in the following graph.



ExhaustiveSearch$(4, 3)$ returns

$\mathcal{W}_4 = \{abc, bac, bbc, cbc\}$

$f_4[\langle 4, abc\rangle] = 2$ (3-1-0 4-1-0 3-2-0 4-2-0)

$f_4[\langle 4, bac\rangle] = 2$ (1-4-0 2-4-0)

$f_4[\langle 4, bbc\rangle] = 2$ (3-4-0)

$f_4[\langle 4, cbc\rangle] = 2$ (3-4-0)

ExhaustiveSearch$(3, 3)$ returns

$\mathcal{W}_3 = \{abc, acc, bcc, cbc\}$

$f_3[\langle 3, abc\rangle] = 2$ (0-1-3, 0-2-3)

$f_3[\langle 3, acc\rangle] = 1$ (0-4-3)

$f_3[\langle 3, bcc\rangle] = 2$ (1-4-3, 2-4-3)

$f_3[\langle 3, cbc\rangle] = 2$ (4-1-3, 4-2-3)

So the dictionary $\mathcal{W}$ is

$\mathcal{W} = \mathcal{W}_3 \cup \mathcal{W}_4 = \{abc, acc, bac, bbc, bcc, cbc\}$

The similarity according to the two indices is

$BC(\{3\}, \{4\}) = \frac{2\times(2+0+0+0+0+2)}{4+1+2+2+2+4} = \frac{8}{15}$

$FJ(\{3\}, \{4\}) = \frac{2+0+0+0+0+2}{4+1+2+2+2+4} = \frac{4}{15}$

3.3 Efficient computation

The main hurdle of our problem is to compute the frequency mapping $f_X[\,]$ for some sets $X \in V$, as it can grow up to a size of $|\Sigma|^q$, and its definition requires to explore potentially $|V|^q$ q-paths.

We present a random unbiased estimator based on color coding and sketching with the property that it can be computed efficiently even on large networks, and its expected value is the actual similarity index [10].

First, using the color coding we reduce the number of potentially explored q-paths from $|V|^q$ to $2^{O(q)}|V|$, thus making it feasible for large values of $|V|$, assuming $q = O(\log |V|)$.

Second, instead of calculating the correct value of f_X, we compute its sketch with a size small compared to $|\Sigma|^q$, which is a significant benefit when $|\Sigma|$ or q are large.

preprocess(G, q): Color coding of the q-paths

Now we illustrate how to preprocess the input graph $G = (V, E)$ given an integer $q > 0$, in particular, where $q = O(\log |V|)$.

Note that the preprocessing task is performed independently from the choice of the labeling function L and the subsets A, B to compare. It depends only on the graph G and the value of q, so we can execute the preprocessing once and then reuse the same color coding table for different values of A, B or even L.

Algorithm 5: PREPROCESS: COLOR-CODING

Input : $G = (V, E)$ undirected graph with q random colors.
Output: M = dynamic programming table for color coding.

```
1 parallel foreach u ∈ V do M_{1,u} = ⟨χ(u), 1⟩
2 for i ∈ {2, 3, ..., q} do
3     parallel foreach u ∈ V do
4         foreach v ∈ N(u) do
5             foreach ⟨C, f⟩ ∈ M_{i−1,v} such that χ(u) ∉ C do
6                 f′ ← M_{i,u}(C ∪ {χ(u)})
7                 M_{i,u} ← ⟨C ∪ {χ(u)}, f′ + f⟩
8 return M
```

The next goal is to list all the colorful q-paths in G using a dynamic programming approach.

First of all we assign a random coloring $\chi : V \to [q]$, so that each node $u \in V$ has a color $\chi(u)$ independently and uniformly chosen from $[q]$. Algorithm 5 build and return a table M of size $q \times |V|$ where $M_{i,j}$ stores the collection of pairs $\langle C, f \rangle$ where $C \subseteq [q]$ is a color set such that $|C| = i$ and there are f colorful i-paths leading to the node j.

Our assumption that $q = O(\log |V|)$ allows us to implement, using bit manipulations, operations on color sets in $O(1)$ time as they fit in a machine words.

Note that each entry $M_{i,j}$ contains at most $\binom{q}{i}$ sets, each with i colors. Hence the computation of the row i can be done in parallel as it depends only from the row $i-1$ and require $O(|E| \binom{q}{i-1})$ time (as we scan all the adjacency list). The entire computation requires thus $O(|E| \Sigma_{i=1}^{q} \binom{q}{i-1}) = O(|E| \, 2^q)$ time.

For what concern space, the table M, as we already said, has a total of $q \times |V|$ entries, each of which contains at most $\binom{q}{i}$ pairs $\langle C, f \rangle$.

Each pair can be stored in $O(1)$ as they are simply 2 integer, we have a total size of $O(\Sigma_{i=1}^{q} \Sigma_{j=1}^{|V|} \binom{q}{i}) = O(|V| \, \Sigma_{i=1}^{q} \binom{q}{i}) = O(|V| \, 2^q)$.

Lemma 3.4. *Given an undirected graph $G = (V, E)$ random colored in $[q]$, where $q = O(\log |V|)$, Algorithm 5 (preprocess(G,q)) returns the dynamic programming table M of color coding in $O(|E| \, 2^q)$ time and $O(|V| \, 2^q)$ space.*

It is not difficult to modify the Algorithm 5 to list also the colorful q-grams, printing $L(\pi)$ for each colorful q-path π. This makes the algorithms inefficient, indeed we still have to face with the problem that $\mathcal{L} \sim \Sigma^q$.

So we will pass to the next step.

query(A, B): Sampling and sketching colorful paths

Now using the color coding table M, and given two set of nodes A, B, we want to approximate the values of $BC(A, B)$ and $FJ(A, B)$.

As already said, we cannot explore all the colorful q-grams, so our idea is to construct a sample of $\mathcal{L}$, without explicitly calculate it, by sampling r q-paths from M, where $r < |\mathcal{L}|$ is a user-selectable parameter.

We will use as a sample r q-paths without repetition. This can be also seen as a bottom-r sketch of all the q-paths.

Our algorithm for QUERY(A, B) consist of three phases as follows:

- Compute a suitable sketch $W \subset \mathcal{L}$ such that $\tau = |W|$ is at most r, by sampling r colorful q-paths using M and setting W as the sets of the q-grams of these paths.
- Compute $f_A[x]$, $f_B[x]$ for each $x \in W$.
- Approximate $BC(A, B)$ as $BC_W(A, B)$ and $FJ(A, B)$ as $FJ_W(A, B)$.

Where $BC_W(A, B)$ and $FJ_W(A, B)$ are defined as:

$$BC_W(A, B) = \frac{2 \times \Sigma_{x \in W} \min(f_A[x], f_B[x])}{\Sigma_{x \in W} f_A[x] + f_B[x]} \tag{3.1}$$

$$FJ_W(A, B) = \frac{\Sigma_{x \in W} \min(f_A[x], f_B[x])}{\Sigma_{x \in W} f_{A \cup B}[x]} \tag{3.2}$$

Phase 1: Colorful sampler

Sampling uniformly using a dynamic programming approach is a topic already covered in the literature, e.g. Martin Dyer in [12] or Eric Vigoda in [18]. In particular, we are interested in sampling τ q-grams from colorful q-paths, leading to nodes belonging to X, using the color coding table M.

As we are dealing with weighted sets (where the weights are frequencies), the sample must depend on the frequencies of the q-grams ending in $x \in X$, as in the case of consistent weighted sampling, where more frequent q-grams need to be sampled more often.

As we do not know a priori the frequency of q-grams before sampling, we sample q-paths uniformly at random so that the probability of getting a q-gram is proportional to the number of paths having that q-gram, i.e. its frequency. The uniform sampling of paths can be done by looking at the frequencies of colorful q-paths in M. We know that the number of colorful q-paths ending in v is $M_{q,v}([q])$, hence we extract the starting node $x \in X$ of our weighted random q-paths with a probability:

$$p_X(v) = \frac{M_{q,v}([q])}{\Sigma_{x \in X} M_{q,x}([q])} \tag{3.3}$$

Then we generate a random q-path by scanning the color coding table M backward from $q - 1$ to 1, choosing nodes with a probability similar to $p_X(v)$ (except that during step i we look at row i and in the complementary of the color set of the current i-path).

We define out sampling algorithm as shown in Algorithm 6.

Algorithm 6: COLORFUL-SAMPLER

Input : X = multiset of nodes from graph G
M = color coding table for G
τ = number of colorful paths to sample.
Output: W = random sample composed by q-grams of q-paths leading to $x \in X$

$R \leftarrow \{\}$
parallel for $j \in [r]$ **do**
  $u \leftarrow$ randomly chosen $v \in X$ with probability $p_v = \frac{M_{q,v}([q])}{\sum_{z \in X} M_{q,z}([q])}$
  $\pi \leftarrow$ RANDOM-PATH-TO(u)
  if $\pi \notin R$ **then** $R \leftarrow R \cup \{\pi\}$
  else $j \leftarrow j - 1$ //repeat the step
return $W = \{L(\pi) : \pi \in R\}$

The procedure RANDOM-PATH-TO is defined in Algorithm 7.

Algorithm 7: RANDOM-PATH-TO

Input : M = color coding table for G
u = leading node of the path
Output: π = random colorful path

$P \leftarrow \langle u \rangle$
$D \leftarrow [q] \setminus \{\chi(u)\}$
for $i \in \{q-1, \ldots, 1\}$ **do**
  $u \leftarrow$ randomly chosen $v \in N(u)$ with probability $p_v = \frac{M_{i,v}(D)}{\sum_{z \in N(u)} M_{i,z}(D)}$
  $P \leftarrow u \cdot P$
  $D \leftarrow D \setminus \{\chi(u)\}$
return P

Note that the method RANDOM-PATH-TO always finds a colorful q-path, as at each step we choose nodes only among the ones that will lead to a colorful q-path (i.e. the probability p_v is 0 for nodes that don't lead to a colorful q-path). This property is guaranteed by the way the color coding table M is generated by Algorithm 5.

Lemma 3.5. *For any multiset of nodes X, Algorithm 6 returns a random sample $W \subset |\Sigma^q|$ with a complexity of $O(rq)$ both in time and space, where $q = O(\log |V|)$ and $r < |\mathcal{L}| \leq |\Sigma|^q$.*

Phase 2: Frequency count

Now that we have a sample W, of suitable size, composed by q-grams, we are interested, for a multiset of nodes X, in calculating $f_X[x]$ for each $x \in W$. Algorithm 8, for steps $i = 1, 2, \ldots, q$, proceeds by expanding in BFS order only the i-paths ending in a node $u \in X$ and having i-grams that are suffixes of W (this operation can be made more space efficient by using tries or by performing a binary search in a set of strings).

We maintain a multiset T of these i-grams, each represented by a triple $\langle z, x, C\rangle$ to indicate that there exists a i-path starting from z and leading to a node $u \in X$ whose i-gram is x and its colorset is C (note that the same triple $\langle z, x, C\rangle$ can appear more times in T as there might exist multiple paths from z to u labeled with the same i-gram x).

Also in this case, considering that the computation for the i-gram depends only on the $(i-1)$-grams, we can parallelize the operations for the triple with same length i.

Algorithm 8: F-COUNT: exactly counting frequencies of sampled q-grams

Input : X = multiset of nodes from graph G
W = sample of its colorful q-grams
Output: $f_X[x]$ = frequency of each $x \in W$

$T \leftarrow [\,]$ // step $i = 1$
parallel foreach $u \in X$ *such that* $L(u)$ *appears at the end of a* q*-gram in* W **do**
  $T \leftarrow T \cup [\langle u, L(u), \{\chi(u)\}\rangle]$
for $i \in \{2, 3, \ldots, q\}$ **do**
  $T' \leftarrow [\,]$
  parallel foreach $\langle z, x, C\rangle \in T$ **do**
    foreach $v \in N(z)$ *such that* $\chi(v) \notin C$ **do**
      if $L(v) \cdot x$ *is a suffix of a* q*-gram in* W **then**
        $T' \leftarrow T' \cup [\langle v, L(v) \cdot x, C \cup \{\chi(v)\}\rangle]$
  $T \leftarrow T'$
$f_X \leftarrow (0, \ldots, 0)$
foreach $\langle z, x, C\rangle \in T$ **do** $f_X[x] \leftarrow f_X[x] + 1$
return f_X

It may happen that, in some big instance, Algorithm 8 explores many colorful paths as it expands the paths in $X \cup N^{<q}(X)$.

To alleviate this issue we present a modified version of the Algorithm 6, called F-SAMP, that estimate the value of $f_X[x]$ after having computed the sketch.

In Algorithms 9, as we already did in BRUTE-FORCE, we keep track of the leading nodes of all the q-paths, in this way we can use f_X to construct f_A, f_B and $f_{A\cup B}$. In addition, we estimate, with the lines 8 and 9, the value of f_X using the sampled q-paths R.

Of course, this speed up the computation time, on the other hand, as we will see in the next chapter, the accuracy will be affected and we will need a greater value of τ to have a better estimation of the similarity indices.

Algorithm 9: F-SAMP

Input : X = multiset of nodes from graph G
M = color coding table for G
τ = number of colorful paths to sample

Output: W = random sample set of colorful q-grams $x \in L(X)$ with probability $p_X(x)$
$f_X[\langle u_q, x\rangle]$ = frequency of each $x \in \mathcal{W}$ leading to u_q

1 $R \leftarrow \{\}$
2 **parallel for** $j \in [r]$ **do**
3 $\quad u \leftarrow$ randomly chosen $v \in X$ with probability $p_v = \frac{M_{q,v}([q])}{\sum_{z\in X} M_{q,z}([q])}$
4 $\quad \pi \leftarrow$ RANDOM-PATH-TO(u)
5 $\quad$ **if** $\pi \notin R$ **then** $R \leftarrow R \cup \{\pi\}$
6 $\quad$ **else** $j \leftarrow j - 1$ //repeat the step
7 $W \leftarrow \{L(\pi) : \pi \in R\}$
8 $f_X \leftarrow (0, \ldots, 0)$
9 **foreach** $\pi = \langle u_1, \ldots, u_q\rangle \in R$ **do** $f_X[\langle u_q, L(\pi)\rangle] \leftarrow f_X[\langle u_q, L(\pi)\rangle] + 1$
10 **return** $\langle W, f_X\rangle$

Lemma 3.6. *For any multiset of nodes X, Algorithm 9 (F-SAMP(X, M, r)) return a random sample $W \subset |\Sigma^q|$ and the map frequency $f_X[x]$ with a complexity of $O(rq)$ both in time and space, where $q = O(\log |V|)$ and $r < |\mathcal{L}| \leq |\Sigma|^q$.*

Phase 3: Indices estimation

Now that we have defined all the generic algorithms, we will use them to estimate both the Bray-Curtis index and the Frequency Jaccard Index.

The sampling algorithms, COLORFUL-SAMPLER and F-SAMP, can be used for estimating both the Bray-Curtis index ($X = A \uplus B$) and the Frequency Jaccard Index ($X = A \cup B$).

In this way, in the Bray-Curtis index, we give more weight of being extracted to the q-paths leading to $u \in A \cap B$, as in the multisets union ($\uplus$) we count the frequency of the elements that belong to the intersection twice.

We now present the four final algorithms for estimating both Bray-Curtis index and Frequency Jaccard index, using both F-COUNT and F-SAMP.

F-Count

For estimating the Bray-Curtis index, using the F-COUNT approach, we first create the sketch $\mathcal{W}$ by sampling r q-grams leading to $X = A \uplus B$, using the COLORFUL-SAMPLER algorithm. Then we calculate the exact values of $f_A[x]$ and $f_B[x]$, for each $x \in \mathcal{W}$, using the F-COUNT algorithm.

Finally, we estimate the real value $BC(A, B)$ with $BC_{\mathcal{W}}(A, B)$, i.e. the Bray-Curtis index restricted to the strings in $\mathcal{W}$ as defined in (3.1).

Algorithm 10: F-COUNT-BC

Input : A, B = sets of nodes from graph G
M = color coding table for G
r = number of colorful paths to sample

Output: $BC_W(A, B)$ = estimation of the Bray-Curtis index between A and B according to the F-COUNT algorithm

1 $\mathcal{W} \leftarrow$ COLORFUL-SAMPLER$(A \uplus B, M, r)$
2 $f_A \leftarrow$ F-COUNT(A, W)
3 $f_B \leftarrow$ F-COUNT(B, W)
4 **return** BRAY-CURTIS$(\mathcal{W}, f_A, f_B)$

For estimating the Frequency Jaccard index, we create the sketch $\mathcal{W}$ with $X = A \cup B$, always using the COLORFUL-SAMPLER algorithm.

Then the values of $f_A[x]$ and $f_B[x]$ are calculated in a different way, as we also want to calculate the value of $R = \Sigma_{x \in \mathcal{W}} f_{A \cup B}[x]$.

Using the property $|R| = \Sigma_{u \in A \cup B} \Sigma_{x \in \mathcal{W}} f_u[x]$ and $f_X[x] = \Sigma_{u \in X} f_u[x]$, we calculate, for each $u \in A \cup B$, the exact value f_u, which is the frequency of each q-gram leading to u, for q-grams belonging to the sampled dictionary $\mathcal{W}$.

We sum all the frequencies $f_u[x]$, for $x \in \mathcal{W}$, to R and finally merge f_u to f_A, if $u \in A$, and f_B, if $u \in B$.

Algorithm 11: F-COUNT-FJ

Input : A, B = sets of nodes from graph G
M = color coding table for G
r = number of colorful paths to sample

Output: $FJ_W(A, B)$ = estimation of the Frequency Jaccard index between A and B according to the F-COUNT algorithm

```
1  W ← Colorful-sampler(A ∪ B, M, r)
2  f_A ← (0, ..., 0)
3  f_B ← (0, ..., 0)
4  R ← 0
5  foreach u ∈ A ∪ B do
6      f_u ← F-count([u], W)
7      foreach x ∈ W do
8          R ← R + f_u[x]
9      if u ∈ A then
10         f_A ← f_A ∪ f_u
11     if u ∈ B then
12         f_B ← f_B ∪ f_u
13 return Frequency-Jaccard(W, f_A[x], f_B[x], R)
```

F-Samp

Estimating the two indices using the F-SAMP algorithm is easier than using F-COUNT, as F-SAMP compute the sketch $\mathcal{W}$ and the frequency map $f_X[x]$ at the same time.

To estimate the Bray-Curtis index we first call F-SAMP with $X = A \uplus B$, then we approximate the values of f_A and f_B from f_X by looking if the leading nodes of the q-paths belongs to A, B or both.

To estimate the Frequency Jaccard index we set $X = A \cup B$ and calculate R using the property $R = \Sigma_{x \in \mathcal{W}} f_X[x]$.

Algorithm 12: F-SAMP-BC

Input : A, B = sets of nodes from graph G
M = color coding table for G
r = number of colorful paths to sample
Output: $BC_W(A, B)$ = estimation of the Bray-Curtis index between A and B according to the F-SAMP algorithm

```
1 ⟨W, f_X⟩ ← F-SAMP(A ⊎ B, M, r)
2 f_A ← (0, ..., 0)
3 f_B ← (0, ..., 0)
4 foreach ⟨u, x⟩ ∈ f_X do
5     if u ∈ A then f_A[x] ← f_A[x] + f_X[⟨u, x⟩]
6     if u ∈ B then f_B[x] ← f_B[x] + f_X[⟨u, x⟩]
7 return Bray-Curtis(W, f_A, f_B)
```

Algorithm 13: F-SAMP-FJ

Input : A, B = sets of nodes from graph G
M = color coding table for G
r = number of colorful paths to sample
Output: $FJ_W(A, B)$ = estimation of the Frequency Jaccard index between A and B according to the F-SAMP algorithm

```
1 ⟨W, f_X⟩ ← F-SAMP(A ∪ B, M, r)
2 f_A ← (0, ..., 0)
3 f_B ← (0, ..., 0)
4 R ← 0
5 foreach ⟨u, x⟩ ∈ f_X do
6     R ← R + f_X[⟨u, x⟩]
7     if u ∈ A then f_A[x] ← f_A[x] + f_X[⟨u, x⟩]
8     if u ∈ B then f_B[x] ← f_B[x] + f_X[⟨u, x⟩];
9 return Frequency-Jaccard(W, f_A, f_B, R)
```

Conclusion

We have shown how to estimate the two Bray-Curtis and Frequency Jaccard similarity indices using the two approaches F-COUNT and F-SAMP. In particular, as demonstrated in [10], for F-COUNT, both $BC_W(A, B)$ and $FJ_W(A, B)$ are, respectively, unbiased estimators for $BC(A, B)$ and $FJ(A, B)$, meaning that $BC(A, B) = \mathbb{E}[BC_W(A, B)]$ and $FJ(A, B) = \mathbb{E}[FJ_W(A, B)]$ for every possible choice of $|W| = 1$. On the other hand, it has been proved that the estimation done by F-SAMP for $f_A[x]$ and $f_B[x]$, for each $x \in W$, is unbiased.

3.4 Baseline algorithm

In order to validate the effectiveness of our approaches, we compare the previously seen algorithms against a naive randomized approach, which is the following baseline algorithm BASE that find random paths by simply performing random walks.

Algorithm 14: BASE, the baseline sampler

Input : X = array of nodes from graph G
r = number of paths to sample
Output: W = dictionary of q-grams sampled
$f_X[x]$ = frequency of each $x \in W$, where W = naive random sample multiset of q-grams for X.

```
1  R ← {}
2  parallel for j ∈ [r] do
3      u ← randomly chosen v ∈ X with uniform probability
4      P ← NAIVE-RANDOM-PATH-TO(u)
5      if P ≠ null and P ∉ R then R ← R ∪ {P}
6      else j ← j − 1    //repeat the step
7  W ← [L(P) : P ∈ R]
8  f_X ← (0, ..., 0)
9  foreach x ∈ W do  f_X[x] ← f_X[x] + 1
10 return ⟨W, f_X⟩
```

And the algorithm NAIVE-RANDOM-PATH-TO:

Algorithm 15: NAIVE-RANDOM-PATH-TO

Input : u = leading node of the path
Output: π = random q-path leading to u or `null`

```
1  π ← ⟨u⟩
2  for i ∈ {q − 1, ..., 1} do
3      if N(u) \ π = ∅ then return null
4      u ← randomly chosen v ∈ N(u) \ π with uniform prob.
5      π ← u · π
6  return π
```

Note that, because this is a naive approach, the NAIVE-RANDOM-PATH-TO may fail to find a q-path leading to u as it goes to explore dead-end paths.

Also in this case we estimate $BC(A, C)$ using $X = A \uplus B$ and $FJ(A, B)$ using $X = A \cup B$ and $|R| = \Sigma_{x \in W} f_X[x]$.

Chapter 4

Project development

To confirm the validity, both in terms of correctness and performance, of our algorithms we implemented all the procedures previously illustrated. The most important parts of the code can be found in the appendix of this thesis.

4.1 Implementation choices and steps

The algorithms have been implemented using the C++ programming language, as it provides good performance in practice and a lot of well implemented data structures in the Standard Template Library.

We have first implemented the BRUTE algorithm as it was the simpler and gave us the correct answers. Then, we have implemented the three algorithms F-CONT, F-SAMP, BASE, checking whether some practical test gave us some reasonable values. After making sure that every algorithm works correctly, we started to parallelize them.

The parallelization has been implemented using the OpenMP API [5], which defines a simple and flexible interface for developing parallel applications, in particular, we used it to manage the parallel for-loops and the critical sections.

To make the tests repeatable we used random generators with fixed seed, the subgraphs A and B were generated in three different ways: two random and independent subsets of V, two connected components (generated by choosing two random nodes and then expanding them through a BFS), two ego-networks.

All the code was written in a modular and highly customizable way in order to better test the various algorithms. In the results we will explicitly show the parameter used to execute the tests.

4.2 Tuning the parameters in practice

The problem can be applied to a lot of context. That is why it is very important to choose the right domains for the values of the V, E, L, Σ, q:

- V are the objects we want to model.
- E represent the set of interactions, two vertices are connected if there exists a relation between them.
- L and Σ are the categories that partition V, $|\Sigma|$ should not be too high or too low, note that if $|\Sigma| = 1$ the labeling is useless as V is not really partitioned.
- q should be low as $N^{<q}(u)$ could be a large portion of G, (e.g. in Facebook for $q \simeq 4$ we have $N^{<q}(u) \simeq G$) [4].

4.3 Dataset

For the experiments we used two different kinds of dataset, a small one so we can easily brute-force the real indices and compare the relative errors, and a big one in order to benchmark the performance of the different approaches on a real world complex network.

NetInf This graph represents the flow of information on the web among blogs and news websites. The graph was computed by the *NetInf* approach, as part of the *SNAP* project [17], by tracking cascades of information diffusion to reconstruct "who copies who" relationships.

- V is the set of blog or news website, $|V| = 854$.
- E, each website is connected to those who frequently copy their content, $|E| = 3824$.
- Σ is the set of ranking classes of websites (0 top 4%, 1 next 15%, 2 next 30%, 3 last 51%), $|\Sigma| = 4$.
- L, each website is labeled according to its importance, using Amazon's Alexa ranking [1].

Considered query: compute the similarity of two websites a and b or two sets of websites.

IMDb In this graph, taken from the *Internet Movie Database* [14] we have:

- V is the set of all movies in *IMDb*, $|V| = 1\,060\,209$.
- E, two movies are connected if their casts share at least one actor, $|E| = 288\,008\,472$.
- Σ is the set of movies genre, $|\Sigma| = 36$.
- L, each movie is labeled with its principal genre.

Considered query: similarity of actors' ego-networks. Given two actors a and b, let A and B be their ego-networks, i.e., the sets of nodes corresponding to movies in which respectively a and b starred, compute the similarity of A and B.

The way we generate the IMDb graph is an example of collaboration graph and is known in literature as Co-stardom network.

Another famous example is the collaboration graph of mathematicians, where two mathematicians are connected if they have co-authored a paper. This collaboration graph is also known as Erdős collaboration graph [3], in honor of the famous mathematician Paul Erdős, in this graph is defined also the *Erdős number* as the distance in term of collaboration between Paul Erdős and another person.

4.4 Experimental results

We describe the experimental evaluation for our approach. Our computing platform is a machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40GHz, 24 virtual cores, 128 Gb RAM, running Ubuntu Linux v.4.4.0-22-generic. Code written in C++17, compiled with g++ v.5.4.1 and OpenMP 4.5.

To better analyze the different approaches described, we take several kinds of experiments. In each of them [1], all times are expressed in milliseconds.

An important fact to take into account is that we make large use of parallelization, so all the running time scale (approximately) linearly on the number of CPU cores used.

[1] Unless otherwise stated, all the results are the average of 100 identical experiments, in order to reduce the possible errors randomly caused by the machine.

Running time

In this experiment we compare the different running times, of all the parts, from all the algorithms.

First of all we test how much we can go up in Brute-Force with the value of q and the sample size, as this is our bottleneck to analyze the relative errors for the approximated methods.

Dataset	q	$\lvert A \cup B \rvert$	Brute-Force
NetInf	4	100	200
NetInf	4	200	420
NetInf	4	500	870
NetInf	5	100	1 206
NetInf	5	200	2 736
NetInf	5	500	6 080
NetInf	6	100	22 715
NetInf	6	200	49 828
NetInf	6	500	104 129

As expected, we can see that the running time for the bruteforce approach is linear in the size of $|A \cup B|$ and exponential in the value of q.

The second bottleneck for our algorithms is the preprocessing time for the dynamic programming table of color-coding, so that we test for both the dataset how can we go up with the value of q. We recall that, from initial assumptions, the value of q should not be too high.

Dataset	q	Color-Coding
NetInf	7	20
NetInf	9	80
NetInf	11	185
NetInf	13	340
NetInf	15	1 433
IMDb	3	48 220
IMDb	4	105 943
IMDb	5	241 224
IMDb	6	557 481

We can observe that, even in IMDb dataset, the value of q could go high as expected, always remaining under 10 minutes of running time.

To better understand these values, the official IMDb statistic [15] told us that, out of 1 837 357 actors analyzed, 1 579 193 ($\sim$ 86%) are distant $q = 3$ from the actor *Kevin Bacon* and 1 795 352 ($\sim$ 98%) are distant $q = 6$.

Finally, we test the running time for the different approaches for different value of q and number τ of q-paths sampled.

Dataset	q	$\lvert A\rvert$	$\lvert B\rvert$	τ	F-Count	F-Sample	Base
NetInf	3	100	100	1 000	20	4	2
NetInf	3	100	100	5 000	60	30	15
NetInf	5	100	100	1 000	2 682	426	3
NetInf	5	100	100	5 000	4 767	784	20
NetInf	7	100	100	100	5 455	4	2
NetInf	7	100	100	200	16 634	197	2
IMDb	3	10	10	100	5 035	66	1
IMDb	4	10	10	1000	/	2 829	14
IMDb	5	10	10	1000	/	4 739	20
IMDb	6	10	10	1000	/	9 783	36

The running time of base is always extremely low, unlike F-count, that, as we already anticipated, is not suitable for sampling too many q-paths. Instead the running time of F-samp turns out to be affordable for all the instances.

This is because both the F-samp and the base algorithms have a complexity of $O(rq)$ while F-count, that analyze the q-paths inside $A \cup N^{<q}(A)$ and $B \cup N^{<q}(B)$, could possibly traverse all the graph.

Relative error and variance

In this experiment we will compare, for increasing value of τ, how accurate and stable are the different algorithms using the NetInfo dataset.

The accuracy is calculated with the average of the relative error between the exact solution of brute and the analyzed algorithms F-count, F-samp or base, while the stability is calculated as the variance of the results, considering 100 experiments for each combination of parameters.

q	$\|A\|$	$\|B\|$	τ	ϵ_{BC}	VAR_{BC}	ϵ_{FJ}	VAR_{FJ}
3	100	100	10	0.02617187	0.00082663	0.02431909	0.000190515
3	100	100	100	0.02258048	0.00003059	0.02324100	0.000007628
3	100	100	1 000	0.03952676	0.00000070	0.04030510	0.000000132
4	100	100	10	0.03828302	0.00127922	0.03703453	0.000341645
4	100	100	100	0.01232044	0.00005457	0.00939392	0.000016680
4	100	100	1 000	0.01810665	0.00000240	0.02072427	0.000000750
5	100	100	10	0.04120389	0.00199562	0.04766193	0.000590912
5	100	100	100	0.01418216	0.00021613	0.01550921	0.000045352
5	100	100	1 000	0.02144092	0.00000647	0.02015720	0.000018239

Table 4.1: Relative error and variance of the F-COUNT approach

q	$\|A\|$	$\|B\|$	τ	ϵ_{BC}	VAR_{BC}	ϵ_{FJ}	VAR_{FJ}
3	100	100	10	0.53290243	0.02463258	0.64549929	0.01098586
3	100	100	100	0.26679417	0.00291718	0.35897713	0.00141635
3	100	100	1 000	0.05437719	0.00023471	0.12111130	0.00015040
4	100	100	10	0.56332930	0.03922519	0.71466000	0.01504646
4	100	100	100	0.42694364	0.00315346	0.58182255	0.00148827
4	100	100	1 000	0.17956068	0.00028600	0.26846896	0.00016087
5	100	100	10	0.56603667	0.03097334	0.72217070	0.01117576
5	100	100	100	0.60814392	0.00324602	0.73568322	0.00098974
5	100	100	1 000	0.37832023	0.00030943	0.49424173	0.00010248

Table 4.2: Relative error and variance of the F-SAMP approach

q	$\|A\|$	$\|B\|$	τ	ϵ_{BC}	VAR_{BC}	ϵ_{FJ}	VAR_{FJ}
3	100	100	10	0.79011542	0.023361286	0.83428722	0.00522323
3	100	100	100	0.38049732	0.003518397	0.40706656	0.00123490
3	100	100	1 000	0.10418507	0.000494349	0.10331303	0.00011619
4	100	100	10	0.89923793	0.013469196	0.90575658	0.00365555
4	100	100	100	0.64715221	0.004050390	0.65129934	0.00117385
4	100	100	1 000	0.23606907	0.000409090	0.24419065	0.00008983
5	100	100	10	0.91908669	0.009465890	0.95246880	0.00215748
5	100	100	100	0.82803890	0.001517523	0.83137675	0.00062314
5	100	100	1 000	0.44637460	0.000352671	0.46965620	0.00004772

Table 4.3: Relative error and variance of the BASE approach

From the three previous tables, we can clearly see that F-COUNT provides the best approximation for both the indices, with high precision and extremely low variance even for $R = 10$. The F-SAMP approach has a lower relative error compared to BASE, however here the variance between the two approaches seems to be nearly the same.

We further investigate the variance between F-SAMP and BASE, this time using IMDB as dataset, in order to study the stability in a real application.

				F-SAMP		BASE	
q	$\|A\|$	$\|B\|$	τ	VAR_{BC}	VAR_{FJ}	VAR_{BC}	VAR_{FJ}
3	100	100	1 000	0.00000971	0.00001004	0.00011746	0.00019368
4	100	100	1 000	0.00000114	0.00000736	0.00012097	0.00002175
5	100	100	1 000	0.00000594	0.00000085	0.00004424	0.00000624
6	100	100	1 000	0.00000109	0.00000020	0.00001050	0.00000154

Table 4.4: Variance of F-SAMP and BASE

Now we can see that, for both indices, the variance of F-SAMP is one, or in some case two, orders of magnitude fewer compared to BASE.

Finally, in the last test, to compare F-SAMP and BASE on IMDB, we show some real values of the two indices comparing the ego-networks of the famous comic duo Laurel and Hardy ($q = 3$, $R = 1\,000$, $|A| = 186$ and $|B| = 415$).

	F-SAMP		BASE	
	BC	FJ	BC	FJ
	0.928940	0.780303	0.821951	0.638258
	0.934292	0.759470	0.730479	0.549242
	0.929231	0.770833	0.764780	0.575758
	0.945752	0.787879	0.829152	0.657197
	0.933196	0.780303	0.758974	0.560606
	0.950655	0.793561	0.800000	0.621212
	0.941658	0.761364	0.759051	0.575758
	0.934292	0.776515	0.801980	0.613636
	0.933333	0.761364	0.799020	0.617424
	0.931282	0.768939	0.766917	0.579545
Mean	0.936167	0.774053	0.783230	0.598864
Variance	0.000055	0.000136	0.001005	0.001265

Table 4.5: Values of estimated BC and FJ setting A and B respectively the movie ego networks of Stan Laurel & Oliver Hardy

Fixed relative error

In this experiment we set the relative error and compare, for each approach, how many paths τ we need to sample in order to get this relative error.

		F-COUNT			F-SAMP			BASE		
q	ϵ	τ	T	VAR	τ	T	VAR	τ	T	VAR
3	0.20	2	1	0.0725	400	1	0.1194	420	1	0.1150
3	0.10	3	1	0.0692	1 000	1	0.0601	900	1	0.1338
3	0.05	4	1	0.0535	3 200	1	0.0273	1 500	1	0.1025
4	0.20	3	2	0.0677	1 300	1	0.1194	1 300	1	0.2424
4	0.10	5	4	0.0532	3 200	2	0.0992	2 500	2	0.1806
4	0.05	10	8	0.0518	8 000	4	0.0612	7 900	3	0.1081
5	0.20	5	6	0.0511	5 000	4	0.1678	6 000	3	0.2234
5	0.10	10	18	0.0370	20 000	12	0.0745	30 000	8	0.1234
5	0.05	20	58	0.0204	80 000	30	0.0376	/	/	/

Table 4.6: Dataset NETINF, $|A| = |B| = 100$

We can clearly see that F-COUNT performed very well, as it needs to sample a very little amount of q-paths to reach ϵ. Instead F-SAMP and BASE needed many more q-paths to reach the precision of F-COUNT. Note that in the last test BASE cannot reach the preestablished relative error.

Actors' ego-networks

In order to show a real application easy to understand, we compare some pairs of actors ego-networks (using F-COUNT algorithm with $q = 3$ and $R = 1\,000$):

Actor/actress	Actor/actress	BC index	FJ index
Stan Laurel	Oliver Hardy	0.936167	0.774053
Robert De Niro	Al Pacino	0.730935	0.231474
Woody Allen	Meryl Streep	0.556071	0.222857
Meryl Streep	Roberto Benigni	0.482909	0.160181

The values obtained are consistent with the theory for many reasons.

The Bray-Curtis index, as we already said, is always greater than the Frequency Jaccard and takes more into account the intersection.

The ego-networks of the famous comic duo Laurel and Hardy have a big intersection, this make the Bray-Curtis value very close to 1, however the Frequency-Jaccard is much smaller as Oliver Hardy starred in about 300 movie without Stan Laurel.

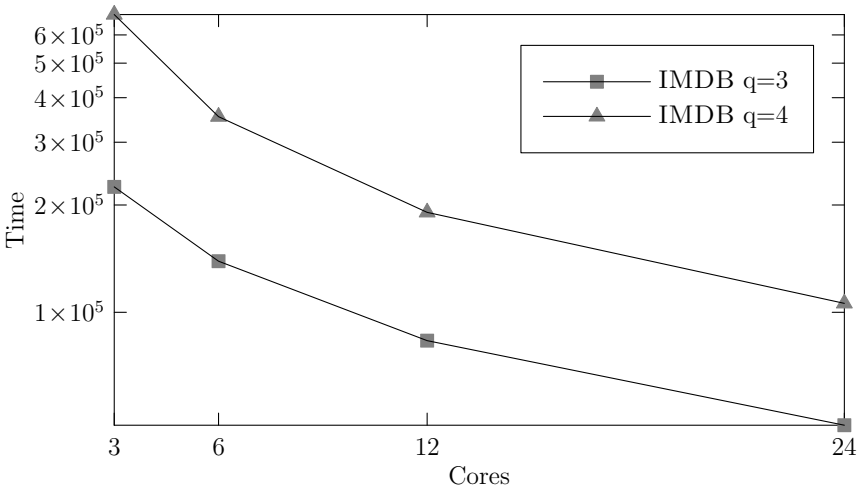
Concerning the couple Meryl Streep and Roberto Benigni, we have a big difference between the Bray-Curtis and the Frequency Jaccard. This can be due to the fact they are both famous actors (both won the Oscar Prize) who starred with a lot of other famous actors but they have not starred together.

Parallelization efficiency

As a last test, we show the parallelization efficiency in the computational time of the color coding table for different numbers of cores used.

		Core			
Dataset	q	24	12	6	3
NetInf	11	185	199	220	358
NetInf	13	340	503	948	1753
NetInf	15	1 433	2 235	4 296	7 654
IMDb	3	48 220	83 271	139 107	224 815
IMDb	4	105 943	190 719	353 856	684 342

In NetInf, as it is a small dataset, we see only a slight improvement. However, in IMDb times this improvement (approximately) doubles as the number of cores used is halved. We can see the times on IMDb in the following plot (logarithmic scale for time, linear scale for cores).



Chapter 5

Conclusion and future works

We presented randomized algorithms and data structures for approximate a subgraph similarity measure, which takes into account both the internal structure of subgraphs and their interface to the rest of the network.

The proposed algorithms, F-SAMP and F-COUNT, guarantee a good efficiency and approximation (as unbiased estimators) of the Bray-Curtis index and the Frequency Jaccard index, and show good practical performance compared to a less refined baseline sampler BASE. In particular the steady running time of F-SAMP on networks with hundreds of millions of edges suggests its usefulness as an estimator on very large networks.

A great advantage of the proposed algorithms is that they are highly parallelizable, which makes them suitable for analyzing massive datasets using today's datacenter with thousands of CPU cores running simultaneously.

As a future work, we note that the assumption that the graph is undirected with one label per node can be removed, and it would be interesting to study further similarity indexes that can be approximated with our algorithms.

Appendix A

Code snippets

The code written for this thesis can be found in the personal GitHub page[1].

Definitions

Global definitions of some utilities.

```
typedef long long ll;

typedef COLORSET uint32_t  // COLORSET is a bitset of 32 bit
unsigned int N, Q;         // Numer of nodes and length of paths
int color[N];              // Random coloring of nodes
char label[N];             // Labels of nodes
vector<int> G[N];          // Adjacency list for every node in G
map<COLORSET, ll> M[Q][N]; // Color coding table

// Get p-th bit of colorset n
bool getBit(COLORSET n, int p){ return ((n >> p) & 1) == 1; }

// Set p-th bit of colorset n to 1
COLORSET setBit(COLORSET n, int p){ return n |= 1 << p; }

// Reset p-th bit of colorset n to 0
COLORSET clearBit(COLORSET n, int p){ return n &= ~(1 << p); }

// Complementary colorset of n
COLORSET getCompl(COLORSET n){ return ((1 << q) - 1) & (~n); }
```

[1] https://github.com/GaspareG/SubgraphSimilarity

Similarity Indices

Algorithms 1 and 2

```
double BCW(set<string> W, map<string, ll> freqA,
                          map<string, ll> freqB) {
  ll num = 0ll;
  ll den = 0ll;
  for (string x : W) {
    ll fax = freqA[x];
    ll fbx = freqB[x];
    num += 2 * min(fax, fbx);
    den += fax + fbx;
  }
  return (double)num / (double)den;
}
```

```
double FJW(set<string> W, map<string, ll> freqA,
                          map<string, ll> freqB, ll R) {
  ll num = 0ll;
  for (string x : W) {
    ll fax = freqA[x];
    ll fbx = freqB[x];
    num += min(fax, fbx);
  }
  return (double)num / (double) R;
}
```

Color coding

Implementation of Algorithm 5

```
void preprocess() {
  #pragma omp parallel for schedule(guided)
  for (unsigned int u = 0; u < N; u++)
    M[1][u][setBit(0, color[u])] = 1ll;

  for (unsigned int i = 2; i <= q; i++) {
    #pragma omp parallel for schedule(guided)
    for (unsigned int u = 0; u < N; u++) {
      for (int v : G[u]) {
        for (auto d : M[i - 1][v]) {
          COLORSET s = d.first;
          ll f = d.second;
          if (getBit(s, color[u])) continue;
          ll fp = M[i][u][setBit(s, color[u])];
          M[i][u][setBit(s, color[u])] = f + fp;
        }
      }
    }
  }
}
```

Colorful sampling

Implementation of Algorithms 6 and 7

```
set<string> colorfulSampler(vector<int> X, int r) {
  set<string> W;
  set<vector<int>> R;
  vector<ll> freqX;
  for (int x : X)
    freqX.push_back(M[q][x][getCompl(0)]);
  discrete_distribution<int> distr(freqX.begin(), freqX.end());
  while (R.size() < (size_t)r) {
    int u = X[distr(eng)];
    vector<int> P = randomPathTo(u);
    if (R.find(P) == R.end()) R.insert(P);
  }
  for (auto r : R) {
    reverse(r.begin(), r.end());
    W.insert(L(r));
  }
  return W;
}
```

```
vector<int> randomPathTo(int u) {
  vector<int> P;
  P.push_back(u);
  COLORSET D = getCompl(setBit(0l, color[u]));
  for (int i = q - 1; i > 0; i--) {
    vector<ll> freq;
    for (int v : G[u])
      freq.push_back(M[i][v][D]);
    discrete_distribution<int> distr(freq.begin(), freq.end());
    #pragma omp critical
    {
      u = G[u][distr(eng)];
    }
    P.push_back(u);
    D = clearBit(D, color[u]);
  }
  reverse(P.begin(), P.end());
  return ret;
}
```

Frequency count

Implementation of Algorithm 8

```
map<string, ll> FCount(set<string> W, multiset<int> X) {
  set<string> WR;
  for (string w : W) {
    reverse(w.begin(), w.end());
    WR.insert(w);
  }

  vector<tuple<int, string, COLORSET>> old;
  for (int x : X)
    if (isPrefix(WR, string(&label[x], 1)))
      old.push_back(make_tuple(x, string(&label[x], 1),
                               setBit(0, color[x])));

  for (int i = q - 1; i > 0; i--) {
    vector<tuple<int, string, COLORSET>> current;
    current.clear();
    #pragma omp parallel for schedule(guided)
    for (int j = 0; j < (int)old.size(); j++) {
      auto o = old[j];
      int u = get<0>(o);
      string LP = get<1>(o);
      COLORSET CP = get<2>(o);
      for (int v : G[u]) {
        if (getBit(CP, color[v])) continue;
        COLORSET CPv = setBit(CP, color[v]);
        string LPv = LP + label[v];
        if (!isPrefix(WR, LPv)) continue;
        #pragma omp critical
        {
          current.push_back(make_tuple(v, LPv, CPv));
        }
      }
    }
    old = current;
  }

  map<string, ll> frequency;
  for (auto c : old) {
    string s = get<1>(c);
    reverse(s.begin(), s.end());
    frequency[s]++;
  }
  return frequency;
}
```

Frequency sampling

Implementation of Algorithm 9

```
map<pair<int, string>, ll> FSamp(vector<int> X, int r) {
  map<pair<int, string>, ll> W;
  set<vector<int>> R;
  vector<ll> freqX;
  freqX.clear();
  for (int x : X)
    freqX.push_back(M[q][x][getCompl(0)]);
  discrete_distribution<int> distr(freqX.begin(), freqX.end());
  while( R.size() < (size_t)r) {
    int rem = r - R.size();
    #pragma omp parallel for schedule(guided)
    for(int i=0; i<rem; i++) {
      int u;
      #pragma omp critical
      {
        u = X[distr(eng)];
      }
      vector<int> P = randomPathTo(u);
      #pragma omp critical
      {
        R.insert(P);
      }
    }
  }
  for (auto r : R) {
    reverse(r.begin(), r.end());
    W[make_pair(*r.begin(), L(r))]++;
  }
  return W;
}
```

F-Count Bray-Curtis

Implementation of Algorithm 10

```
double FCountBrayCurtis(set<int> A, set<int> B, int tau) {
  multiset<int> mA(A.begin(), A.end());
  multiset<int> mB(B.begin(), B.end());
  multiset<int> X(A.begin(), A.end());
  X.insert(B.begin(), B.end());
  set<string> W = colorfulSampler(X, tau);
  map<string, ll> freqA = FCount(W, mA);
  map<string, ll> freqB = FCount(W, mB);
  return BCW(W, freqA, freqB);
}
```

F-Count Frequency Jaccard

Implementation of Algorithm 11

```
double FCountFrequencyJaccard(set<int> A, set<int> B, int tau) {
  map<string, ll> freqA, freqB;
  set<int> AB(A.begin(), A.end());
  AB.insert(B.begin(), B.end());
  multiset<int> X(AB.begin(), AB.end());
  set<string> W = colorfulSampler(X, tau);
  ll R = 0;
  for(int x : X)
  {
    multiset<int> ma(&x,&x+1);
    map<string, ll> freqAB = processFrequency(W, ma);
    bool inA = A.find(x) != A.end();
    bool inB = B.find(x) != B.end();
    for (auto w : freqAB) {
      string s = w.first;
      ll f = w.second;
      R += f;
      if (inA) freqA[s] += f;
      if (inB) freqB[s] += f;
    }
  }
  return FJW(W, freqA, freqB, R);
}
```

F-Sample Bray-Curtis

Implementation of Algorithm 12

```
double FSampleBrayCurtis(set<int> A, set<int> B, int tau) {
  multiset<int> X(A.begin(), A.end());
  X.insert(B.begin(), B.end());
  map<pair<int, string>, ll> freqX = FSamp(X, tau);
  map<string, ll> freqA, freqB;
  set<string> W;
  for (auto w : freqX) {
    int u = w.first.first;
    int s = w.first.second;
    ll f = w.second;
    W.insert(s);
    if (A.find(u) != A.end()) freqA[s] += f;
    if (B.find(u) != B.end()) freqB[s] += f;
  }
  return BCW(W, freqA, freqB);
}
```

F-Sample Frequency Jaccard

Implementation of Algorithm 13

```
double FSampleFrequencyJaccard(set<int> A,
                               set<int> B, int tau) {
  set<int> AB(A.begin(), A.end());
  AB.insert(B.begin(), B.end());
  vector<int> X(AB.begin(), AB.end());
  map<pair<int, string>, ll>  freqX = FSample(X, tau);
  map<string, ll> freqA, freqB;
  set<string> W;
  ll R = 0;
  for (auto w : freqX) {
    int u = w.first.first;
    int s = w.first.second;
    int f = w.second;
    W.insert(s);
    R += f;
    if (A.find(u) != A.end()) freqA[s] += f;
    if (B.find(u) != B.end()) freqB[s] += f;
  }
  return FJW(W, freqA, freqB, R);
}
```

Bibliography

[1] Alexa. Website traffic, statistics and analytics. `https://www.alexa.com/siteinfo/`, Accessed October 2017.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995.

[3] Vladimir Batagelj and Andrej Mrvar. Some analyses of erdos collaboration graph. *Social Networks*, 22(2):173 – 186, 2000.

[4] Smriti Bhagat, Moira Burke, Carlos Diuk, Ismail Onur Filiz, and Sergey Edunov. Three and a half degrees of separation. *Facebook research*, February 2016.

[5] OpenMP Architecture Review Board. Openmp 4.5. `http://www.openmp.org/`, Accessed October 2017.

[6] Andrei Z. Broder. *Identifying and Filtering Near-Duplicate Documents*, pages 1–10. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[7] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 327–336, New York, NY, USA, 1998. ACM.

[8] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.*, 19(3-4):255–259, March 1998.

[9] Edith Cohen and Haim Kaplan. Summarizing data using bottom-k sketches. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 225–234, New York, NY, USA, 2007. ACM.

[10] Alessio Conti, Gaspare Ferraro, Roberto Grossi, Andrea Marino, Kunihiko Sadakanem, and Takeaki Uno. Subgraph similarity in real-world labeled networks. October 2017.

[11] Pawan Deshpande, Regina Barzilay, and David R Karger. Randomized decoding for selection-and-ordering problems. In *HLT-NAACL*, pages 444–451, 2007.

[12] Martin Dyer. Approximate counting by dynamic programming. pages 693–699, 2003.

[13] Michael R. Garey and David S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness /.* San Francisco : W. H. Freeman, 1979, 338 p. CALL NUMBER: QA76.6 .G35, 1979.

[14] IMDb. Imdb datasets. `http://www.imdb.com/interfaces/`, Accessed October 2017.

[15] IMDb. Imdb statistic. `http://www.imdb.com/stats`, Accessed October 2017.

[16] P. Legendre and L.F.J. Legendre. *Numerical Ecology.* Developments in Environmental Modelling. Elsevier Science, 1998.

[17] SNAP. Netinf. `http://snap.stanford.edu/netinf/`, Accessed October 2017.

[18] Eric Vigoda. Lecture notes on an fpras for knapsack. 2010.

Ringraziamenti

Desidero ringraziare chi, direttamente o indirettamente, ha contribuito nella realizzazione di questa tesi.

Un ringraziamento speciale ai miei relatori Roberto Grossi e Andrea Marino con il sostegno di Alessio Conti, per avermi pazientemente seguito, aiutato e consigliato.

A tutti i miei professori, fonti di grande ispirazione, che hanno contribuito, sotto ogni aspetto, alla mia crescita e formazione.

Ad Elena, per essermi sempre stata vicina e per avermi sempre supportato, sopportato e consigliato quando ne avevo più bisogno.

Ai miei genitori che, anche nelle piccole cose, mi sostengono ogni giorno e per tutti i sacrifici che hanno fatto per permettermi di arrivare ad oggi.