1 Question 1 (2 points)

1. Evaluate the impact of window size on the density of the graph, defined as $\frac{|E|}{|V|(|V|-1)}$ For a directed graph and accessible via the .density() igraph method. What do you observe ?

Here, we will be looking at how does the window size affect the graph density on a small toy document. We will be looking at the density of G, our graph, for a window size going from 4 up to 10. We expect it to get higher and higher as the window size grows.

W = 2	3	4	5	6	7	8	9
D = 0,106	0.212	0.318	0.416	0.522	0.584	0.628	0.666

Figure 1: Graph G density for different window sizes

Of course, as the number of nodes stays, the denominator stays constant, while the number of relation, the nominator increases, we get a graph who gets denser and denser. The density goes towards one, but wouldn't reach it, as we work with directed graphs.

2 Question 2 (10 points)

1. What is the time complexity of our naive version of the k-core algorithm (Alg. 1) in the *unweighted* case ?

Algorithm 1 k-core decomposition

Input: graph $G = (V, E)$
Output: dict of core numbers c

$p \leftarrow \{v : degree(v)\}\ \forall v \in V$
while $|V| > 0$ **do**
  $v \leftarrow$ element of p with lowest value
  $c[v] \leftarrow p[v]$
  neighbors $\leftarrow \mathcal{N}(v, V)$
  $V \leftarrow V \setminus \{v\}$
  $E \leftarrow E \setminus \{(u, v) | u \in V\}$
  for $u \in$ neighbors **do**
    $p[u] \leftarrow \max(c[v], degree(u))$
  end for
end while

As there is a While as well as a for, the complexity of those lines are the only importants one, as nothing else is really complex.

1. $\mathcal{O}(V)$, Here, the while will work on V getting us a main term of final complexity
2. $\mathcal{O}(2 * |E|)$ for the "for"

As such, the complexity over the whole algorithm is $\mathcal{O}(|V|^2 + 2 * |E|)$, which we will now try and prove "theoretically".
To use the k-core algorithm in G, we have two steps.

1. The fist one is to choose a node in V with the minimum number of edges. If there are more, we just choose one randomly.

2. For the node we have chosen, we have to remove all edges connected to the node, and update the weights of all the other node using the give formula. This takes a time $\mathcal{O}(2 * E)$, which we will do V times

3. We remove the node from the graph G, taking $\mathcal{O}(1)$

3 Question 3 (2 Points)

1. What can you say about the performance of the different approaches ?

We've studied the keywords extraction performance over four different approaches, to which we calculated the $F1$ score. The four approaches being the k-core and its weighted version, as well as the TFIDF and the PageRank. We will now give our result and compare them.

	K-core	Weighted K-core	PageRank	TDIDF
Precision	51.86	63.68	60.18	59.21
Recall	62.56	48.64	38.3	38.5
F1 Score	51.55	46.52	44.96	44.85

Figure 2: Comparison of our different methods scores

Before comparing the methods, let us redefine the terms :
The **Precision** is the ratio of correctly predicted positive observations to the total predicted positive observations
The **Recall** is the ratio of the correctly predicted positives over all the true positives
The **F1Score** is the harmonic mean of the Precision and the Recall
Ranking solely by the F1 Score, we can see that the best keyword extractor is, in order from best to worse, the K-score, Weighted K-Score, PageRank, and the TFIDF which are close. The K-score being however the classifier with the worst precision out of them, but a recall far greater, opposed to the Weighted K-Score which has inverse strengths.

4 Question 4 (4 points)

1. What are the respective (dis)advantages of (un)weighted k-core?

The k-core algorithm has the great advantage of being implementable with a linear complexity $\mathcal{O}(E)$. It also has the highest Recall of all possible methods, meaning we have the fewest keywords missing from the ground truth, even if it comes at the cost precision, the tradeoff remains interesting. which in turn shows that we have many words falsely tagged as keywords compared to the other models. It has the highest F1 Score, meaning to our metrics, it is the best algorithm for keyword extraction. However, only keeping the main core is not optimal, as not all the keywords lie in it.

5 Question 5 (2 points)

1. How could these approaches be improved ?

By looking at [1], we can derive that changing our measure for the graph centrality may increase the performance. We might also want to split the datasets in different sizes of abstracts, in order to choose an appropriate window size for each lengths. Lastly, maybe we can choose the best level of coreness by heuristics.

References

[1] Antoine Tixier, Fragkiskos Malliaros, and Michalis Vazirgiannis. A graph degeneracy-based approach to keyword extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1860–1870, Austin, Texas, November 2016. Association for Computational Linguistics.