Finding densest subgraphs in linear time

Kanvaly FADIGA[1]

[1]Télécom paris, Ecole polytechnique

Finding dense region in a graph is an interesting problem in graph mining. We work on the implementation of the 2-approximation algorithm for the densest subgraph problem. In this report we present our implementation of this algorithm in C++ and the data strcuture that's allow us to have linear running time in the size of the input graph.

Graph | Community detection

1. Introduction

Problem definition: Given a graph $G = (V_G; E_G)$, find a subgraph H of G with maximum average degree density.

An algorithm for finding a 2-approximation solution to this problem is the following one :

Algorithm 1 2-approximation algorithm for the densest subgraph problem

Result: the densest graph encountered

```
1 H = G;
2 while (G contains at least one edge) do
3 |   let v be the node with minimum degree δ_G(v) in G;
4 |   remove v and all its edges from G;
5 |   if ρ(G) > ρ(H) then
6 |   |   H ← G
7 |   end
8 end
```

if at line 3 we can retrieve the node with minimum degree in $\Theta(1)$, then, this algorithm time complexity will be $\Theta(V + E)$. Indeed, line 2 will be in $\Theta(V)$ and line 4 till the end will go through all edges and delete them so $\Theta(E)$. So we have find a data structure that combine hashmap and linkedlist to efficiently do the step at line 3.

2. Our approach

Data structure.

We use unordered_map combine with linkedlist to track the minimum degree. given a unique hash by elements (which is our case), an unordered_map gives the advantage of retreiving and deleting a node in constant time $\Theta(1)$. linkedlist can delete and inserte a node in a constant time given one of it's node pointer. We will use the linkedlist to sort the degrees and use unordered_map of unordered_map to store node in each degree.

Let's take a look at figure 1. We want to delete node 1. this deletion will reduce the degree of node 2 and node 3 by moving them respectively to degree 1 and degree 3. this degree are not present in our data structure. So knowing the affected node old degrees (here 2 and 4), we will use the unordered_map connected to the linkedlist to go directly to the degree node 2 and 4 to insert before them the degree node 1 and 3 in $\Theta(1)$. Then, because the degree node 4 is empty we will delete it also in $\Theta(1)$. This opperation keep our linkedlist sorted so that at any time we can look at his head to retrieve the minimum degree. given the minimum degree we can use the unordered_map of unordered_map called degree_set to choose a node in $\Theta(1)$.

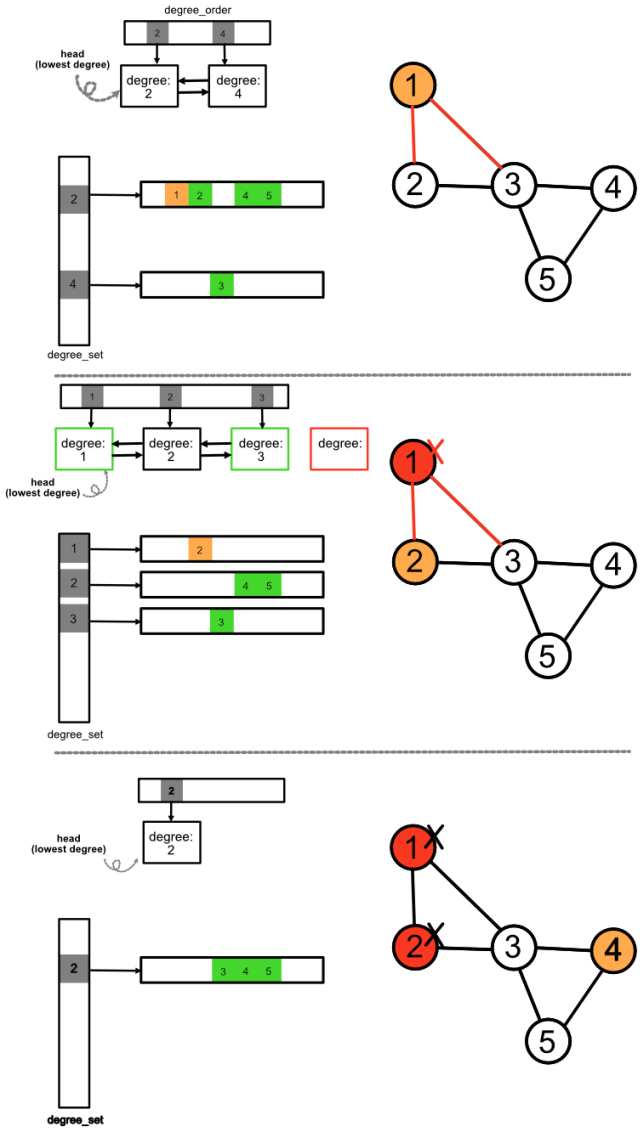

Fig. 1. our data structure example

Code short overview.

```
typedef unordered_map<int,
        unordered_map<int, Edge> > edge_list_t;
typedef unordered_map<int,
        unordered_map<int, bool> > degree_set_t;

struct Graph
{
    /* data */
    public:
    unordered_map<int, Node*> nodes;
    edge_list_t edges;
    int n_edges=0;

    // degree tracking datastructure
    degree_set_t degree_set;
    unordered_map<int, node_t*> degree_order;
    node_t* start = NULL;
    //..
};
```

Dataset.

We are going to evaluate our algorithm on the following datasets.

Dataset	Nodes (V)	Edges (E)	E+V
email-Eu-core	1,005	25,571	26, 576
cit-HepPh	34,546	421,578	456,124
email-EuAll	265,214	420,045	685,259
com-DBLP	317,080	1,049,866	1,366,946
com-Youtube	1,134,890	2,987,624	4,122,514

Table 1. Dataset for evaluation

3. Results

We made 10 runs for each dataset in order to estimate the average execution time.

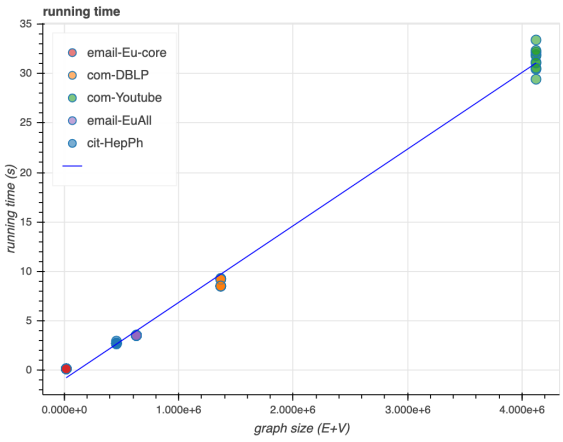

Fig. 2. the running time of the algorithm as a function of the input size

As we can see in figure 2, the running time increase linearly in the size of the input graph.

- Report the density of the subgraphs you found as well the number of nodes in each of the subgraphs.

Dataset	Nodes	Edges	Density
email-Eu-core	226	6230	27.566
cit-HepPh	2842	70879	24.94
email-EuAll	508	16725	32.923
com-DBLP	114	6441	56.5
com-Youtube	1867	85100	45.581

Table 2. Density repport

4. Conclusion

Our data structure work very well and as result we end up with linear time complexity in the graph size.