NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC4052 Assignment 2

Gokul Ramesh U2222182H

Contents

Graph Generation

Before we discuss the various algorithms that I implemented, it is important to take note of the structure of the graphs that were used in testing these algorithms.

All graphs that were used were randomly generated, but also altered to ensure every single node has at least, either an outgoing or incoming edge, ensuring that no nodes are completely disconnected from the rest of the graph.

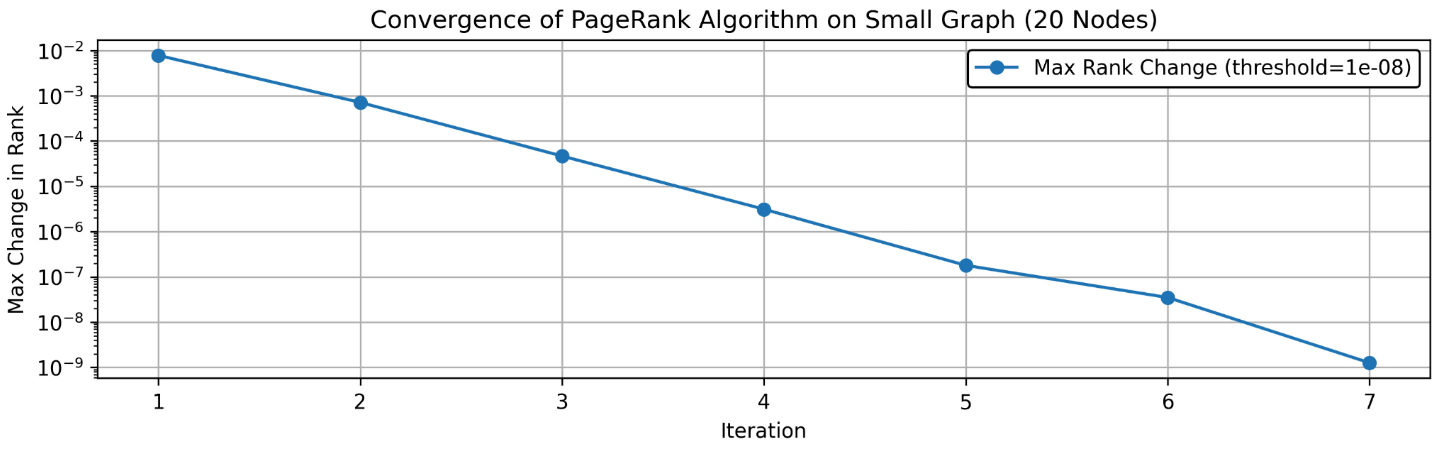
PageRank

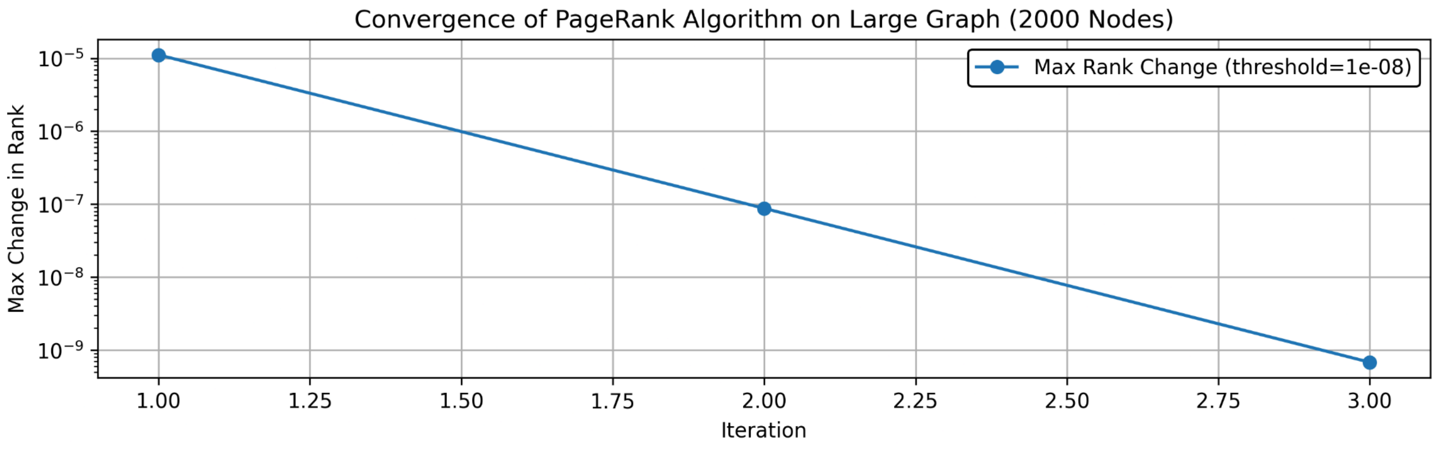
First, we will explore the PageRank algorithm that was taught in the Lecture. The implemented algorithm, follows this equation:

$$PR(u) = \frac{\lambda}{N} + (1-\lambda). \sum_{v \in B_u} \frac{PR(v)}{L_v}$$

For this algorithm, I tracked the maximum_change in rank across all nodes, for every iteration, until this maximum_change fell below a threshold = 1e^{-8}, to determine convergence.

We will look at the plots of the maximum_change on a small graph containing 20 nodes, and a large graph containing 2000 nodes.





An interesting observation from the graphs is that the large graph converged in only 3 iterations, which is fewer than the small graph which took 7 iterations.

This is because in larger graphs, individual node changes have less impact on the overall system. Plus, larger networks also provide more paths and connections, which create more well-defined structural patterns that converge quicker. This behaviour is similar to how a large sample size in statistics leads to more stable and predictable results, as local variations which could cause longer convergence times in smaller graphs, are smoothed out.

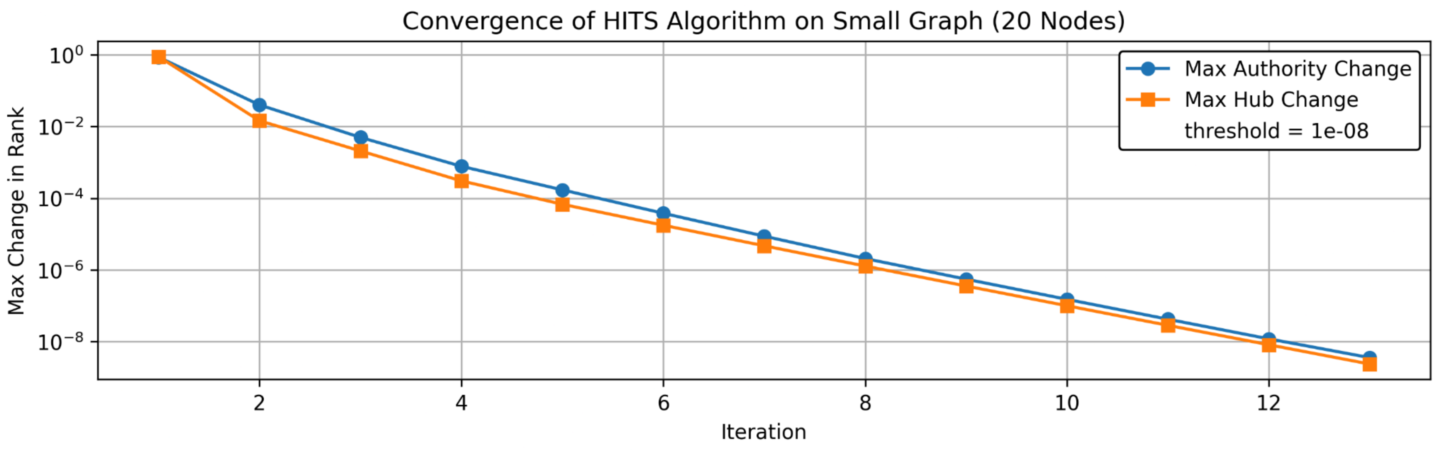
Hyperlink-Induced Topic Search (HITS)

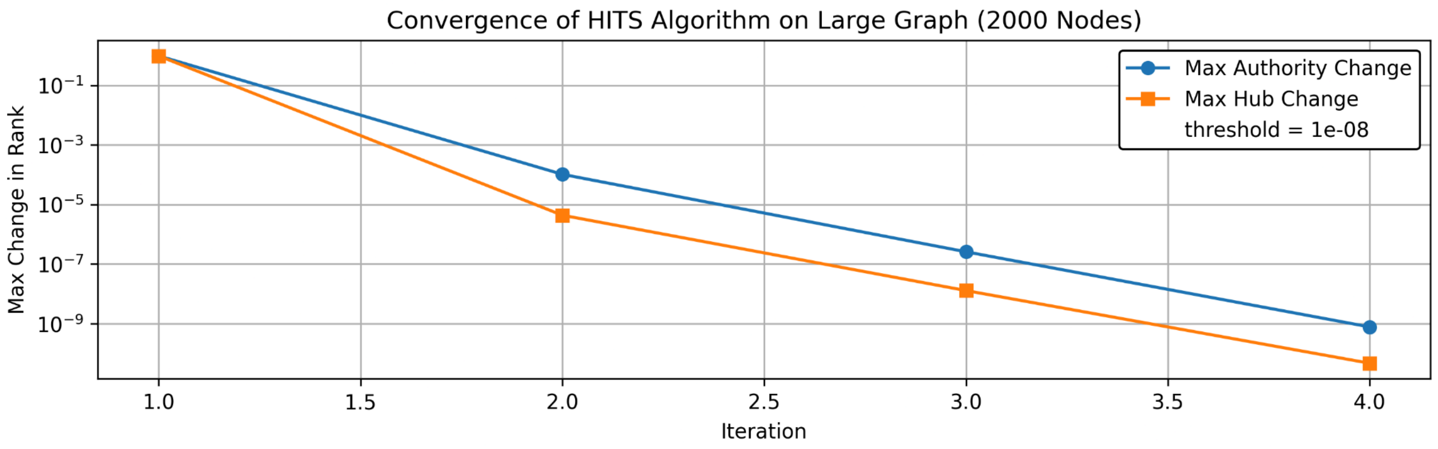
Next, we look at the standard HITS algorithm. While PageRank simulates a random surfer's behaviour by using probability of landing on each page as its score, HITS uses authority and hub scores, where a page has a high authority score if it is linked to by many pages, and a high hub score if it links to many authoritative pages. Due to this, PageRank is more suitable for general-purpose ranking, while HITS excels in scenarios where distinguishing between different types of important pages is valuable.

The algorithm follows this equation:

$$a_i^{(t+1)} = \sum_{\{j:j\to i\}} h_j^{(t)};\ h_i^{(t+1)} = \sum_{\{j:i\to j\}} a_j^{(t+1)}$$

Next, we look at the performance of the algorithm on the graphs, with the same threshold = 1e^{-8}:





We can observe similar behaviour where the large graph converged in fewer iterations, 4 iterations, then the small graph, 13 iterations.

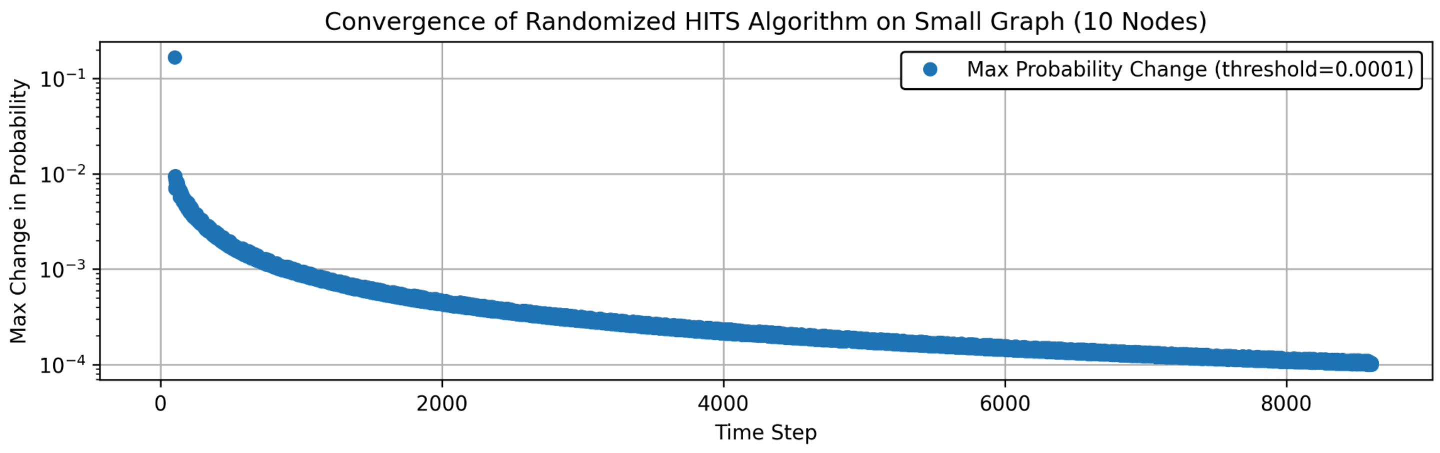
We can also see that although the large graph converges in a similar number of iterations as in PageRank, the small graph takes almost twice the number of iterations using HITS at 13, compared to 7 using PageRank.

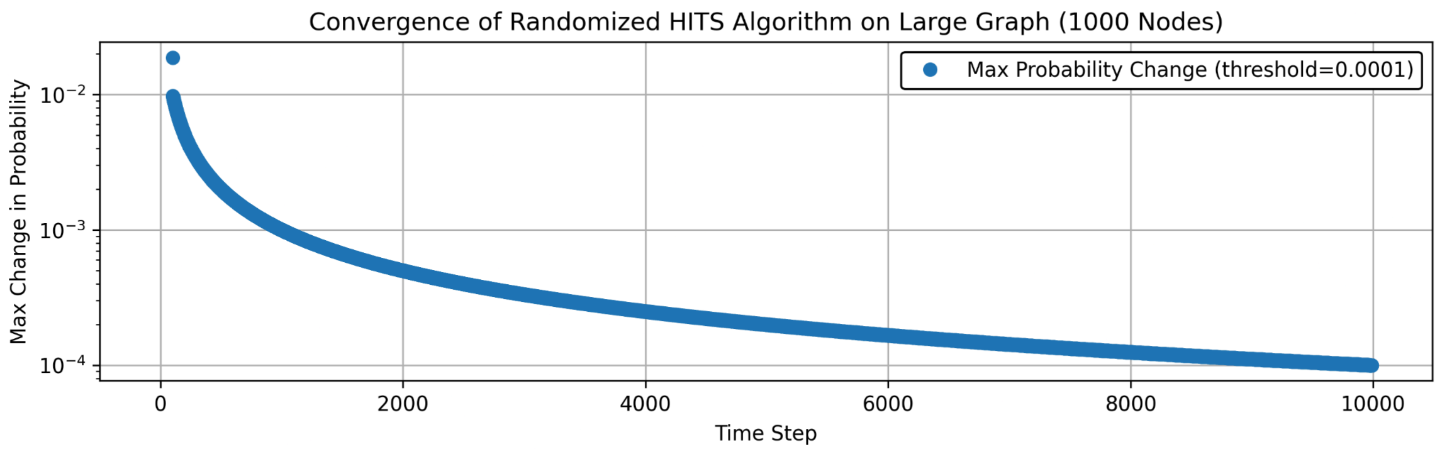
Randomized HITS

The next algorithm was one of two algorithms proposed in the “Stable Algorithms for Link Analysis” paper. Both Randomized HITS and PageRank use random walk principles, but their approaches differ where PageRank uses a deterministic calculation with a damping factor, while Randomized HITS uses a separate algorithm for walking, alternating between following forward and backward links to update hub and authority scores. Additionally, where the random jump probability in PageRank is used to handle dead ends and isolated components, random jumps in Randomized HITS do the same thing while also maintaining the dual scoring system of traditional HITS.

Also, while PageRank uses looping through all nodes as an iteration, Randomized HITS uses time steps, where a step is taken every time the random surfer walks, starting from the starting node, meaning nodes are only updated when the surfer reaches them.

Now, we will look at the effect of running the algorithm on a small graph of 10 nodes, and a large graph of 1000 nodes, with threshold = 1e^{-4}.





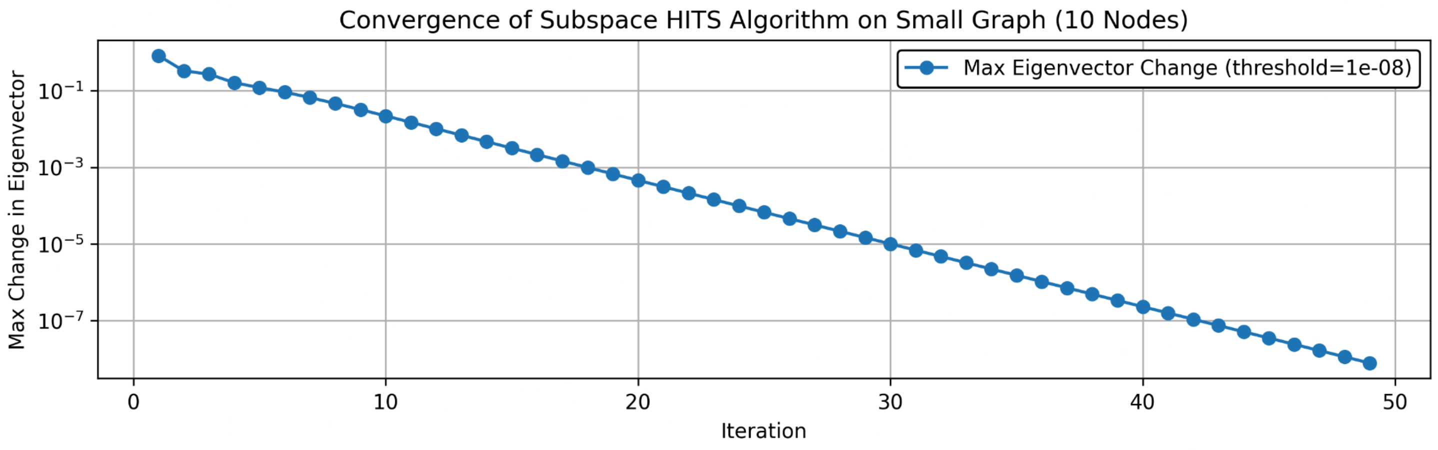
This time, we see that both graphs take a similar number of time steps to converge, where the small graph takes 8608 time steps, and the large graph takes 9989 time steps.

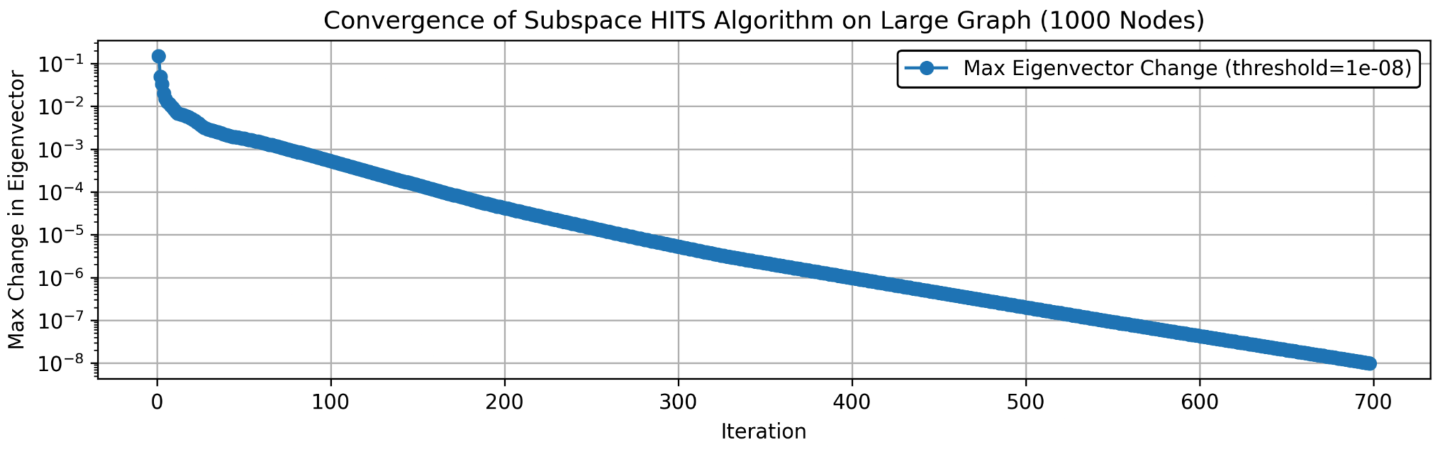
This in contrast to the behaviour from previous algorithms where large graphs converge more quickly. The reason for this is while the scores for the nodes in large graphs converge more quickly, since we only reach a single node per time step, on average it takes much more time steps to reach all nodes in the large graph. This effect counters the faster convergence in large graphs, leading to similar time steps in both graphs.

Subspace HITS

The final algorithm, Subspace HITS, was the other algorithm proposed in the previously mentioned paper. Subspace HITS works by discovering multiple dimensions in a network through eigenvector analysis, unlike PageRank's single dimension approach. Subspace HITS also extends the traditional HITS algorithm by finding multiple orthogonal sets of hub and authority scores. These multiple sets of hub and authority vectors are iteratively updated and orthogonalized, to ensure each dimension captures a unique aspect of the network's linking structure. These features make it better at discovering different communities or topics within a network, even if there is overlap.

Similar to Randomized HITS, I ran this algorithm on a small and large graph which have 10 nodes and 1000 nodes respectively, with threshold = 1e^{-8}.





Unlike previous algorithms, where the large graph converged faster or at similar speed to the small graph, for this algorithm, the large graph takes much more iterations to converge at 698 iterations, while the small graph takes 49 iterations. This is because Subspace HITS tries to identify multiple distinct, orthogonal patterns in the graph structure, and with more nodes there are more possible patterns and relationships to sort through. Since each eigenvector must capture a unique aspect of the network structure, in large graphs, these patterns are more intricate and take more iterations to stabilize, thus unlike PageRank, Subspace HITS does not benefit from larger structural patterns in large graphs.

Using LLMs to Revamp Algorithms to Make Generative Search Engine

The advantage of LLMs is their ability to learn from past data and generate human-like text. While the PageRank and HITS algorithms treat pages as simple nodes with authority and hub scores based purely on link structure, we can enhance these algorithms using LLMs before incorporating them into a Generative Search Engine.

For PageRank, instead of treating all links equally with fixed weights, we can use LLMs to analyze the content surrounding each link, such as the content of the pages in the surroundings which would help to determine more accurate transition probabilities. For example, if a link appears to be highly relevant contextually to the user's query in the search engine, the LLM could assign it a higher weight in the PageRank calculation. This creates a content-aware PageRank where the random walk is guided by an understanding of the user query rather than just link structure and only whether or not a link contains the same words as the query.

For the HITS based algorithms, we can enhance both authority and hub calculations using LLMs. When computing authority scores, instead of just counting incoming links, the LLM can analyze the content of linking pages to determine how contextually similar they are to the target page. Similarly for hub scores, the LLM can evaluate how well a page curates and organizes its outgoing links by analyzing the contextual relationships between the linked pages. This process creates a more meaningful understanding of what makes a good hub or authority page, which also performs better.

These enhanced algorithms can then be used to build a Generative Search Engine. When a user submits a query, the system uses the LLM-enhanced PageRank and HITS scores to identify the most relevant and authoritative pages. Then, instead of simply returning a ranked list of these pages, the LLM processes their content to generate a comprehensive answer. The authority and hub scores help ensure that the generated response is based on reliable sources, while the LLM's understanding of content relationships helps synthesize information across multiple pages.

In conclusion, enhancing the search algorithms allowed us to create a Generative Search Engine, transforming the simple ranking system into a tool that can better answer queries by generating the answer by pooling content from numerous pages and tailoring it according to the user's query.

References

1. Ng, A. Y., Zheng, A. X., & Jordan, M. I. (2001). Stable algorithms for link analysis. *Stable Algorithms for Link Analysis*. https://doi.org/10.1145/383952.384003
2. Wikipedia contributors. (2024, December 27). *HITS algorithm*. Wikipedia. https://en.wikipedia.org/wiki/HITS_algorithm

Appendix

1. Jupyter Notebook file with Code : https://github.com/xGokull/SC4052/blob/main/Assignment%202/Code.ipynb