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

List of Figures

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 a lot of information.

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 $v_1, v_2, \dots, v_k$ is called path if $(v_i, v_{i+1}) \in E \ \forall i = 1, \dots k-1$; 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:

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

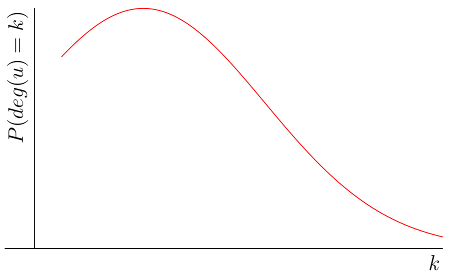

Figure 1.1: Degree distribution of a random network

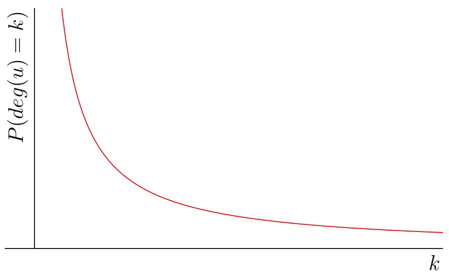

Figure 1.2: Degree distribution of a scale-free complex network

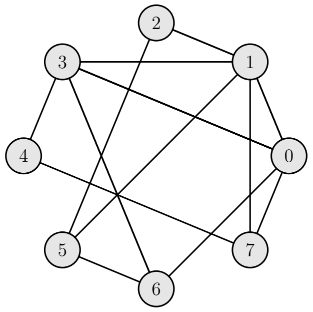

Figure 1.3: Random network with $|N| = 8$ and $|E| = 13$

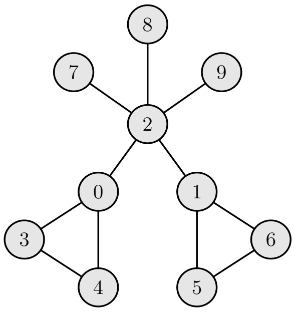

Figure 1.4: Complex network with $|N| = 10$ and $|E| = 11$

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 $N^{<q}(u) \simeq G$)[2].

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 Calculation of indices

Now we illustrate the procedures to calculate the 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 $\mathcal{W} = L(A) \cup L(B) \subseteq \Sigma^q$ 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.

```
contenuto...
```

```
contenuto...
```

Defined this procedures, in the next algorithm we focus only

2.2 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:

$$J(A,B) = \frac{|\{1,4,8\}|}{|\{1,2,3,4,5,6,7,8\}|} = \frac{3}{8} \tag{2.3}$$

$$BC(A,B) = \frac{2 \times |\{1,4,8\}|}{|\{1,3,4,5,7,8\}| + |\{1,2,4,6,8\}|} = \frac{6}{11} \tag{2.4}$$

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 $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$ we define the following operations:

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

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

$$FJ(A, B) = \frac{\sum\limits_{i=1}^{n} min(a_i, b_i)}{\sum\limits_{i=1}^{n} max(a_i, b_i)} \tag{2.5}$$

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

$$BC(A, B) = \frac{2 \times \sum\limits_{i=1}^{n} min(a_i, b_i)}{\sum\limits_{i=1}^{n} a_i + b_i} \tag{2.6}$$

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

$$J(A, B) = \frac{0 + 0 + 1 + 1 + 0 + 2}{2 + 2 + 3 + 3 + 1 + 2} = \frac{4}{13} \tag{2.7}$$

$$BC(A,B) = \frac{2 \times (0+0+1+1+0+2)}{2+2+4+4+1+4} = \frac{8}{17} \tag{2.8}$$

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

2.3 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 2.8. 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-gram may also be called shingles.

Example 2.9. 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 2.10. 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:

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

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

2.4 Graphs similarity

2.5 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 $L \rightarrow \Sigma$ 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.

[1] Alternatively we can labeling edges in E instead of nodes in V without making too many changes in the following definitions.

Now we present a little example to better understand.

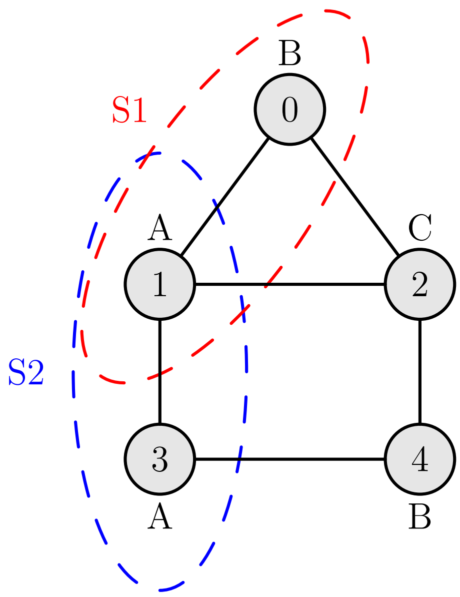

Figure 2.1: Labeled graph with $|V| = 5$ and $|E| = 6$

Example 2.11. We want to calculate the similarity between the two set $S_1 = \{0, 1\}$ and $S_2 = \{1, 3\}$ using their 3-grams:

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

So we have that:

$$FJ(S_1, S_2) = \frac{4}{11}$$

$$BC(S_1, S_2) = \frac{8}{15}$$

2.6 Sketches

Min-wise permutation

Bottom-k sketches

2.7 Color Coding

[1]

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 Naive approach

```
Algorithm 1: preprocess: BRUTE-FORCE
  Input  : $G = (V, E)$ undirected graph with $q$ random colors.
  Output: $(f_A[x], f_B[x])$ dynamic programming table for color coding.
  return $M$
```

3.2 Efficient computation

Color coding

Algorithm 2: *preprocess*: COLOR-CODING

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

parallel foreach $u \in V$ do $M_{1,u} = \langle \chi(u), 1 \rangle$
for $i \in \{2, 3, \ldots, q\}$ do
    parallel foreach $u \in V$ do
        foreach $v \in N(u)$ do
            foreach $\langle C, f \rangle \in M_{i-1,v}$ such that $\chi(u) \notin C$ do
                $f' \leftarrow M_{i,u}\,(C \cup \{\chi(u)\})$
                $M_{i,u} \leftarrow \langle C \cup \{\chi(u)\}, f' + f \rangle$
return $M$
```

Colorful sampling

Frequency count

Frequency sampling

Estimating similarity indices

3.3 Baseline algorithm

Chapter 4

Project development

In this chapter we describe

4.1 Implementation choices

4.2 Dataset

For the experiments we use two differents kind of dataset, a small one so we can easily brute-force the real indices and compare the relative error, and a big one in order to

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. Each node represents a blog or news website, and a website is connected to those who frequently copy their content. The graph contains 854 nodes and 3824 edges. We labelled websites according to their importance, using Amazon's Alexa ranking [?]: the labels correspond to respectively the websites ranked in the top 4%, the following 15%, the following 30%, and the remaining 51% (i.e. $|\Sigma| = 4$).

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* [?], nodes correspond to movies, and there is a link between two movies if their casts share at least one actor. The graph contains 1 060 209 movies (nodes) and 288 008 472 edges. Each movie is labeled with one of $|\Sigma| = 36$ genres.

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 version 4.4.0-22-generic. Code written in C++ and compiled with g++ version 5.4.1 with OpenMP.

Chapter 5

Conclusion and future works

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(0l

        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, L
                        }
                }
                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,
  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> distribution(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> fre
           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] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995.

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