KILANI Al houceine — NAOUMI Salmane
al_houceine.kilani@ens-paris-saclay.fr — houcinekila@gmail.com — salmane.naoumi@ens-paris-saclay.fr — salmane.naoumi@gmail.com

1 Task 1 - Task 2

below the resulting list containing [source, target, and weight] for all the edges

```
[['method', 'solut', 2], ['method', 'system', 2], ['method', 'linear', 1], ['solut', 'system', 3], ['solut', 'linear', 1],
['system', 'linear', 2], ['solut', 'algebra', 1], ['system', 'algebra', 2], ['linear', 'algebra', 2], ['system', 'equat',
2], ['linear', 'equat', 2], ['algebra', 'equat', 2], ['linear', 'm-dimension', 2], ['algebra', 'm-dimension', 2],
['equat', 'm-dimension', 2], ['algebra', 'lambda', 2], ['equat', 'lambda', 2], ['m-dimension', 'lambda', 2], ['equat',
'system', 1], ['m-dimension', 'system', 1], ['lambda', 'system', 1], ['m-dimension', 'linear', 1], ['lambda', 'linear',
1], ['lambda', 'algebra', 1], ['equat', 'matric', 1], ['m-dimension', 'matric', 1], ['lambda', 'matric', 1], ['m-
dimension', 'method', 1], ['lambda', 'method', 1], ['matric', 'method', 1], ['lambda', 'solut', 1], ['matric', 'solut',
1], ['matric', 'system', 1], ['method', 'numer', 1], ['solut', 'numer', 1], ['system', 'numer', 1], ['numer', 'system',
1], ['system', 'special', 2], ['numer', 'special', 1], ['numer', 'kind', 1], ['system', 'kind', 1], ['special', 'kind',
1]]
```

Figure 1: Description of the graph built using a window size of 4

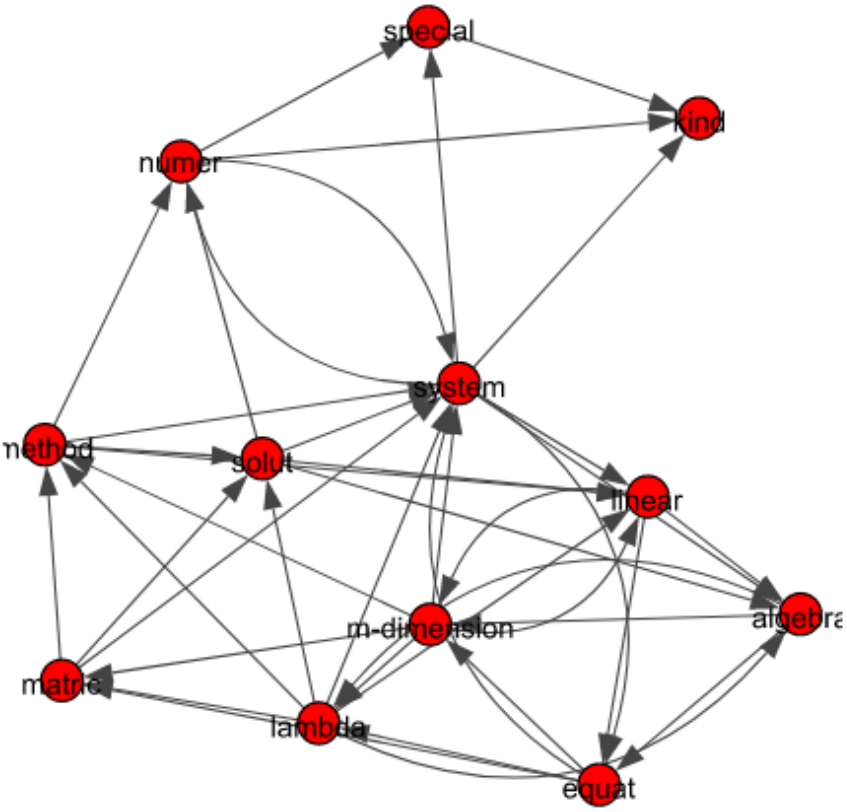

Figure 2: Network graph visualization

A quick double-check ensures us that the graph built corresponds indeed to the figure of the graph in the instruction sheet.

2 Question 1

The results of the evolution of the graph density are illustrated below :

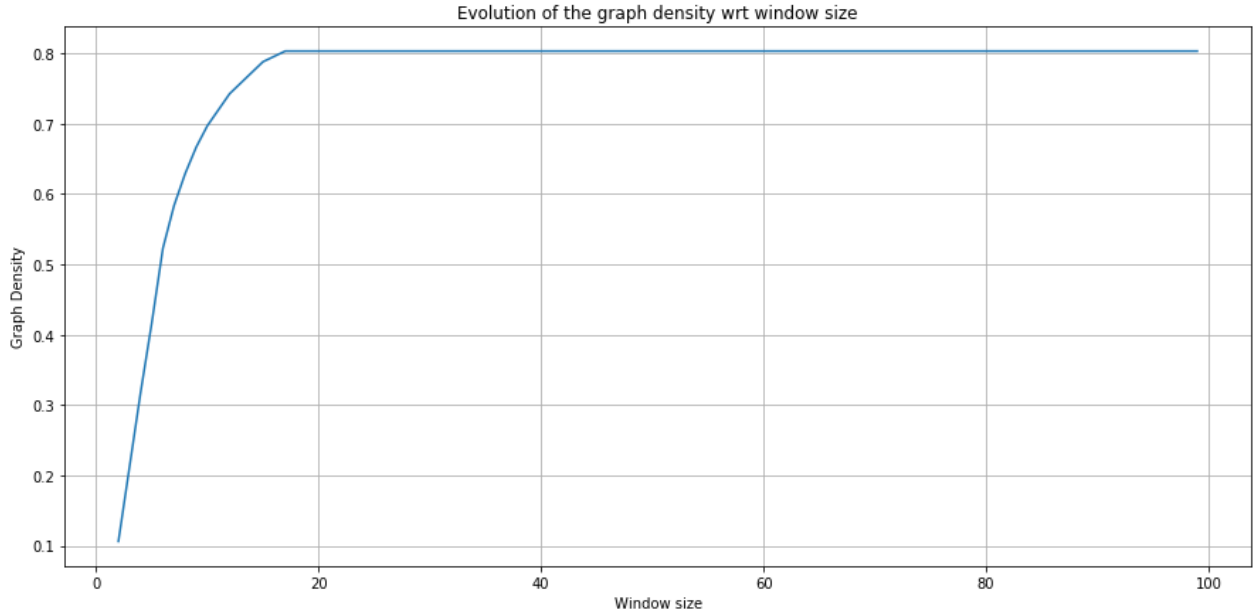

Figure 3: Impact of the window size on the graph density

Let the density be defined as : $\frac{|E|}{|V|(|V|-1)}$. Based on the distributional hypothesis of a sliding window:

- We observe that the density increases with the window size(which is normal as the number of edges increases)
- Then it stagnates at 80% (here when the window size reaches the value 17), i.e no more edges are created.
- Conclusion : the density of the graph keeps increasing until it stagnates.

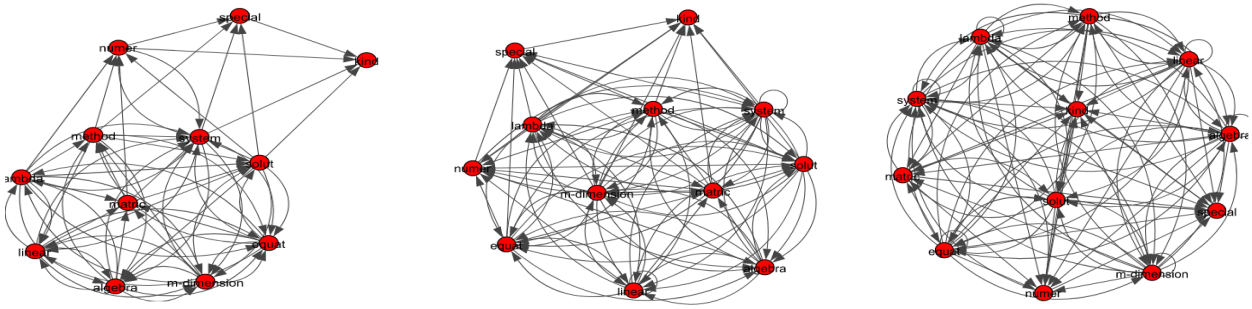

Figure 4: Evolution of graphs wrt window size

3 Question 2

1. **Line 1 :** $\mathcal{O}(|V|)$ because we will have to visit each vertex of the graph (row of the adjacency matrix) to assess his degree (the number of elements in the list/row)

2. **Line 2 :** $\mathcal{O}(|V|)$ (the while loop)

3. **Line 3 :** In the worst case, the element of the lowest value of the degree will be the last vertex stocked in the adjacency matrix, so this operation is $\mathcal{O}(|V|)$

4. **Lines 4-5-6-7 :** These operations have a constant cost $\mathcal{O}(Cst_1)$

   Let $N_{avg_{neighbors}}$ be the average number of neighbors of a node in this graph.

5. **Line 8 :** $\mathcal{O}(N_{avg_{neighbors}})$

6. **Line 9 :** This operation has a constant cost $\mathcal{O}(Cst_2)$

So to sum it up, the complexity of this naive version of the algorithm is :

$\mathcal{O}(|V|) + \mathcal{O}(|V|) \times (\mathcal{O}(|V|) + \mathcal{O}(Cst_1) + \mathcal{O}(N_{avg_{neighbors}}) \times \mathcal{O}(Cst_2))$
$\approx \mathcal{O}(|V|) \times (\mathcal{O}(|V|) + \mathcal{O}(N_{avg_{neighbors}}))$.
But since $N_{avg_{neighbors}} < |V|$, we have then a complexity of :
$\mathcal{O}(|V|) + \mathcal{O}(|V|) \times \mathcal{O}(|V|) = \mathcal{O}(|V|^2)$

4 Task 3 - Task 4

After filling the gaps in the function core_dec() and running it, we get the following decomposition which is identical to the one of coreness method.

It is important to mention that we lose the vertex **Reduc** after applying the pos_filtering when pre-processing the data, thus the results are different from those of the fig 4 in the homework.

$'algebra' : 6.0,' equat' : 6.0,' kind' : 3.0,' lambda' : 6.0,' linear' : 6.0,' m-dimension' : 6.0,' matric' : 6.0,$
$'method' : 6.0,' numer' : 4.0,' solut' : 6.0,' special' : 3.0,' system' : 6.0$

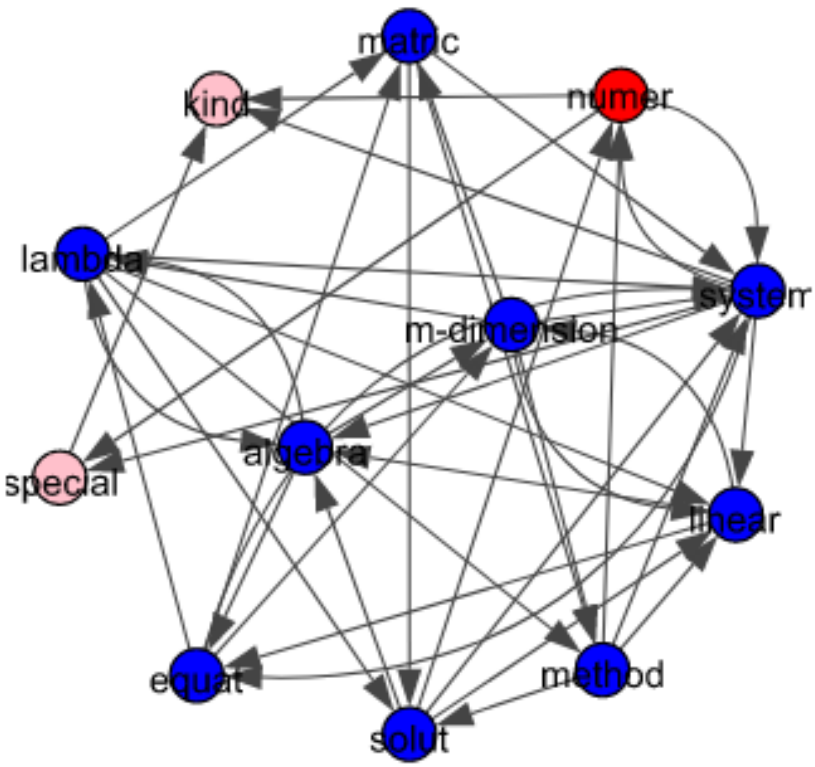

Figure 5: k-Core decomposition for the graph

5 Question 3

After filling the gaps in the corresponding python functions. We execute all the 4 algorithms to see their performance on the Hulth2003 dataset.

- First we consider the performance of both k-core and wk-core algorithms by changing the window size :

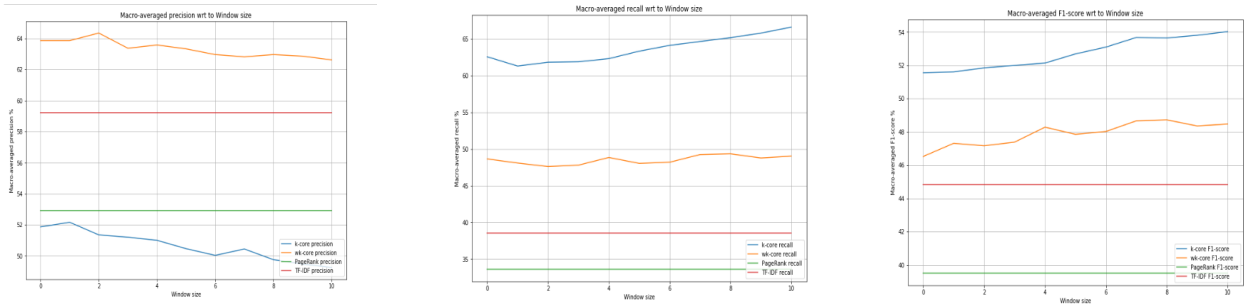

Figure 6: Macro-Averaged scores wrt window size

With a percentage of keywords retained set to 33% and a windows size equal to 6 we get the following results :

Algorithm	K-core	Weighed K-core	PageRank	TF-IDF
Macro-Averaged Precision	51.35	64.35	58.22	59.21
Macro-Averaged Recall	61.81	47.6	37.13	38.5
Macro-Averaged F1-score	51.84	47.16	43.58	44.58

- The K-core decomposition algorithms performs better than PageRank and TF-IDF; They seem to be efficient on quantifying nodes importance since they retrieve words within the densest parts of the graph. Keywords are mostly localized in those parts of the graph. The weighted K-core decomposition allows a high precision but low recall. In total, this approach seems to be more suitable for keyword extraction.
- TF-IDF is the only algorithm based on the BoW model. Although its limited by the fact that it does not take into account words positions, semantics and also co-occurrences, the algorithm allows a good precision but a lower recall. PageRank allows also a good precision. However, it tends to favor nodes with high degrees without tacking into account the density or the cohesiveness of the neighborhood of the node.

6 Question 4

The main advantages of k-core decomposition are :

- It is a computationally efficient method (can be computed in a nearly linear time).
- it captures well the importance of words since it considers the cohesiveness and not only the individual centrality and it is much more underlined with the weighted K-core method (Give much more attention to multiple cohesiveness in the text)
- It's adapted to keyword extraction: the number of extracted words (i.e, the size of the densest core) is eventually near to the number of true keywords since it depends on the structure of the graph.

A limitation of this approach is that it actually discards many other keywords that are in lower cores. We need then an appropriate way to choose the best cores that will increase recall while not defecting precision too much.

7 Question 5

- Computationally, it would be more efficient to begin by ordering the list of vertexes degrees before iterating.

- We can consider other approaches with different and strict criterion such as the K-truss algorithm to get more cohesive structures.

References

A Graph Degeneracy-based Approach to Keyword Extraction, 2016, Antoine J.-P. Tixier, Fragkiskos D. Malliaros, Michalis Vazirgiannis