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

Ai miei genitori
per non avermi tagliato i viveri

Contents

Chapter 1

Introduction

With the spread of Internet and more importantly of the social networks, efficient data analysis becomes increasingly important. Graphs are a powerful data structure that model in a natural way 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 $v_1, v_2, \ldots, v_k$ 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 modelling 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 (usualy $2 < \gamma < 3$), 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 $|N| = 100$ and $|E| = 1000$

Figure 1.4: Complex network with $|N| = 100$ and $|E| = 1000$

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 Pratical 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
- 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, note that if $|\Sigma| = 1$ the labeling is useless.
- 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 $N^{<q}(u) \simeq G$).

Furthermore, we have to choose $G1$ and $G2$, like ego networks or connected components.

Chapter 2

Basic tools

In this chapter we first give a definition of subgraphs similarity

2.1 Similarity indices

Definition 2.1. Given two set A and B we define the **Jaccard index** as the ratio between the size of intersection and 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:

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

We can easily extended the two previous definition to multiset:

Definition 2.3. Given two multiset $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ we define the **weighted 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 2.4. 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:

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

2.2 Documents similarity

...

2.3 Graphs similarity

2.4 Subgraphs similarity

...

2.5 Sketches

2.5.1 min-wise permutation

2.5.2 bottom-k sketches

2.6 Color Coding

Chapter 3

Computation of subgraph similarity

Algorithm 1: BRAY-CURTIS

Input : W a dictionary of strings
$f_A[W], f_B[W]$
Output: $BC(A, B)$ Bray-Curtis index

1 **return** M

3.1 Naive approach

Algorithm 2: *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.

1 **return** M

Algorithm 3: *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
```

3.2 Efficient computation

Color coding

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.

Appendix A

Code snippets

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

A.1 Color Coding

```
map<COLORSET, long long> M[Q][V];
void ColorCoding() {
  #pragma omp parallel for schedule(static, 1)
  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, 1)
    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;
        }
      }
    }
  }
}
```

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

A.2 Colorful sampling

```
vector<int> randomPathTo(int u) {
  list<int> P;
  P.push_front(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]);
    disc_distr<int> dist(freq.begin(), freq.end());
    u = G[u][dist(eng)];
    P.push_front(u);
    D = clearBit(D, color[u]);
  }
//  reverse(P.begin(), P.end());
  return vector<int>( begin(P), end(P));
}

set<string> colorfulSample(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)]);
  disc_distr<int> dist(freqX.begin(), freqX.end());
  while (R.size() < (size_t)r) {
    int u = X[dist(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;
}
```

A.3 Frequency count

```
map<string, ll> frequencyCount(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,

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

```
}
```

A.4 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;
}
```

A.5 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;
}
```