Graph Anonymization: *k*-Degree Anonymization

Sanchayan Bhunia (4849650)

Sahitya Reddy Bollavaram (4849759)

March 18, 2021

Instructors: Dr. Alessio Merlo and Mr. Davide Caputo

UNIVERSITÀ DEGLI STUDI
DI GENOVA

Road-map of the presentation

- Graph and it's Degree Sequence.
- Anonymity of a Graph.
- k-Degree Anonymity of a Graph.
    - Dynamic Programming Method of Anonymization.
        - Optimization of the search space.
        - Memorization Technique.
        - Memorization Technique on optimized search space.
        - Comparison through Experiment Results.
    - Greedy Method of Anonymization.
    - Greedy vs Dynamic Algorithms (Experiment Results).
        - Temporal Performance.
        - The Performance Ratio (Accuracy).
    - Graph Re-Construction.
    - Results and visualization.
- Conclusion

Graph

A Graph is a structure to represent a set of pair-wise related objects.

Each object can be represented by a **node** and the relation between pairs can be represented by an **edge** connecting them.

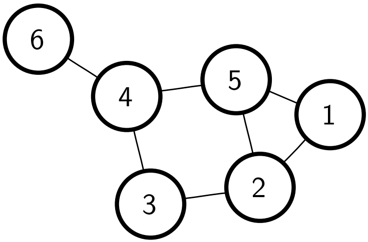
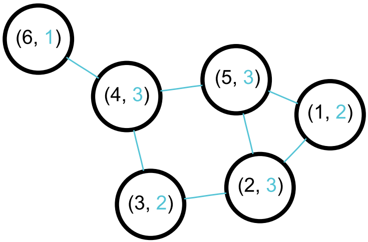


Figure: Simple Graph

Graph and Degree Sequence

A graph can be represented as a tuple by the index of the nodes and the number of edges connecting that node. For example,

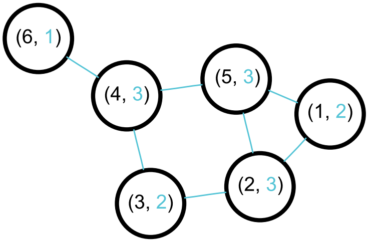


$$G = \{(v_1, e_1), (v_2, e_2), \ldots, (v_i, e_i)\} \tag{1}$$

where $v_i \in V$ and $e_i \in E$

Graph and Degree Sequence

A *degree* of a vertice (v) is the the number of edges, (e) it has.



The **Degree Sequence** (d_G) of a Graph (G) is just re-ordered set of *degrees* in a decreasing sequence. E.g. The degree sequence of our example graph is : $\{3, 3, 3, 2, 2, 1\}$.

The dimension of a degree sequence is same as the total number of nodes.

k-Degree Anonymity

Def^n: **For every node v in the graph G there exists alteast $k-1$ other nodes with the same degree as v.**

If the graph G has n nodes, the degree sequence (d_G) of G is an n dimensional vector with, $d_G(i) \geq d_G(j)\ \forall\ j > i$ where i and j are two the indices of the vector d_G.

The vector d_G is *k-anonymous* if every distinct value in d_G appears **atleast** k times.

The graph G corresponding to d_G will be called k-Degree anonymous *iff* d_G is *k-anonymous*.

Cost of Degree Anonymization

Given a graph G and its anonymized form $\hat{G}$ the cost function L_1 returns the total cost of turning d_G into $d_{\hat{G}}$.

$$L_1(d_G,\, d_{\hat{G}}) = \sum_{i=1}^{n} \left| d_{\hat{G}}(i) - d_G(i) \right| \tag{2}$$

For our example graph to be 3-**Degree anonymous**, we have to modify the degree sequence, $\{3, 3, 3, 2, 2, 1\}$ to $\{3, 3, 3, 2, 2, 2\}$. And Cost of anonymization,

$$L_1 = |3-3| + |3-3| + |3-3| + |2-2| + |2-2| + |2-1| = 1$$

k-Degree Anonymization Algorithms

k-Degree Anonymity can be achieved both by using **dynamic programming** and **greedy-swap algorithms** but anonymization cost L_1, might differ form one method to another.

Dynamic Programming

Let suppose, we have a degree sequence d of dimension n and by definition it is sorted. i.e.

$$d(1) \geq d(2) \geq ... \geq d(n) \tag{3}$$

Let, $Da(d[1,i])$ be the degree anonymization cost of the sub-sequence $d[1,i]$.

Also let, $I(d[i,j])$ be the anonymization cost when all the elements from $d(i), d(i+1), ..., d(j)$ are assigned the same value as $d(i)$. i.e.

$$I(d[i,j]) = \sum_{l=i}^{j} (d(i) - d(l)) \tag{4}$$

Condition 1: $i < 2k$

It is *impossible* to form two groups with distinct values.

E.g. for a 3-Degree anonymization of a sub-sequence $d[1,4] = \{3,3,2,2\}$ has to be $\{3,3,3,3\}$.

The cost of anonymization,

$$Da(d[1,i]) = I(d[1,i]) \tag{5}$$

Condition 2: $i \geq 2k$

$$Da(d[1,i]) = \min\left\{ \min_{k \leq t \leq i-k} \{Da(d[1,t]) + I(d[t+1,\ i-k])\}, I(d[1,i]) \right\}$$

When,

$i \geq 2k$ the degree sequence can be broken down in smaller sub-sequences recursively until the size of the sub-sequence follows **Condition 1**. For the rest of the sequence, we put them in the same group and calculate the total cost.

Algorithm Visualization

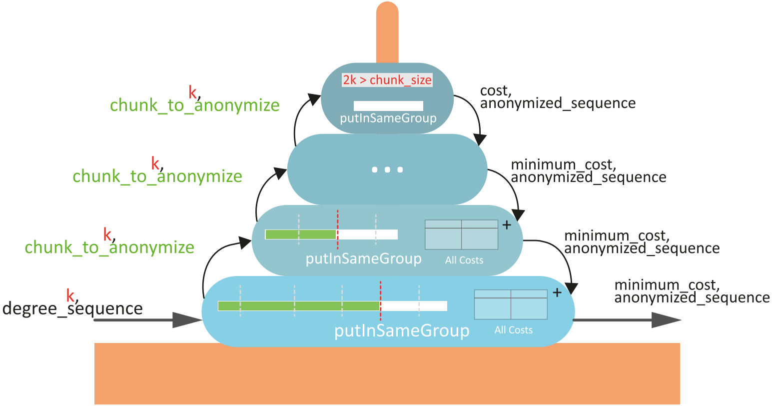


Figure: Recursion Heap

Time Complexity

We will generate degree sequence of various length and we'll plot time taken in order to anonymize them for differen values of k.

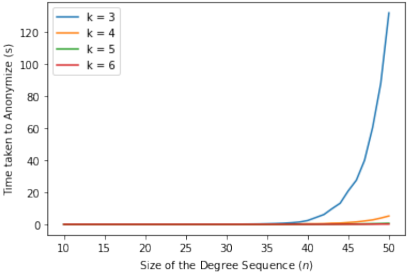


Figure: Time Complexity of Dynamic Programming

- Lower $k \longrightarrow$ higher time
- **Very Slow**

Optimizing the Range

- *No anonymous group* is larger than $2k - 1$.
- Anything larger can be broken down without increasing loss.
- Which essencially turns out that the preprocessing step $I(d[i, j])$ does not need to consider all of the combinations of (i, j) but only, $k \leq j - i + 1 \leq 2k - 1$.

Optimized Algorithm

For every i, we do not have to consider all t's in the range $k \leq t \leq i - k$ in Recursion, but only t's in the range $\max\{k, i - 2k + 1\} \leq t \leq i - k$.

$$Da(d[1,i]) =$$
$$\min\left\{ \min_{\max\{k, i-2k+1\} \leq t \leq i-k} \{Da(d[1,t]) + I(d[t+1, i-k])\}, I(d[1,i]) \right\}$$

Improved Running Time

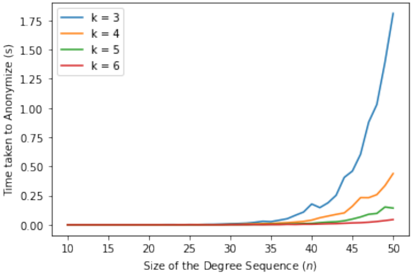


Figure: Optimized Dynamic Programming

- Took around 1.5 seconds to 3-degree anonymize a sequence of same length.
- Approximately 100 times faster than unoptimized algorithm.

Memorization Technique

- There is one more way to bring down the run-time of this algorithm, that is by *memorization*.
- This algorithm is dramatically faster than any of the Dynamic programing Algorithm mentioned in the paper(LT08).
- The memorized algorithm can also be **unoptimized** or **Range Optimized**.

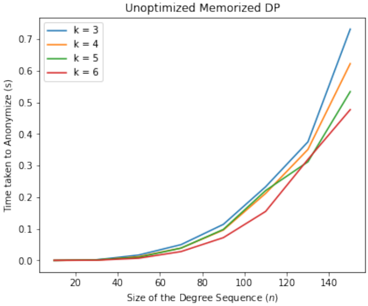
What to memorize?

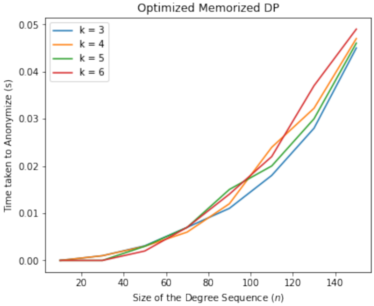
- From very close observation deep inside the recursion we have found that the pair (t, i) is repeated multiple number of times.
- So, we memorize the anonymized sequence and the anonymization cost for (t, i) pair.
- In the program we call it (chunk, number˙of˙nodes) for any specific layer in the heap.
- Then whenever in the program we encounter the same pair, we just search the anonymized sequence and cost in stead of computing them.

How to Implement

- We could do it by using 4 different lists $\longrightarrow$ one for t, one for i, one for the anonymized sequence and the last one for the cost.
- But this approach forces us to loop through two lists in order to find a matching pair which takes time.
- Rather, we can hash the pair (t, i) itself and use it as key to store the (anonymization cost, anonymized sequence) as values.
- Turns out that in Python we can use a single Dictionary to perform the entire prescription.

Memorized unoptimized vs Memorized optimized





- We can eliminate the dependence on k.
- The optimized version $10\times$ faster than the unoptimized one.
- Both of them are significantly faster than not using the memorization.

Summary so Far

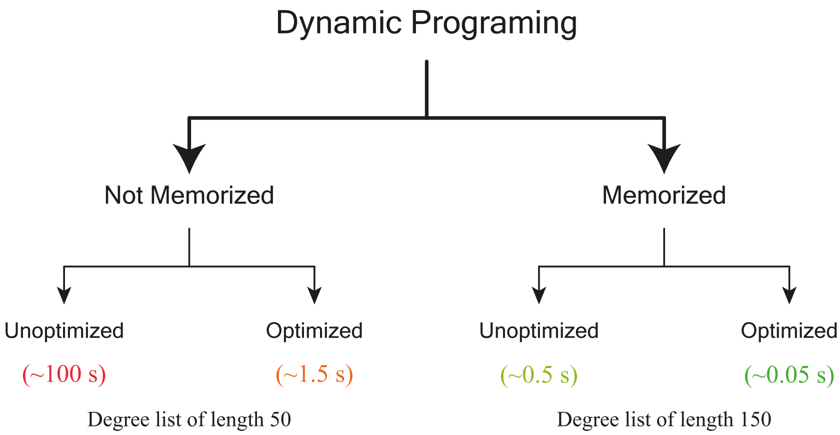
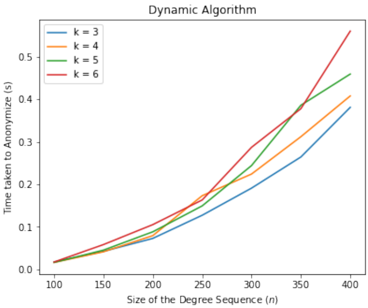


Figure: Time taken by various Dynamic Algorithms

Dynamic Programming vs Greedy Method



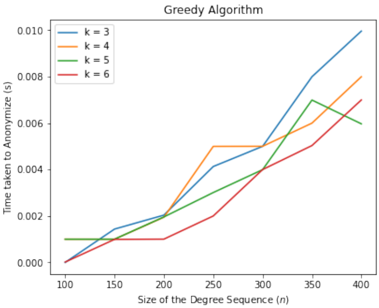


Figure: Running-time comparison

Performance Ratio (R)

Performance can be in terms of accuracy of the outcome as well.

In that case, the loss function L_1 is the key to measure such performance. Here we define an indicator called **Performance Ratio** (R),

$$R = \frac{L(\hat{d}_{Greedy} - d)}{L(\hat{d}_{Dyanmic} - d)} \tag{6}$$

Now, for the rest of the experiments, we will move from the synthetic data to two datasets found online "Quakers.csv" and "Netscience.gml"

Performance Ratio in two Datasets

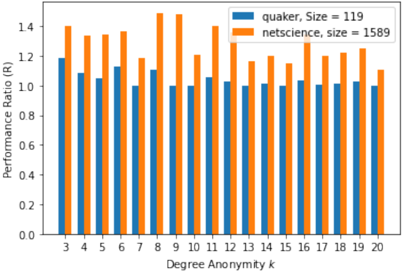


Figure: Caption

We can see that in most of the cases,

$$R1 \implies \frac{L(\hat{d}_{Greedy} - d)}{L(\hat{d}_{Dyanmic} - d)} > 1 \implies L(\hat{d}_{Greedy} - d) > L(\hat{d}_{Dyanmic} - d)$$

Graph Reconstruction I

All Degree sequences that we get after anonymization do not represent a valid graph.

So, after we have anonymized the Degree sequence it is necessary to reconstruct the graph.

Graph Reconstruction II: Quakers Data

Values of $k(3,20)$ For **Dynamic** Algorithm where graphs *cannot* be reconstructed are - $7, 10, 11, 12, 14, 15, 17, 18, 20$
Total: 9

and

Values of $k(3,20)$ For **Greedy** Algorithm where graphs *cannot* be reconstructed are - $3, 4, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20$
Total: 12

Values of k and R For **Both Algorithms** where graphs *can* be reconstructed are $(k : 5,\ R : 1.05)$; $(k : 9,\ R : 1.0)$; $(k : 13,\ R : 1.0)$; $(k : 19,\ R : 1.0289)$

Graph Reconstruction III: Netscience Data

Values of k $(3, 20)$ For Dynamic Algorithm where graphs *cannot* be reconstructed are - $4, 5, 10, 11, 15, 18, 19$
Total: 7

and

Values of k For Greedy Algorithm where graphs *cannot* be reconstructed are - $6, 7, 9, 10, 11, 12, 14, 15, 17, 18, 19, 20$
Total: 12

Values of k and R For Both Algorithms where graphs *can* be reconstructed are-
$(k : 3,\ R : 1.4)$; $(k : 8,\ R : 1.489)$; $(k : 13,\ R : 1.1616)$; $(k : 16,\ R : 1.3412)$

Graph Reconstruction IV: Observation

For **both of the data sets** it's the the Greedy algorithm where a graph reconstruction is not possible for more number of ks.

Which means **Dynamic Programming is better** than Greedy Algorithm in terms of performance.

Example: Reconstructed graph from Quaker Data

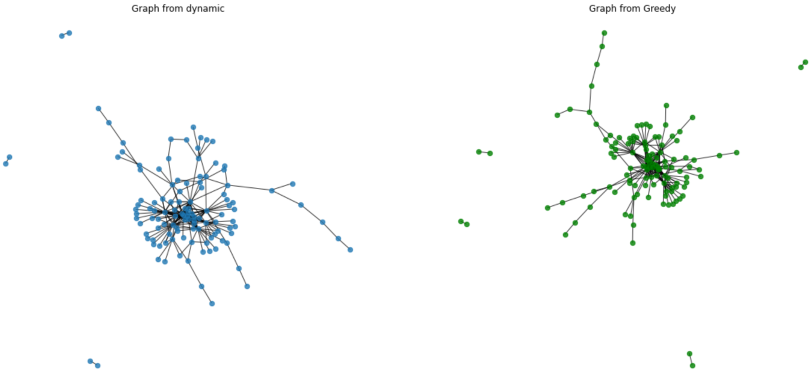


Figure: 9-Degree Anonymized Graph

Conclusion

- The Memorized Dynamic Programming algorithm is the only algorithm that is outside the article and we have created ourselves.
- Memorized one is about 500$\times$ faster than the optimized dynamic algorithm while producing 100% exact results.
- Time complexity of the memorized algorithm appears $O(n)$.
- Greedy Algorithm is the fastest.
- Dynamic algorithm has higher accuracy than Greedy algorithm.

References I

[LT08] Kun Liu and Evimaria Terzi, *Towards identity anonymization on graphs*, Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (New York, NY, USA), SIGMOD '08, Association for Computing Machinery, 2008, p. 93–106.

Thank You!