UNIVERSITÀ DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Triennale in Informatica

Tesi di Laurea

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 model in a natural way 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 $E \subset V \times V$ is the set of edges.

If two vertices $u, v \in V$ are connected by an edge they are called extreme 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, where not specified we will only deal with undirected graphs.

A sequence of nodes $v_1, v_2, \ldots, v_k$ is called path if $(v_i, v_{i+1}) \in E \ \forall i = 1, \ldots k-1$; a path is called simple if $v_i \neq v_j \ \forall i, j \ 1 \leq i < j \leq k$. 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 ($deg\ u = |N(u)|$).

With $N^{<k}(u)$ we indicate the set of vertex 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 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 a induced subgraph of G.

Note that an induced subgraph $G' = (V', E')$ can be uniquely identified by the set of its vertex 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 assign for every node v a symbol of the alphabet Σ. We call $L(u) \in \Sigma$ label of the node u.

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

In this thesis we mainly focus to analyze complex network: special graph with a non-trivial topology like random graph. Complex network occur in graphs modeling real system like social networks or computer networks and are characterized by a specific structural features:

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

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

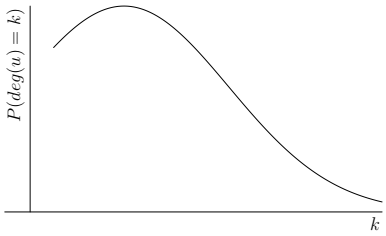


Figure 1.1: Degree distribution of a random network

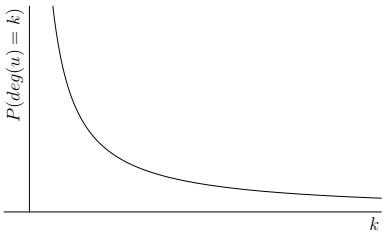


Figure 1.2: Degree distribution of a complex network

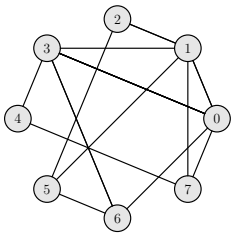


Figure 1.3: Random network

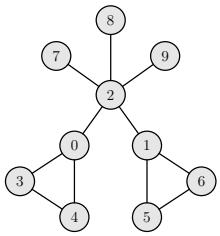


Figure 1.4: Complex network

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 $V_2 \cup N^{<q}(V_2)$.

Will discuss 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 the subgraph and the external 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 $|\Sigma|^q \gg |V|$ and, even worse, the number of simple paths can exponentially explode to $|V|^q$. For the simple reason 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 big network.

1.3 Practical applications

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 out object we want to modeling.
- E represent the set of interactions, two vertices are connected if exists a relation among them.
- L and Σ are the category that partition V, $|\Sigma|$ should not be too high or to 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 $N^{<q}(u) \simeq G$)[1].

Furthermore, we have to choose $G1$ and $G2$ in a way that similarity between two groups answer use some real question, like compare to each other two ego networks or two connected components.

Chapter 2

Basic tools

In this chapter we introduce some notion of similarity already existing in 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 set 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 set 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 set may be limiting as we consider only once the repeated values, we can easily extended the two previous definition to multiset.

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 multiset and curly brackets { } to indicate set.

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

Given two multiset $A = (a_1, \ldots, a_n)$ 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 $c_i = \max(a_i, b_i)$
- multiset union $C = A \uplus B = (c_1, \ldots, c_n)$ where $c_i = a_i + b_i$

Definition 2.5. Given two multiset $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ we define the **Frequency Jaccard index** as:

$$FJ(A, B) = \frac{\sum_{i=1}^{n} min(a_i, b_i)}{\sum_{i=1}^{n} max(a_i, b_i)} \tag{2.3}$$

Definition 2.6. Given two multiset $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ we define the **Bray-Curtis index on multiset** as:

$$BC(A, B) = \frac{2 \times \sum_{i=1}^{n} min(a_i, b_i)}{\sum_{i=1}^{n} a_i + b_i} \tag{2.4}$$

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.

Example 2.7. Given $A = (0, 2, 3, 1, 0, 3)$ and $B = (2, 0, 1, 3, 1, 2)$ we have:

$$FJ(A, B) = \frac{0 + 0 + 1 + 1 + 0 + 2}{2 + 2 + 3 + 3 + 1 + 2} = \frac{4}{13} \tag{2.5}$$

$$BC(A, B) = \frac{2 \times (0 + 0 + 1 + 1 + 0 + 2)}{2 + 2 + 4 + 4 + 1 + 4} = \frac{8}{17} \tag{2.6}$$

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

Definition 2.8. A function $f : A \times A \to \mathbb{R}^+$ is called metric (or simply distance) if satisfy 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 inequity: $f(x, z) \leq f(x, y) + f(y, z)$

2.2 Documents similarity

Documents similarity is an hot topic in Information Retrieval, as it can be seen as the problem of duplicate detection or, from another point of view, plagiarism detection.

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

Definition 2.9. 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-gram may also be called shingles.

Example 2.10. Given the document *"I live and study in Pisa"* all the possible 3-grams are: *"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, documents similarity is not an hard problem as we have only to perform union and intersection between set, or multiset.

Example 2.11. 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 Frequency Jaccard and Bray-Curtis are:

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

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

2.3 Graphs similarity

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

Definition 2.12. An isomorphism between two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is a bijective function from vertices of G to vertices of H that preserve the edge structure, i.e. $f : V_G \to V_H$ s.t. $(v, u) \in E_G \implies (f(v), f(u)) \in E_H$.

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 graph:

Definition 2.13. 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 a NP-complete problem, so the only way to solve it is by using heuristic or approximated methods.

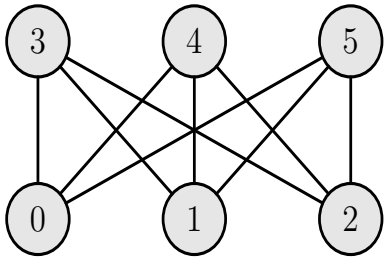


Figure 2.1: Graph G

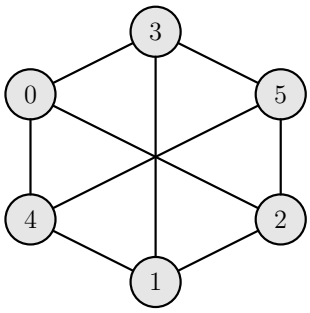


Figure 2.2: Graph H

Example 2.14. 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 $f(5) = 1$.

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 interest in analyzing G using the sequence of labels in its path.

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].

[1] Alternatively we can labeling edges in E instead of nodes in V without making too many changes in the following definitions, for 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-path, one for each orientation: one leading to u_q from u_1 and one leading to u_1 from u_q.

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-path both beginning and starting outside A, as we not considering them influential to define the similarity.

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

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

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

Definition 2.15. 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 2.16. 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:

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

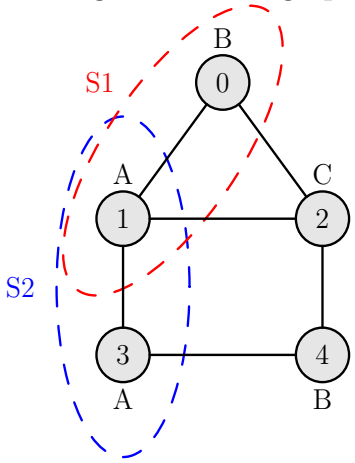
Let $\mathcal{L} = \{x \in \Sigma^q : x \in L(V)\} \subseteq \Sigma^q$ be the set of all distinct q-grams found in the q-paths of G.

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

In general $BC(A, B) \geq FJ(A, B)$. 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.

Example 2.17. 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:



$$L(S_1) = [aab, acb, baa, bca, bca, bcb, cab, cba]$$
$$L(S_2) = [baa, baa, bca, bca, caa, cba, cba]$$

Their union and intersection:

$$L(S_1) \cup L(S_2) = [aab, acb, baa, baa, bca, bca, bcb, caa, cab, cba, cba]$$

$$L(S_1) \cap L(S_2) = [baa, bca, bca, cba]$$

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 compute the exact similarity between two documents is an easy problem as we have only to compute the union and the intersection between the two set of shingles.

More difficult to manage becomes 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*. 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.

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 technique to compute the sketches of a document, for simplicity we assume that our document is composed by all numbers between 1 and n, in practice we will use a ranking function to define a sorting to the elements in documents.

Min-wise permutation

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

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

Note that using the min-wise permutation we can take multiple times the same number, as the permutation are independent.

Bottom-k sketches

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

Instead of using k random permutation $\pi_1, \ldots, \pi_k$ and take the minimum for each of them, like we did before in the min-wise permutation, we choose only one random permutation π and take the bottom k elements with lower value in permutation.

$$S_A = \{\min_1(\pi(A)), \ldots, \min_k(\pi(A))\}^3$$

Note that, unlike the min-wise permutation, in the bottom-k sketches we don't have repeated numbers as we take the numbers from a single permutation.

[3] With $\min_x$ we indicate the x-minimum element

Example 2.18. 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, Yuster and Zwick that efficiently finds simple path, cycles or many other small subgraphs using probabilistic algorithm. We will focus only to finds 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 color.

We restrict our attention to $q = O(log|V|)$ and denote with $\chi : V \to [q]$[4] the coloring function, where each node $u \in V$ have a color $\chi(u) \in [q]$.

After assigning a color to each node, we will focus to find only the colorful q-path. 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 reduce 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, when $q = 3$ we look only for the $\sim 22\%$ colorful q-paths and only for $\sim 4\%$ when $q = 5$.

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

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

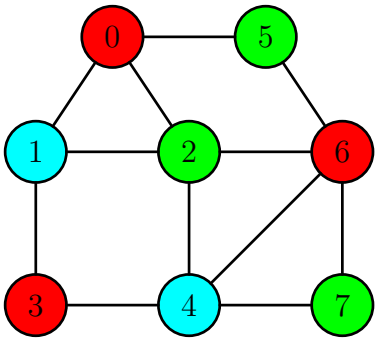
An interesting fact is that the method of color coding can be derandomized using a k-perfect hash family, that enumerating all the possible k-colorings of V. This makes the method exponential in the value of q, as if we set $q = |V|$ the problem became to find an Hamiltonian path, which is NP-complete.

To improve precision we can repeat the random coloring of nodes for e^q t times, so that success probability becomes $\geq 1 - e^{-t}$. Another way is to random coloring nodes using a greater 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 $[q']$).

In practice, choosing a simple random coloring is working pretty well on complex networks, so for don't add overhead to our algorithms we won't utilize the previous optimizations .

Example 2.19.

In this 3-colored graph out of 6 simple 3-path 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 algorithm to compute subgraphs similarity as previously defined: an exhaustive enumeration, two similar randomized approach using the tools described in the previous chapter and a naive randomized approach.

In the following algorithms, we will make use of parallel instruction, but we leave the specific programming choices and the comparison among the different approaches in the next chapter.

3.1 Indices calculation

Now we illustrate the procedures 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 iterate over all the strings in Σ^q we can restrict to $\mathcal{L} \subseteq \Sigma^q$, the set of all possible q-grams found in the q-paths of G.

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

A last note, we can observe that in the Frequency Jaccard index we don't have to explicitly calculate $f_{A\cup B}[x]$ and its summary, as the exact value of $R = \Sigma_{x\in\mathcal{W}} f_{A\cup B}[x]$ can be easily calculate from $f_A[x]$ and $f_B[x]$.

So we define the following procedures:

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
```

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

3.2 Naive approach

The naive approach consists in enumerate all the possible q-grams of simple q-paths leading to $u \in A \cup B$. This can be done by starting a modified DFS 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⟩ ← DFS(⟨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[x], f_B[x], R⟩
```

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 map $\Sigma^q \to \mathbb{N}$ but instead is a map $V \times \Sigma^q \to \mathbb{N}$, where the element in V is the leading node of the q-path associated to the q-gram.

The values of $FJ(A, B)$ and $BC(A, B)$ calculated using this method are exact, we will use it only to compare the precision of the following approaches as it found all the possibile $O(|\Sigma|^q)$ q-gram with a complexity of $O(|V|^q)$.

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

Algorithm 4: DFS

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
$f_u[x]$ = frequency of each $x \in \mathcal{W}$

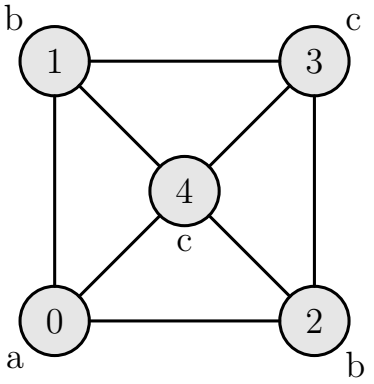
```
1  W ← ∅
2  f_u ← ∅
3  if |π| = q then
4  |   W ← W ∪ {L(π)}
5  |   f_u[⟨u_q, L(π)⟩] ← 1
6  else
7  |   foreach v ∈ N(u_1) \ π do
8  |   |   ⟨W_v, f_v⟩ ← DFS(⟨v⟩ · π, q)
9  |   |   W ← W ∪ W_v
10 |   |   f_u ← f_u ∪ f_v
11 return ⟨W, f_u⟩
```

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

As last thing, with $N(u_1) \setminus \pi$ we avoid to revisit the nodes already present in the path π, so we restrict to the simple q-paths.

Example 3.1.

The call to the method DFS$(0, 3)$ return:
$\mathcal{W} = \{abc, acb, acc\}$
$f_u[\langle 0, abc \rangle] = 4$ (0-1-3 0-1-4 0-2-3 0-2-4)
$f_u[\langle 0, acb \rangle] = 2$ (0-4-1 0-4-2)
$f_u[\langle 0, acc \rangle] = 1$ (0-4-3)



3.3 Efficient computation

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

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

As first thing 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 value of $|V|$ and with $q = O(\log n)$.

Second, instead of calculate the correct value of $f_X[\,]$ we computing 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 preprocessing the input graph $G = (V, E)$ given an integer $q > 0$, in particular we restrict our attention to $q = O(\log |V|)$.

Note that the preprocessing is independent from the labeling function L and from the subsets A, B to compare, as it depends only from the graph G and the value of q, so we can execute the preprocessing once and then reuse the dynamic programming 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 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 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 $|C| = i$ 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 fits 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 told, have a total of $q \times |V|$ entry, each of which contains at most $\binom{q}{i}$ pairs $\langle C, f \rangle$.
Each pairs 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 3.2. *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 dynamic programming 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 told, we can't explore all the colorful q-grams, so our idea is to construct a sketch 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 the method of the bottom-k sketch by taking, without repetition, the first r q-paths.

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

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

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

We are interested in sampling r colorful q-paths , leading to nodes belonging to X, using the color coding table M, sampled .

The number of the colorful q-paths in G is $|W| = \Sigma_{x \in V} M_{q,x}([q])$ of which $M_{q,i}([q])$ are leading to the node i. TODO

Algorithm 6: COLORFUL-SAMPLER

Input : X = array of nodes from graph G
M = color coding table for G
r = number of colorful paths to sample.

Output: W = random sample set of colorful q-grams $x \in L(X)$ with probability $p_X(x)$.

```
R ← {}
parallel for j ∈ [r] do
    u ← randomly chosen v ∈ X with probability p_v = M_{q,v}([q]) / Σ_{z∈X} M_{q,z}([q])
    π ← RANDOM-PATH-TO(u)
    if π ∉ R then R ← R ∪ {π}
    else j ← j − 1    //repeat the step
return W = {L(π) : π ∈ R}
```

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

Phase 2: Frequency count

TODO

Algorithm 8: f-count, exactly counting frequencies of sampled q-grams

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

```
1  T ← [ ]    // step i = 1
2  parallel foreach u ∈ X such that L(u) appears at the end of a q-gram in
     W do
3      T ← T ∪ [⟨u, L(u), {χ(u)}⟩]
4  for i ∈ {2, 3, ..., q} do
5      T' ← [ ]
6      parallel foreach ⟨z, x, C⟩ ∈ T do
7          foreach v ∈ N(z) such that χ(v) ∉ C do
8              if L(v) · x is a suffix of a q-gram in W then
9                  T' ← T' ∪ [⟨v, L(v) · x, C ∪ {χ(v)}⟩]    //critical s.
10     T ← T'
11 f_X ← (0, ..., 0)
12 foreach ⟨z, x, C⟩ ∈ T do  f_X[x] ← f_X[x] + 1
13 return f_X
```

Phase 3: Indices estimation

TODO

3.4 Baseline algorithm

In order to validate the effectiveness of our approach, we compare the previously seen algorithms against a naive randomized approach, the baseline algorithm BASE that find random paths in a simple way.

Algorithm 9: 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 10: 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 go 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 found in appendix of this thesis.

4.1 Implementation choices

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.

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

To make the tests repeatable we used random generators with fixed seed

4.2 Dataset

For the experiments we use two different kind 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 approach.

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 [**?**], 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 class 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.

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* 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.

4.3 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 experiment in each of them .[1]

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

Running time

In this experiment we compare the different running time, of all the parts, from all algorithms. Note that this is an important experiments, as in the real application time is crucial factor.

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.

q	$\lvert A \cup B \rvert$	Brute-Force
4	100	1 234
4	200	1 234
4	500	1 234
5	100	1 234
5	200	1 234
5	500	1 234
6	100	1 234
6	200	1 234
6	500	1 234

Table 4.1: Time in milliseconds

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

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

Dataset	q	Color-Coding
NetInf	7	1 234
NetInf	9	1 234
NetInf	11	1 234
NetInf	13	1 234
NetInf	15	1 234
IMDb	3	1 234
IMDb	4	1 234
IMDb	5	1 234
IMDb	6	1 234

Table 4.2: Time in milliseconds

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

Dataset	q	$\|A\|$	$\|B\|$	R	F-Count	F-Sample	Base
NetInf	7	100	100	1 000	1 000	1 000	1 000
NetInf	7	100	100	1 000	1 000	1 000	1 000
NetInf	7	100	100	1 000	1 000	1 000	1 000
NetInf	7	100	100	1 000	1 000	1 000	1 000
NetInf	7	100	100	1 000	1 000	1 000	1 000
NetInf	7	100	100	1 000	1 000	1 000	1 000
IMDb	3	100	100	1 000	1 000	1 000	1 000
IMDb	3	100	100	1 000	1 000	1 000	1 000
IMDb	3	100	100	1 000	1 000	1 000	1 000
IMDb	3	100	100	1 000	1 000	1 000	1 000

Table 4.3: Time in milliseconds

Relative error and variance

TODO

Dataset	q	Color-Coding	BruteForce
NetInf	3	1234	1234
NetInf	3	1234	1234
NetInf	3	1234	1234
NetInf	3	1234	1234
IMDb	3	1234	1234
IMDb	4	1234	1234
IMDb	5	1234	1234
IMDb	6	1234	1234

Table 4.4: Time in milliseconds

Fixed relative error

In this experiment we set the relative error ϵ_r and compare how many paths R we need to reach such relative error.

TODO

Dataset	q	Color-Coding	BruteForce
NetInf	3	1234	1234
NetInf	3	1234	1234
NetInf	3	1234	1234
NetInf	3	1234	1234
IMDb	3	1234	1234
IMDb	4	1234	1234
IMDb	5	1234	1234
IMDb	6	1234	1234

Table 4.5: Time in milliseconds

Fixed time precision

In this experiment we set the computing time and compare how many path.

TODO

Actors' ego-networks

As last test, in order to show a real application easy to understand, we compare some pairs of actors' ego-networks (using f-count algorithm with

Dataset	q	Color-Coding	BruteForce
NetInf	3	1234	1234
NetInf	3	1234	1234
NetInf	3	1234	1234
NetInf	3	1234	1234
IMDb	3	1234	1234
IMDb	4	1234	1234
IMDb	5	1234	1234
IMDb	6	1234	1234

Table 4.6: Time in milliseconds

$q = 3$ 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 respect the theory very faithfully for many reasons.

The Bray-Curtis index, as we already told, 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.

One last observation about the couple Meryl Streep and Roberto Benigni: we have a big difference between the Bray-Curtis and the Frequency Jaccard, this can be due from the fact they are both famous actors (both won the Oscar Prize) who starred with a lot of other famous actor but they haven't starred together.

Chapter 5

Conclusion and future works

We presented randomized algorithms and data structures for sketching subgraph similarity, which take 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 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. 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 analyze massive dataset using today's datacenter with thousands of cpu cores running simultaneously.

As future work, 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 sketched with our algorithms.

Appendix A

Code snippets

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

Snippet of common definition used in all the algorithms:

```
typedef long long ll;

// We define COLORSET as a bitset of 32 bit
typedef COLORSET uint32_t

// Number of nodes and number of edge
unsigned int N, E;

// Random coloring of nodes
int color[N];

// Labeled of nodes
char label[N];

// Adjacency list for every node in G
vector<int> G[N];

// Dynamic Programming table
map<COLORSET, ll> M[Q][N];
```

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

Color Coding

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

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

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

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

void ColorCoding() {
  #pragma omp parallel for schedule(static)
  for (int u = 0; u < N; u++)
          M[0][u][setBit(0, color[u])] = 1;
  for (int i = 1; i < q; i++) {
    #pragma omp parallel for schedule(static)
    for (int u = 0; u < V; u++) {
      for (int v : G[u]) {
        for (auto d : M[i-1][v]) {
          COLORSET s = d.first;
          long long f = d.second;
          if (!getBit(s, color[u]))
            M[i][u][setBit(s, color[u])] += f;
        }
      }
    }
  }
}
```

Colorful sampling

```
vector<int> randomPathTo(int u) {
        vector<int> P;
        P.push_back(u);
        COLORSET D = getCompl(setBit(0l, color[u]));

        for (int i = q - 2; 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());
                u = G[u][distr(eng)];
                P.push_back(u);
                D = clearBit(D, color[u]);
        }

        reverse(P.begin(), P.end());
        return P;
}

set<string> colorfulSampling(vector<int> X, int r) {
  set<string> W;
  set<vector<int>> R;
  vector<ll> freqX;
  for (int x : X)
  freqX.push_back(M[q-1][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)
W.insert(L(r));
return W;
}
```

Frequency count

```
map<string, ll> processFrequency(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(0ll, color[x])));

  for (int i = q - 1; i > 0; i--) {
    vector<tuple<int, string, COLORSET>> current;
    current.clear();
    #pragma omp parallel for schedule(static)
      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

```
map<pair<int, string>, ll>
  randomColorfulSamplePlus(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(0ll)]);
    discrete_distribution<int>
      distr(freqX.begin(), freqX.end());
    while (R.size() < (size_t)r) {
      int u = X[distribution(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[make_pair(*r.begin(), L(r))]++;
    }
    return W;
}
```

Similarity indices

```
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,
           long long 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;
}
```

Bibliography

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