Le traitement des dangling nodes

L'algorithme que nous avons implémenté tient compte de l'impact potentiel des noeuds pendant sur le résultat du ranking. En conséquence une "normalisation" des noeuds pendant (dangling nodes) est ajoutée à la formule taxée du ranking afin de garantir que la somme des probabilité soit égale à 1.

Explication :

Un **dangling node** est un nœud du graphe qui **n'a pas de liens sortants** (par ex. une page Wikipédia qui ne contient aucun lien interne).

👉 Problème :
Dans PageRank, à chaque itération, chaque nœud redistribue son score à ses voisins.

- Si un nœud n'a **aucun voisin**, il "garde" son score.
- Résultat : une partie de la probabilité « disparaît », et la somme totale des rangs n'est plus égale à 1.

👉 Solution (méthode standard) :
On collecte toute la masse des rangs accumulée sur les dangling nodes (appelée **dangling mass**) et on la **redistribue uniformément à tous les nœuds du graphe**.

Formule :

$$nouveau\_rank(i) = (1 - \beta)/N + \beta(contribution\_reçues(i) + danglingMass/N)$$

- β = facteur d'amortissement (souvent 0.85).
- N = nombre total de nœuds.
- `contributions_reçues(i)` = somme des contributions des voisins vers le nœud i.
- `danglingMass` = somme des rangs des nœuds sans voisins.

👉 Exemple intuitif :

- Graphe avec 4 pages (A,B,C,D).

- À un moment, A et C n'ont pas de liens sortants et ont respectivement 0.1 et 0.2 de score.
- Donc `danglingMass = 0.3`.
- On redistribue 0.3 / 4 = 0.075 à **chaque page**.
- Cela garantit que la probabilité totale reste **1** à chaque itération.

Contexte et paramètres d'exécution :

Le programme a été développé sous VS Code sur un macbook pro M3 max (CPU = 14 cores).

C'est la raison pour laquelle j'ai choisi un partitionnement –partsDF = 28 conformément à la recommandation d'avoir un partitionnement = 2 fois le nombre d'executeurs disponibles et à la formule suivante :

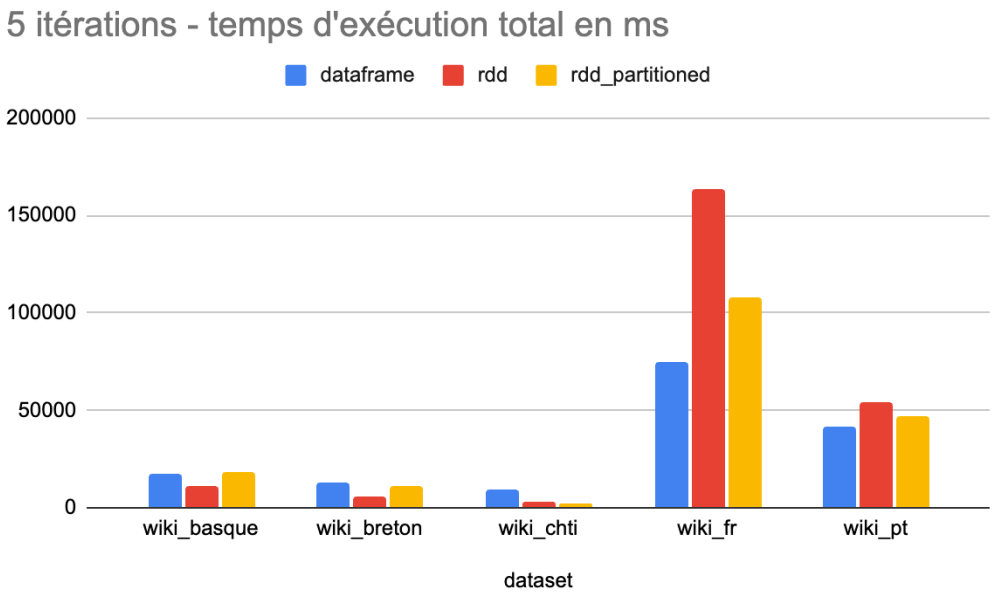
partsDF = min(max(2*cores, 4), max(32, nEdges/2000))

Résultats après 5 itérations :

Temps total d'exécution sur macbook pro M3 max (14 cores) = 10 mn

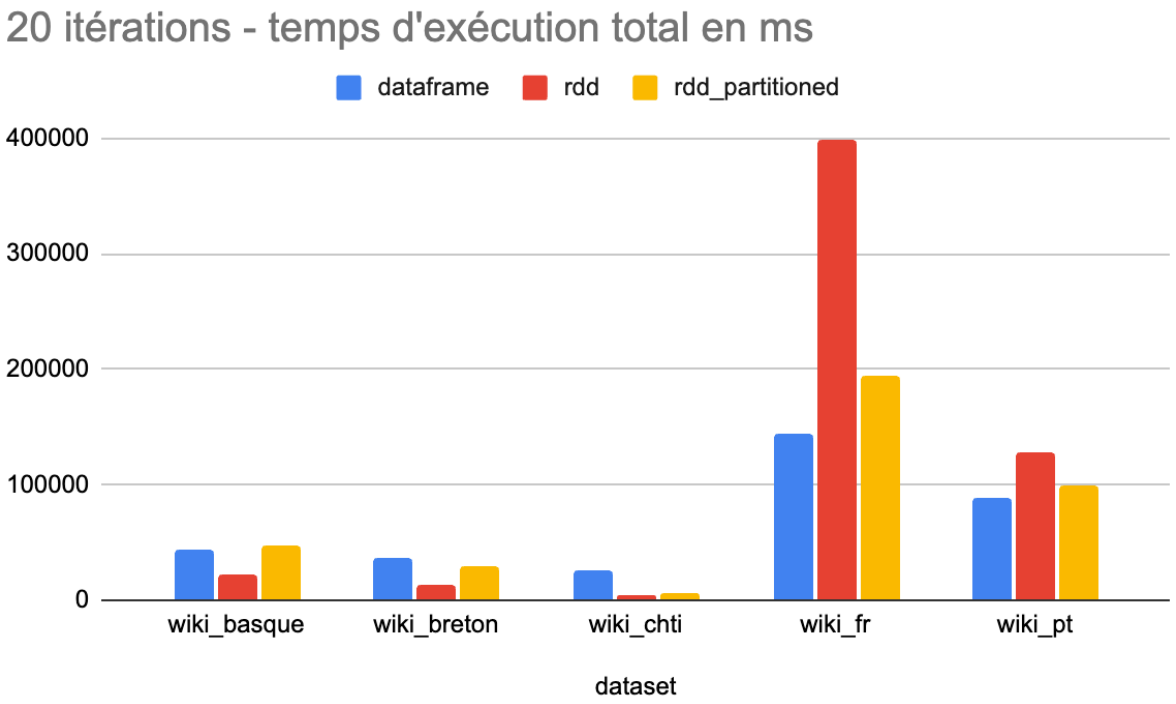
dataset	approach	nNodes	nEdges	parseMs	computeMs	totalMs	peakUsedMemMB
wiki_basque	dataframe	162191	3044392	4543	12943	17486	4929
wiki_basque	rdd	162191	3044392	5098	5956	11054	2627
wiki_basque	rdd_partitioned	162191	3044392	3799	14697	18496	4241
wiki_breton	dataframe	41631	894080	3003	10329	13332	2636
wiki_breton	rdd	41631	894080	2289	3573	5862	702
wiki_breton	rdd_partitioned	41631	894080	1461	9620	11081	1093
wiki_chti	dataframe	4121	16746	1317	7870	9187	2693
wiki_chti	rdd	4121	16746	1000	1728	2728	210
wiki_chti	rdd_partitioned	4121	16746	272	1519	1791	236
wiki_fr	dataframe	741237	38077524	40758	34205	74963	7891

wiki_fr	rdd	741237	38077524	54976	108635	163611	21357
wiki_fr	rdd_partitioned	741237	38077524	52299	55516	107815	22736
wiki_pt	dataframe	434180	17049397	19319	22425	41744	4749
wiki_pt	rdd	434180	17049397	17706	36689	54395	12554
wiki_pt	rdd_partitioned	434180	17049397	16893	30175	47068	11987



Résultat après 20 itérations :

Temps total d’exécution sur macbook pro M3 max (14 cores) = 22 mn


20 itérations - temps d'exécution total en ms
dataframe
rdd
rdd_partitioned
400000
300000
200000
100000
0
wiki_basque
wiki_breton
wiki_chti
wiki_fr
wiki_pt
dataset