Program Structures and Algorithms

Fall 2024 - Final Project: Web Crawler

Group members: Akshit Saxena, Sanskruti Manoria
NUID(s): 002475083, 002643300
GitHub link: https://github.com/saxena-akshit/WebKrawler-INFO6205
App name : WebKrawler
Task: Build a web crawler that uses parallel/asynchronous processing to find web pages, given a set of starting pages.
TechStack Used: Java , Spring Boot Framework, Maven, Neo4j (Graph DB Integrate), Postman (for API testing), AWS EC2 for deployment,React Js (GUI)

1.Abstract

The web crawler project is designed to retrieve and process web pages efficiently using parallel/asynchronous processing. The crawler performs breadth-first traversal of web pages, calculates PageRank, and represents the results as a graph in a graph database.
This report focuses on the use of Priority Queue and Breadth-First Search (BFS) traversal in the implementation, detailing their roles and derived observations.

2.Web-Crawler Implementation (Code Overview & Output)

The Java-based Spring-boot application web crawler efficiently traverses web pages using a combination of a Priority Queue for task scheduling and Breadth-First Search (BFS) traversal for systematic exploration.

URLs are encapsulated as **UrlTask** objects, prioritized based on predefined metrics such as domain relevance, content type, and blacklist status.

A **PriorityBlockingQueue** ensures high-priority URLs are processed first, optimizing the crawling order. BFS traversal is employed to explore pages level by level, avoiding premature deep exploration and maintaining an even distribution of resources.

A **visitedUrls** set prevents redundant processing, while extracted links are dynamically queued with appropriate priorities.

The crawler stores the web graph in a Neo4j database, with Page nodes representing **URLs** and **LINKS_TO** edges representing relationships between them. After crawling, the PageRank algorithm calculates page importance by iteratively distributing rank values across links until convergence, with results updated in the Neo4j database.
This implementation effectively leverages key data structures like PriorityBlockingQueue, Set, and Map to balance performance, scalability, and functionality, ensuring relevant pages are prioritized and efficiently processed.

2.1 Code Implementation- (WebCrwalerService.java)

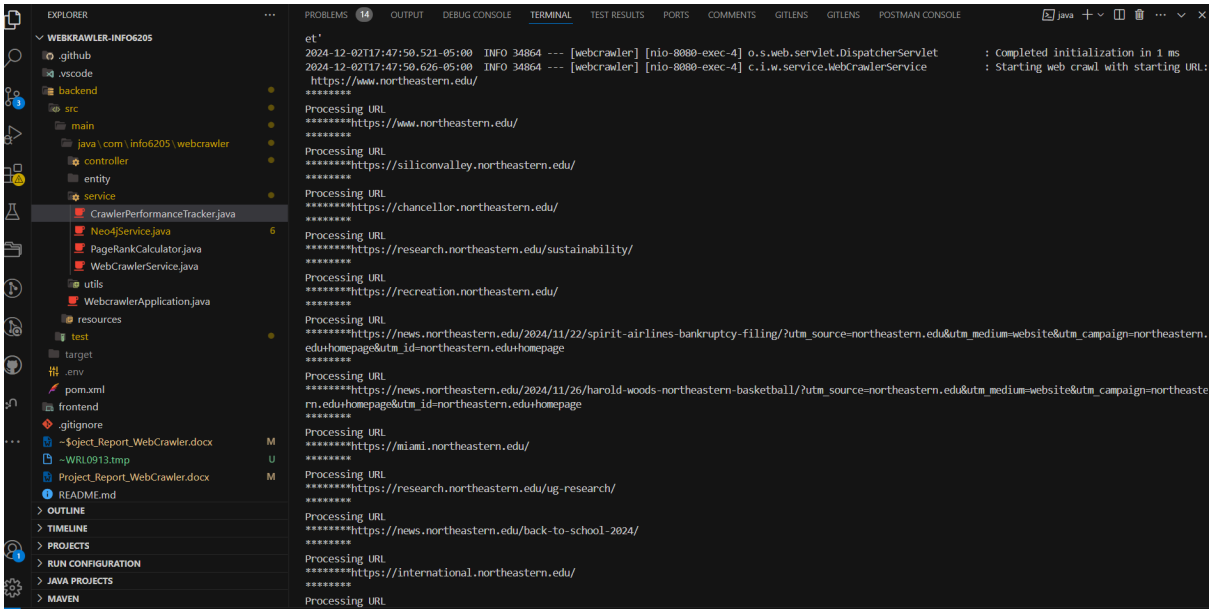
```java
public WebCrawlerService(
        @Value("${crawler.threadPoolSize}") int threadPoolSize,
        @Value("${crawler.maxDepth}") int maxDepth,
        @Value("${crawler.rateLimit}") double rateLimit,
        Neo4jService neo4jService,
        PageRankCalculator pageRankCalculator) {
    this.threadPool = Executors.newFixedThreadPool(threadPoolSize);
    this.rateLimiter = RateLimiter.create(rateLimit);
    this.queue = new PriorityBlockingQueue<>(initialCapacity:10, Comparator.comparingInt(UrlTask::getPriority));
    this.visitedUrls = Collections.synchronizedSet(new HashSet<>());
    this.maxDepth = maxDepth;       saxena-akshit, 7 days ago • feat: dummy neo4j adapter
    this.neo4jService = neo4jService;
    this.pageRankCalculator = pageRankCalculator;
}
```

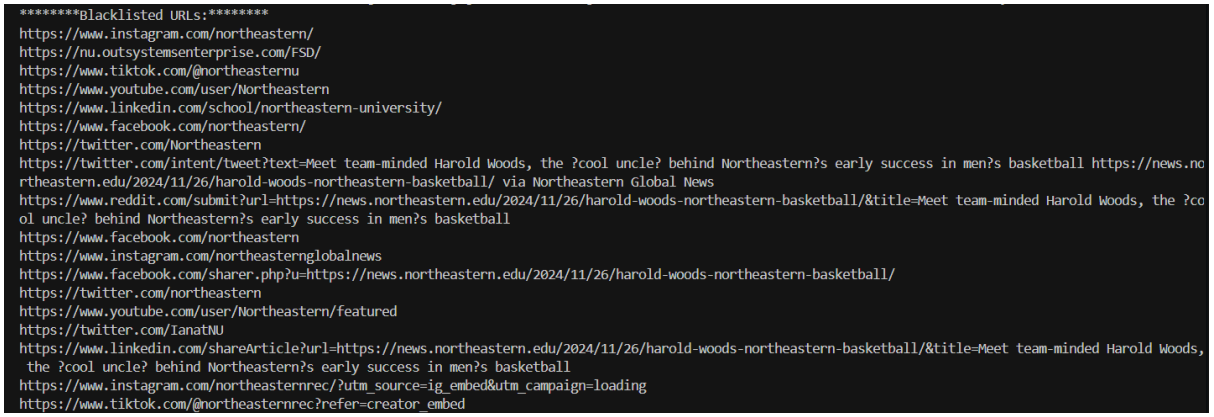
```java
backend > src > main > java > com > info6205 > webcrawler > service > WebCrawlerService.java > Language Support for Java(TM) by Red Hat > WebCrawlerService > sta
33   public class WebCrawlerService {
67       public Map<String, Object> startCrawl(String startUrl) throws InterruptedException {
69           logger.info("Starting web crawl with starting URL: {}", startUrl);
70           saxena-akshit, 7 days ago • feat: dummy neo4j adapter
71           // Clear existing Neo4j graph
72           neo4jService.clearGraph();
73           queue.add(new UrlTask(startUrl, depth:0, calculatePriority(startUrl)));
74
75   >       while (true) {···
99           threadPool.shutdown();
100  >       try {···
102  >       } catch (InterruptedException e) {···
105
106          performanceTracker.endTracking();
107          // Print blacklisted and low priority URLs
108          System.out.println(x:"********Blacklisted URLs:********");
109          blacklistedUrlsList.forEach(System.out::println);
110
111          System.out.println(x:"********Low Priority URLs:********");
112          lowPriorityUrlsList.forEach(System.out::println);
113
114          return calculatePageRankResponse();
115      }
```

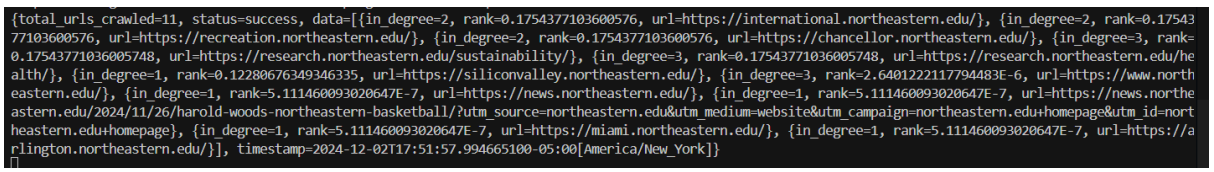
```java
    protected Map<String, Object> calculatePageRankResponse() {
        List<String> nodes = neo4jService.getNodes();
        Map<String, List<String>> graph = neo4jService.getGraph();
        Map<String, Double> pageRanks = pageRankCalculator.computePageRank(nodes, graph);
        neo4jService.updatePageRank(pageRanks);

        List<Map<String, Object>> resultData = pageRanks.entrySet().stream()
                .map(entry -> {
                    String url = entry.getKey();
                    Double rank = entry.getValue();
                    int inDegree = (int) nodes.stream().filter(node -> graph.getOrDefault(node, new ArrayList<>()).contains(url)
                    return Map.<String, Object>of(k1:"url", url, k2:"rank", rank, k3:"in_degree", inDegree);
                })
                .sorted(Comparator.comparingDouble((Map<String, Object> m) -> (Double) m.get(key:"rank")).reversed()) // Sort by
                .collect(Collectors.toList());

        return Map.of(
                k1:"status", v1:"success",
                k2:"timestamp", ZonedDateTime.now().toString(),
                k3:"total_urls_crawled", nodes.size(),
                k4:"data", resultData
        );
    }
```

```java
backend > src > main > java > com > info6205 > webcrawler > service > WebCrawlerService.java > Language Support for Java(TM) by Red Hat > WebCrawlerService > calc
33   public class WebCrawlerService {
267      int calculatePriority(String url) {
269          // Ignore blacklisted URLs
270          for (String blacklistedUrl : BLACKLISTED_URLS) {
271              if (url.contains(blacklistedUrl)) {
272                  synchronized (blacklistedUrlsList) {
273                      if (!blacklistedUrlsList.contains(url)) {
274                          blacklistedUrlsList.add(url);
275                      }
276                  }
277                  return Integer.MAX_VALUE;
278              }
279          }
280          // Ignore dark web URLs
281          if (url.contains(s:".onion")) {
282              return Integer.MAX_VALUE;
283          }
284
285          if (url.matches(regex:".*\\.(jpg|jpeg|png|gif|bmp|mp4|avi|mkv|mov|wmv|flv|webm)$")) {
286              synchronized (lowPriorityUrlsList) {
287                  if (!lowPriorityUrlsList.contains(url)) {
288                      lowPriorityUrlsList.add(url);
289                  }
290              }
291              return 20;
292          }
293          You, 3 days ago • Ignoring blacklisted and low-priority URLs in C...
294          if (url.contains(s:"facebook.com") || url.contains(s:"twitter.com") || url.contains(s:"instagram.com") || url.contains(s
295              synchronized (blacklistedUrlsList) {
296                  if (!blacklistedUrlsList.contains(url)) {
297                      blacklistedUrlsList.add(url);
298                  }
299              }
300              return 20;
301          }
```

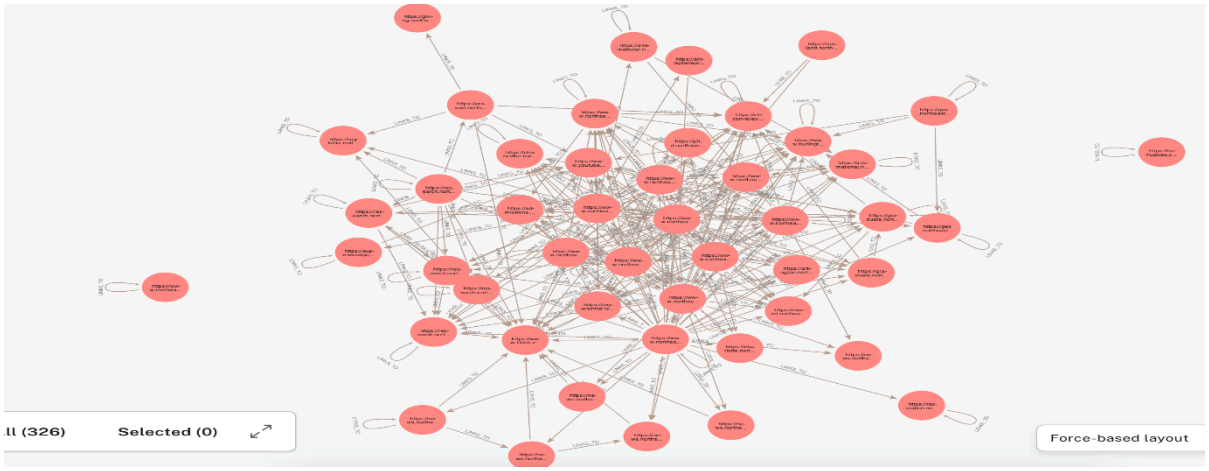
2.2 Code Output-



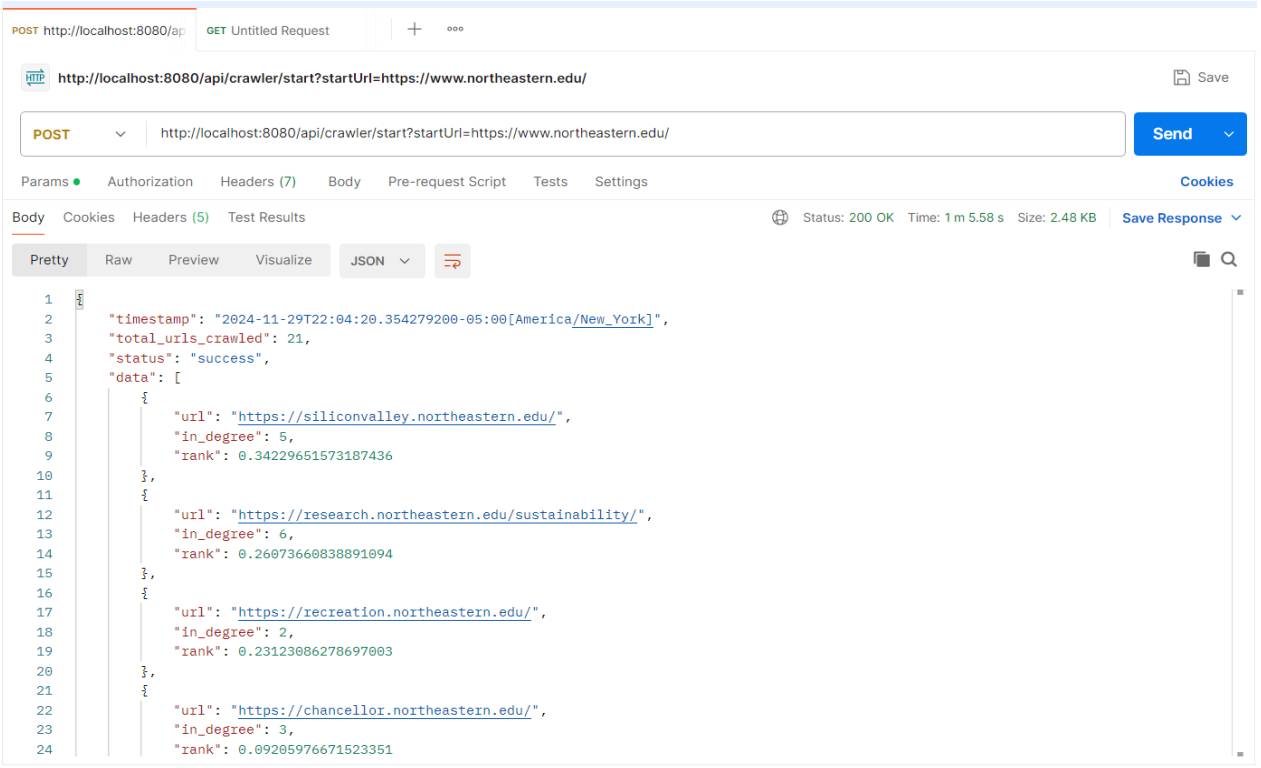




2.3 Neo4j Graph Output- (sample graph with 326 URLs crawled)



2.4 API response (Postman)-



3. Getting Started

Before starting, ensure you have the following installed:

Install Maven: Version 3.6 or later. - Add MAVEN_HOME and include MAVEN_HOME/bin to your system's PATH.

Installation

Clone the repository:

```
git clone git@github.com:saxena-akshit/WebKrawler-INFO6205.git
```

Navigate to Directory:

```
cd backend
```

Create a `.env` file in `/backend`

Configure application.properties: (add the following in .env)

```
NEO4J_URL=neo4j+s://9d807b45.databases.neo4j.io
NEO4J_USERNAME=neo4j
NEO4J_PASSWORD=m5g_-SKT4P7W6zX2Djgg4qGn9TpUR6RIXYqCCGWFLME
crawler.threadPoolSize=10 // change this for benchmarking
crawler.maxDepth=3 // change this for benchmarking
crawler.rateLimit=100
```

Run the application

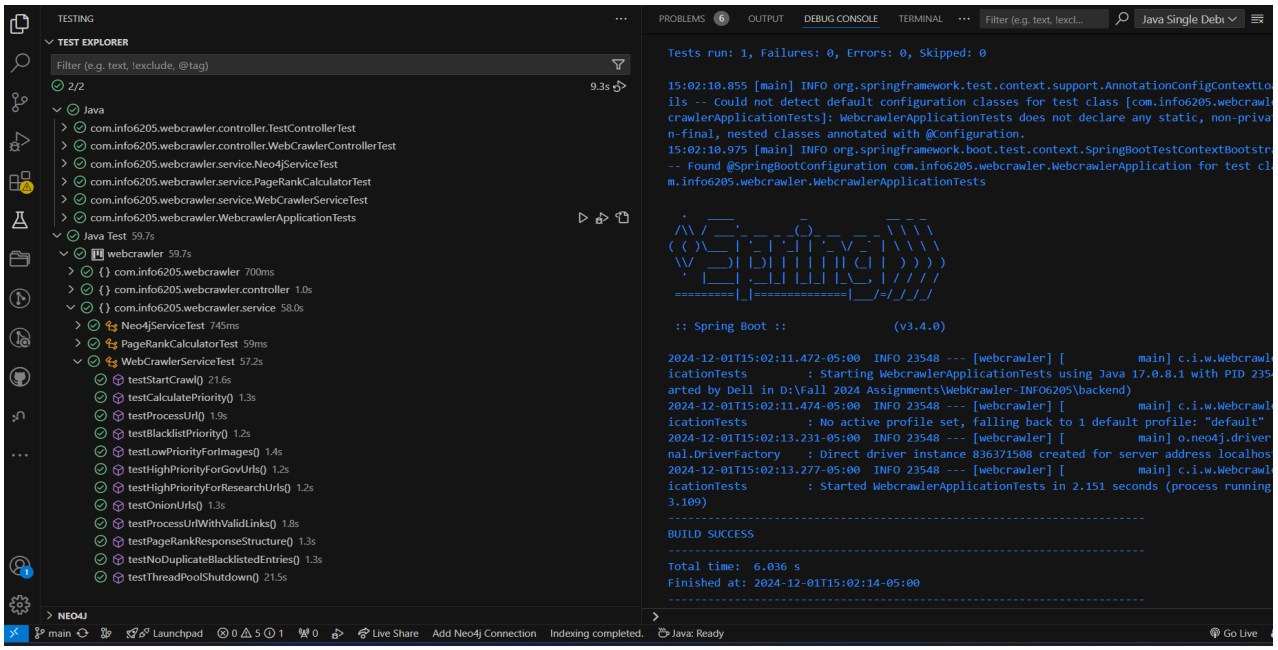
```
mvn clean install && mvn spring-boot:run
```

This starts the server on port 8080.

Crawl API available at `http://localhost:8080/api/crawler/start?startUrl={startUrl}`

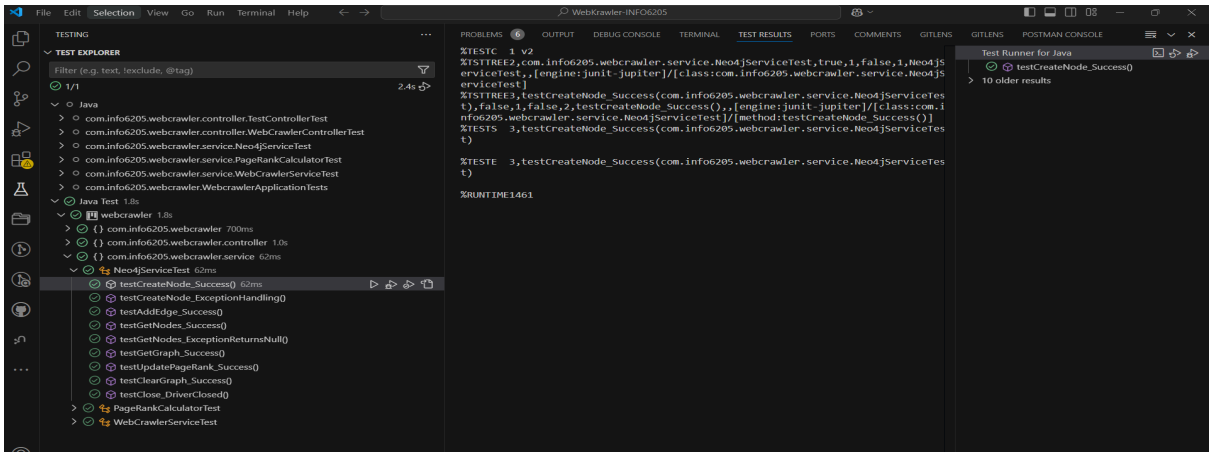
4. Unit Test Cases

Overview of Test cases:

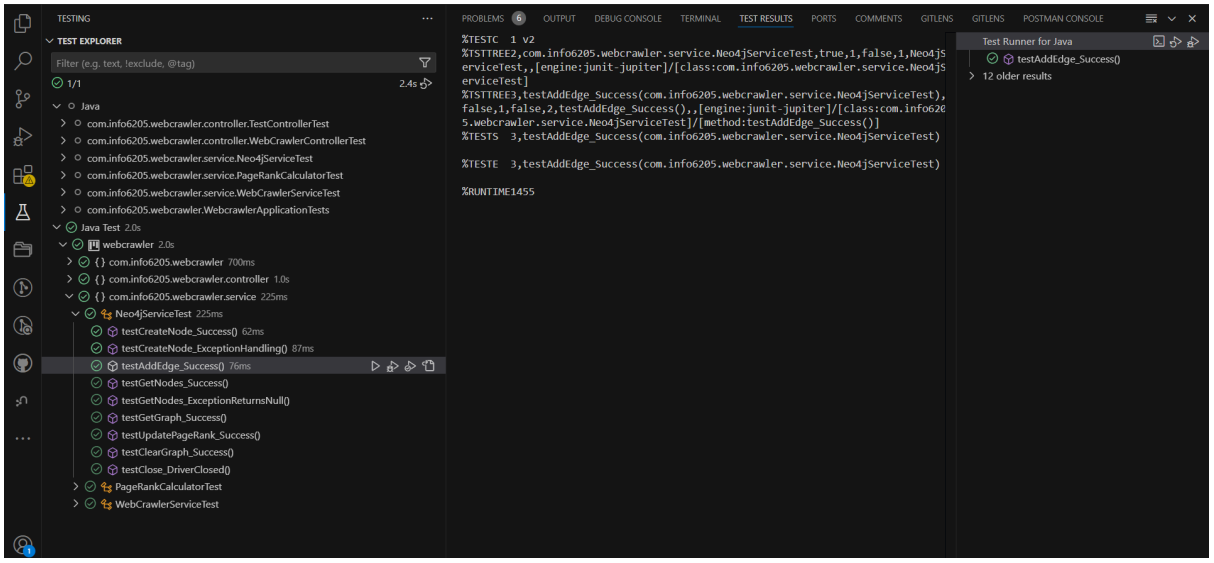


4.1 Neo4j Tests

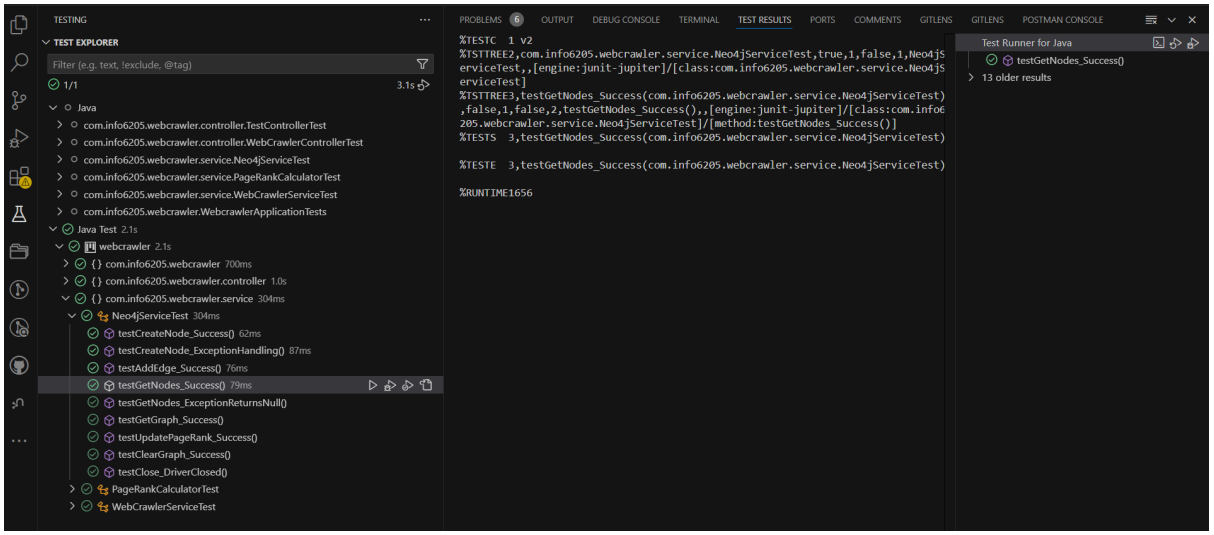
4.1.1 TestCreateNode()



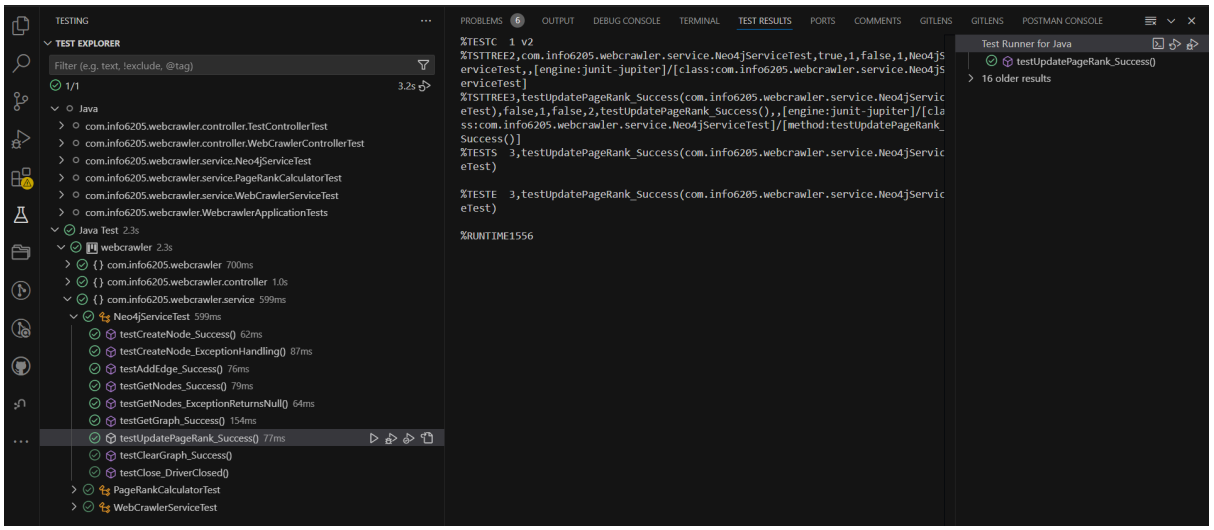
4.1.2 TestAddEdge_Sucess()



4.1.3 TestGetNodes_Sucess()

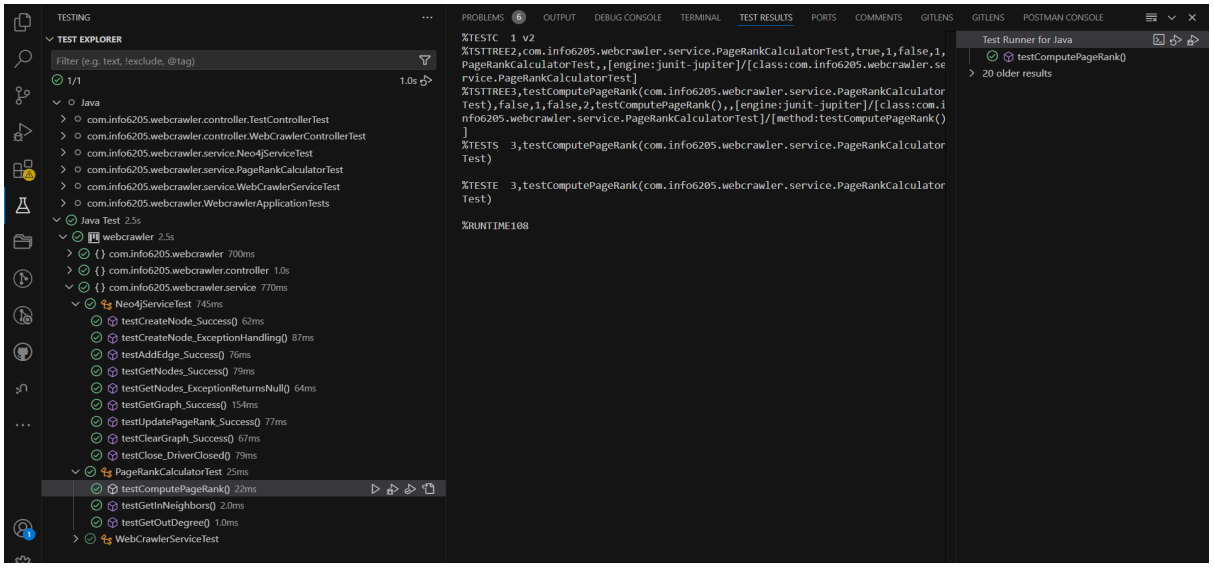


4.1.4 TestUpdatePagerank_Success()

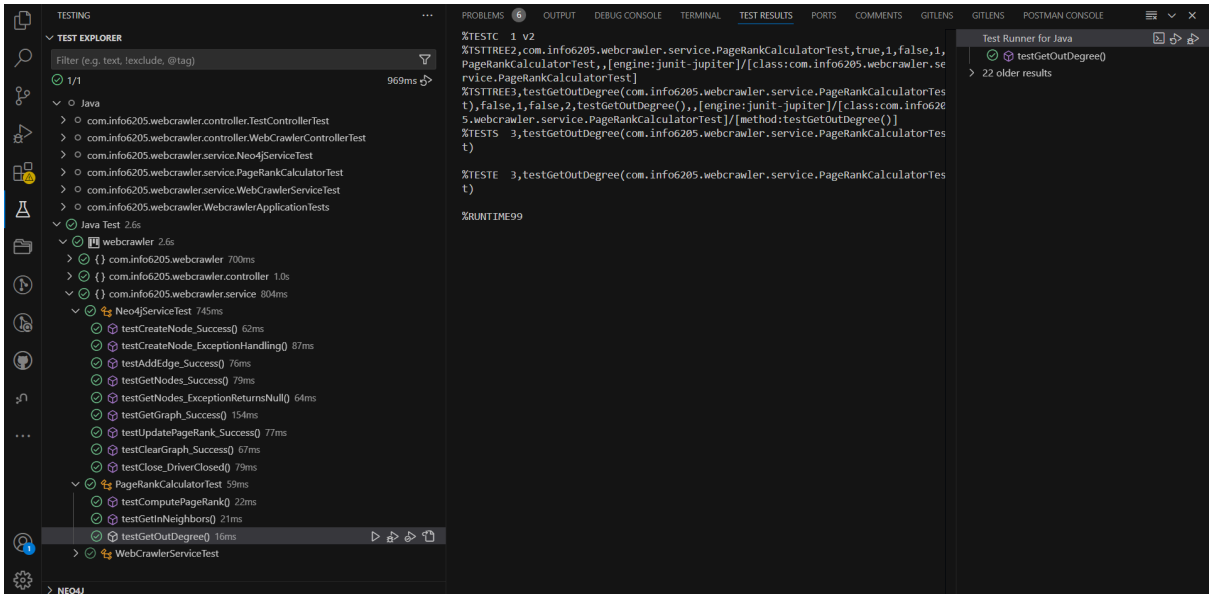


4.2.PageRankCalculatorTest

4.2.1 TestComputePageRank()

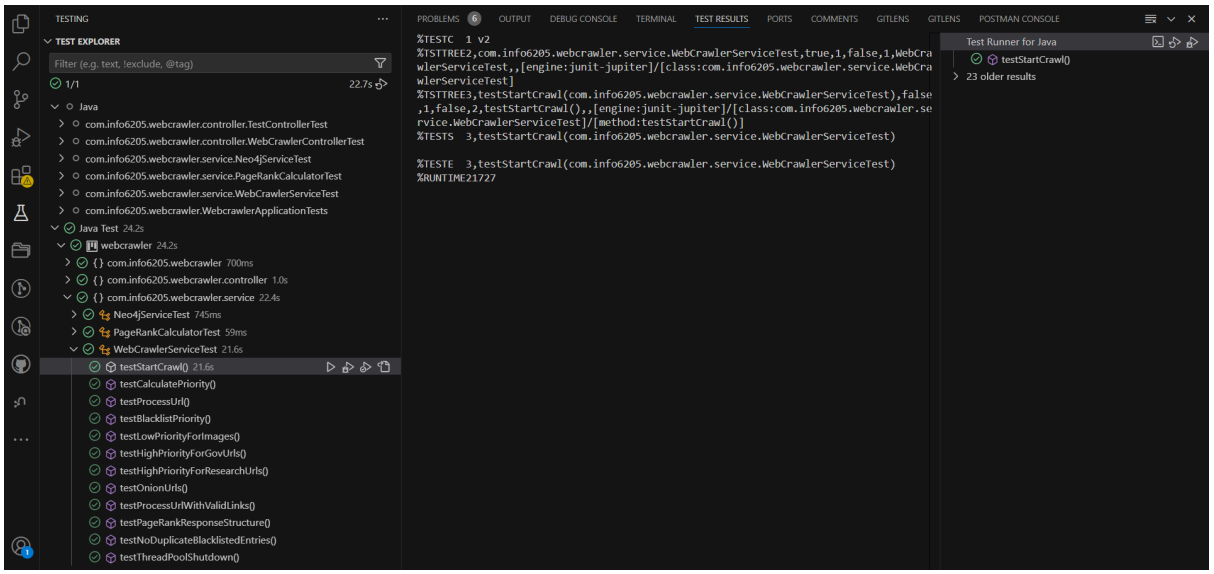


4.2.2 TestGetOutDegree()

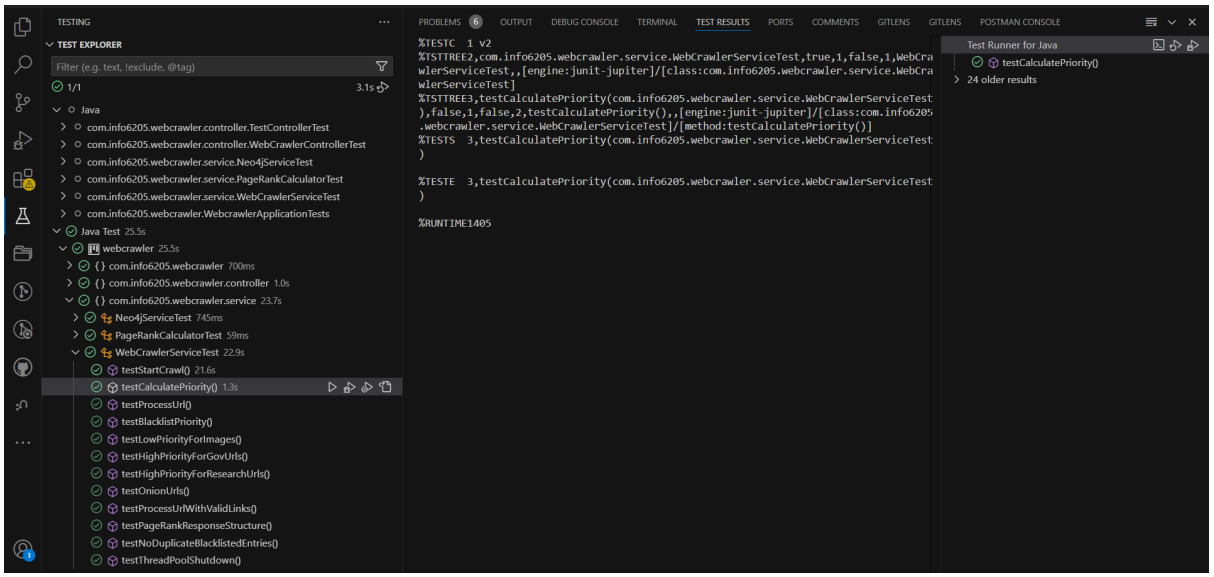


4.3. WebCrawlerServices

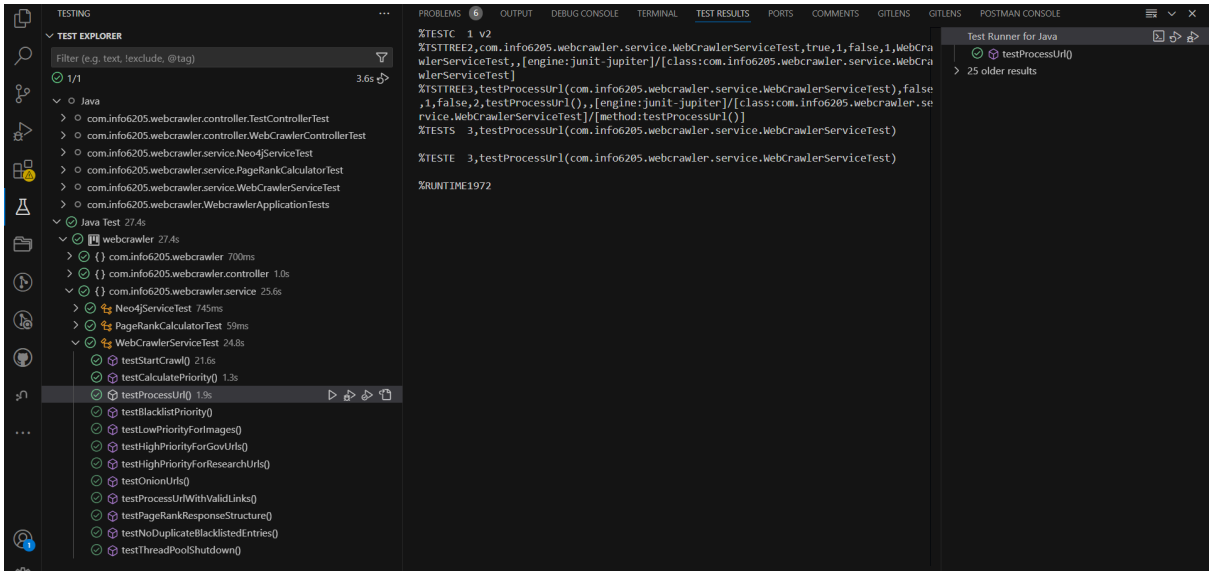
4.3.1 TestStartCrawl()



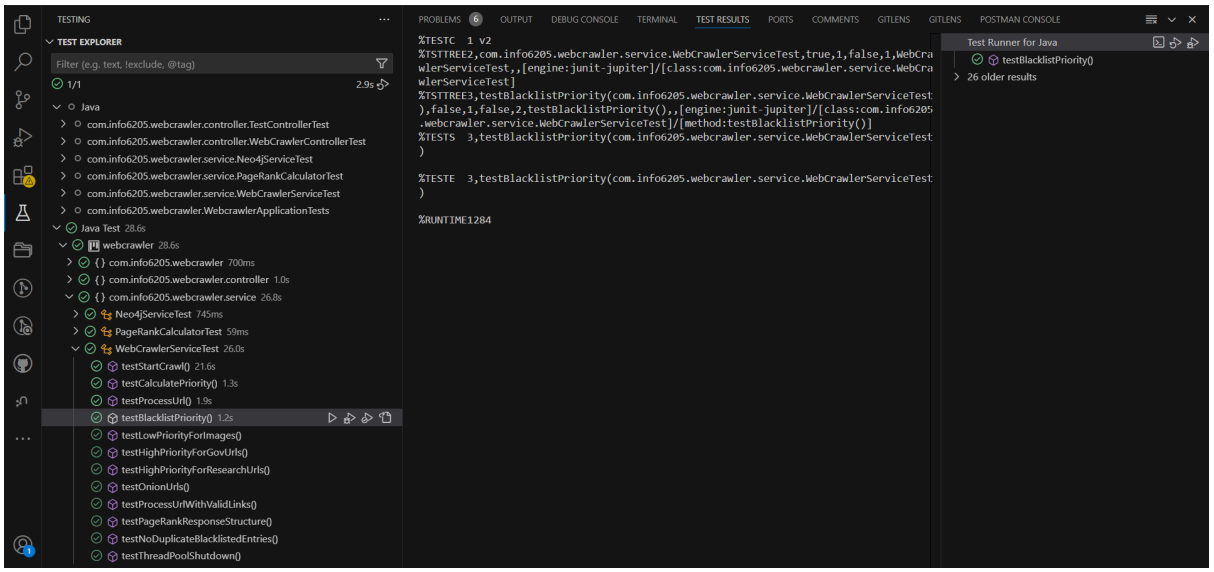
4.3.2 TestCalculatePriority()



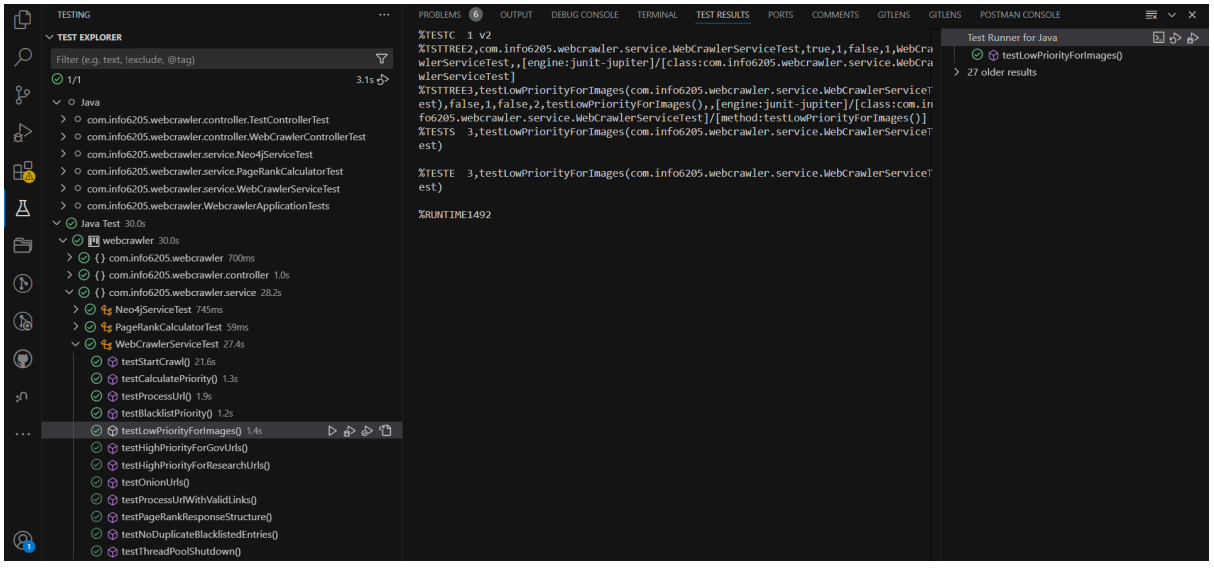
4.3.3 TestProcessURL()



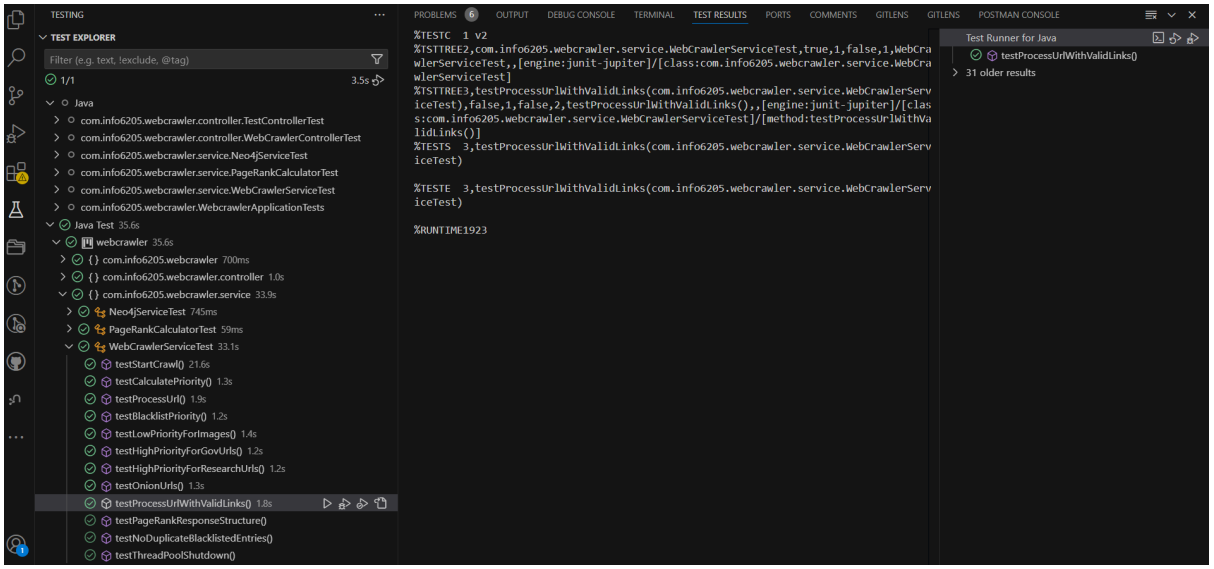
4.3.4 TestBlackListPriority()



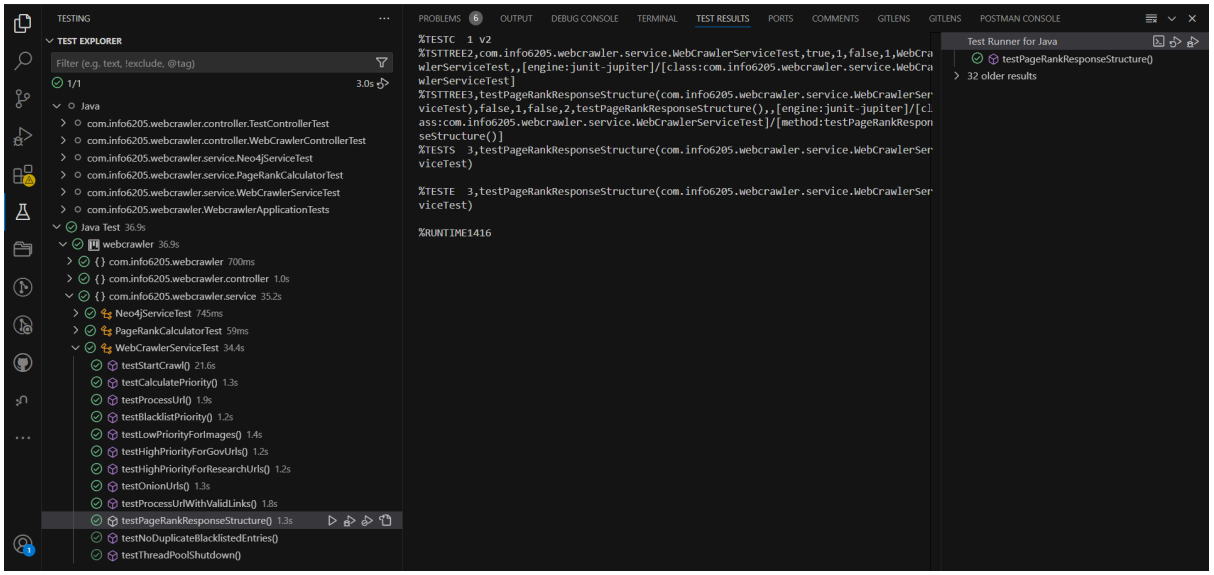
4.3.4 TestLowPriorityForImages()



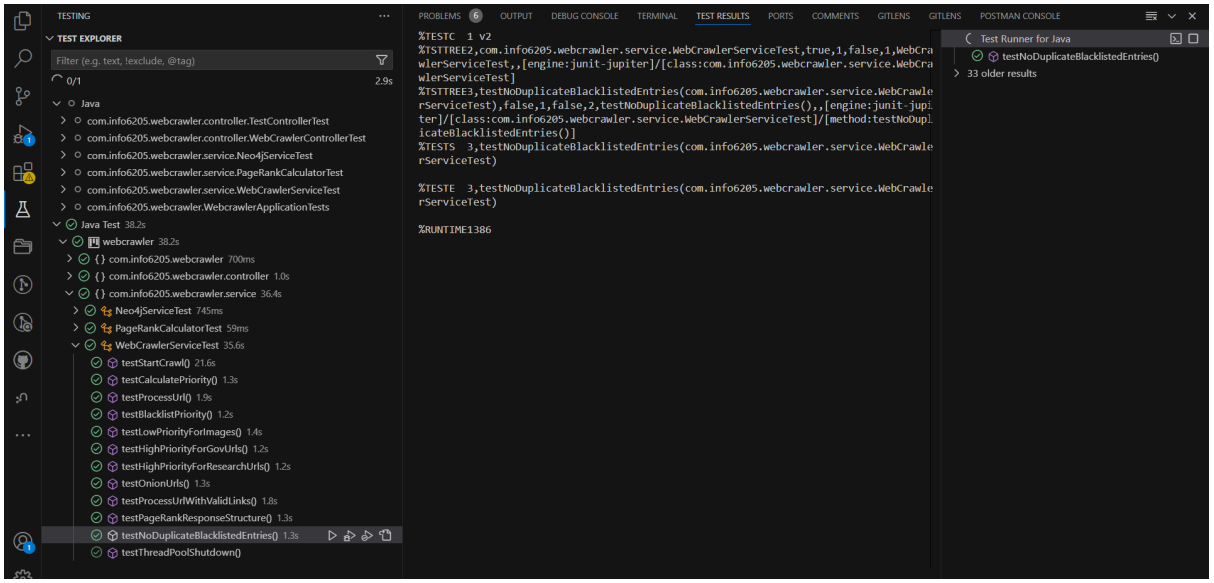
4.3.5 TestProcessURLWithValidLinks()



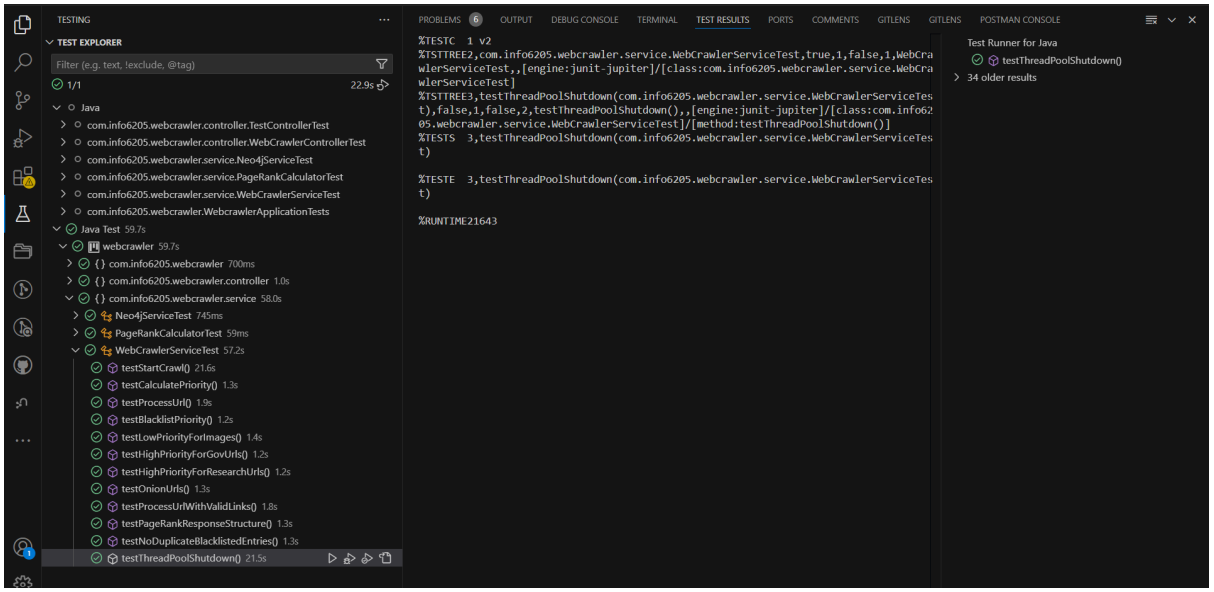
4.3.6 TestPageRankResponseStructure()



4.3.7 TestNoDuplicateBlackListedEntries()



4.3.8 TestThreadPoolShutDown()



5. Benchmarking:

CrawlerPerformanceTracker.java
Root URL: https://www.northeastern.edu/

- Below benchmarking is run for the starting URL https://www.northeastern.edu/, the benchmark is set to cut off at **51 URLs** (for standardizing the results)
- The benchmark runs for the following **MaxDepth** values : {2, 3, 4, 5} and the following **ThreadPoolSize** values : {2, 4, 8, 16, 32, 48}
- The results of the benchmark are written to a file `crawler_performance_metrics.csv` with each run appending a line in that file.
- Below are the sample contents of that file based on our runs.

```
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T00:52:56.304920Z,https://www.northeastern.edu,2,2,51,465886,0.11,62
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T01:37:34.418900Z,https://www.northeastern.edu,2,3,51,456454,0.11,65
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T01:51:25.201176Z,https://www.northeastern.edu,2,4,51,458618,0.11,68
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T02:12:19.317945Z,https://www.northeastern.edu,2,5,51,472200,0.11,65
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T02:25:56.629264Z,https://www.northeastern.edu,4,2,51,235077,0.22,76
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T02:32:19.213639Z,https://www.northeastern.edu,4,3,51,221376,0.23,73
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
```

```
eakMemoryUsedMB
2024-12-02T02:39:05.915695Z,https://www.northeastern.edu,4,4,51,233710,0.22,76
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T02:45:02.128431Z,https://www.northeastern.edu,4,5,51,234324,0.22,70
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:04:36.679933Z,https://www.northeastern.edu,8,2,51,123057,0.41,96
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:15:17.435974Z,https://www.northeastern.edu,8,3,51,122178,0.42,94
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:20:38.494084Z,https://www.northeastern.edu,8,4,51,123859,0.41,92
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T02:50:47.590493Z,https://www.northeastern.edu,8,5,51,124860,0.41,99
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:25:45.539164Z,https://www.northeastern.edu,16,2,51,98994,0.52,142
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:29:18.316433Z,https://www.northeastern.edu,16,3,51,123637,0.41,114
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:33:36.574582Z,https://www.northeastern.edu,16,4,51,131466,0.39,145
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:37:36.739304Z,https://www.northeastern.edu,32,2,51,67836,0.75,247
StartTimestamp,StartUrl,ThreadPoolSize,MaxDepth,TotalUrlsCrawled,CrawlDurationMs,UrlsPerSecond,P
eakMemoryUsedMB
2024-12-02T03:47:06.316673Z,https://www.northeastern.edu,48,5,501,335451,1.49,219
```

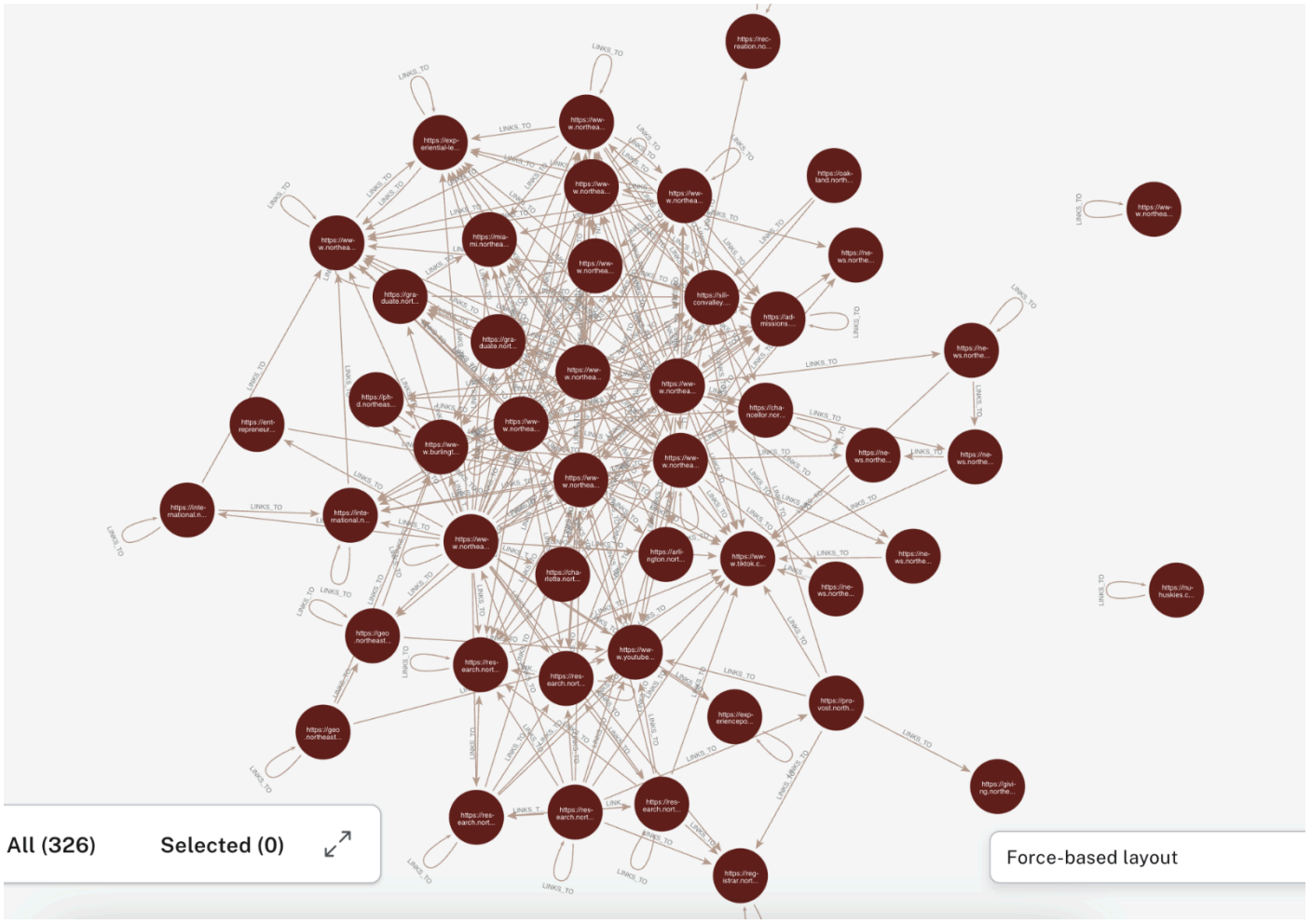
Here are the same results, tabulated.

StartTimestamp	StartUrl	ThreadPoo	MaxDepth	TotalUrlsC	CrawlDura	UrlsPerSe	PeakMemoryUsedMB
2024-12-02T00:52:5	https://www	2	2	51	465886	0.11	62
2024-12-02T01:37:3	https://www	2	3	51	456454	0.11	65
2024-12-02T01:51:2	https://www	2	4	51	458618	0.11	68
2024-12-02T02:12:1	https://www	2	5	51	472200	0.11	65
2024-12-02T02:25:5	https://www	4	2	51	235077	0.22	76
2024-12-02T02:32:1	https://www	4	3	51	221376	0.23	73
2024-12-02T02:39:(	https://www	4	4	51	233710	0.22	76
2024-12-02T02:45:(	https://www	4	5	51	234324	0.22	70
2024-12-02T03:04:3	https://www	8	2	51	123057	0.41	96
2024-12-02T03:15:1	https://www	8	3	51	122178	0.42	94
2024-12-02T03:20:3	https://www	8	4	51	123859	0.41	92
2024-12-02T02:50:4	https://www	8	5	51	124860	0.41	99
2024-12-02T03:25:4	https://www	16	2	51	98994	0.52	142
2024-12-02T03:29:1	https://www	16	3	51	123637	0.41	114
2024-12-02T03:33:3	https://www	16	4	51	131466	0.39	145
2024-12-02T03:37:3	https://www	32	2	51	67836	0.75	247
2024-12-02T03:47:(	https://www	48	5	51	335451	1.49	219

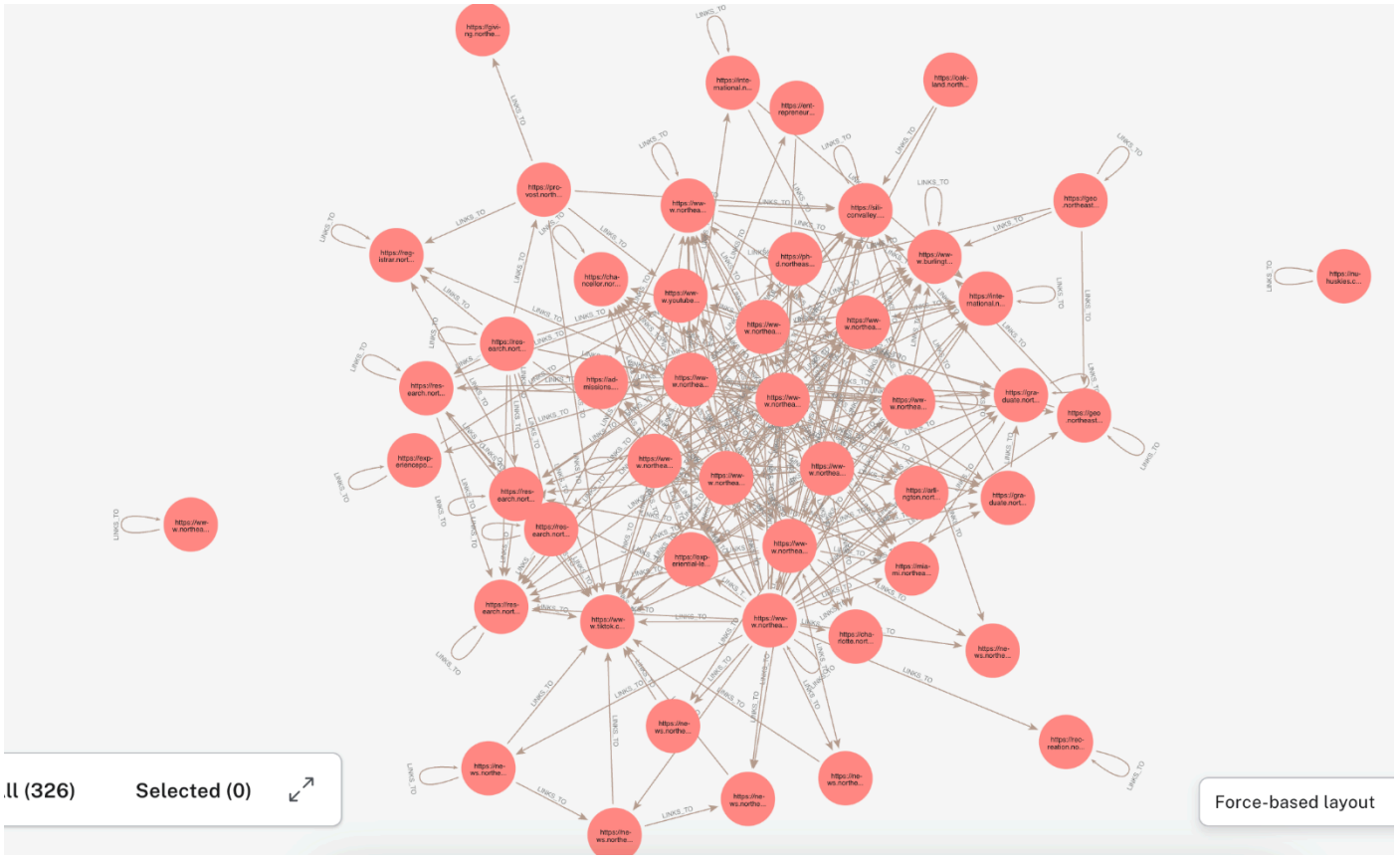
Variations in Graph based on Thread and Depth size

*(Note: Figure name T**a**-T**b** is the graph of a web crawl with ThreadSize **a** and Max Depth **b**)*

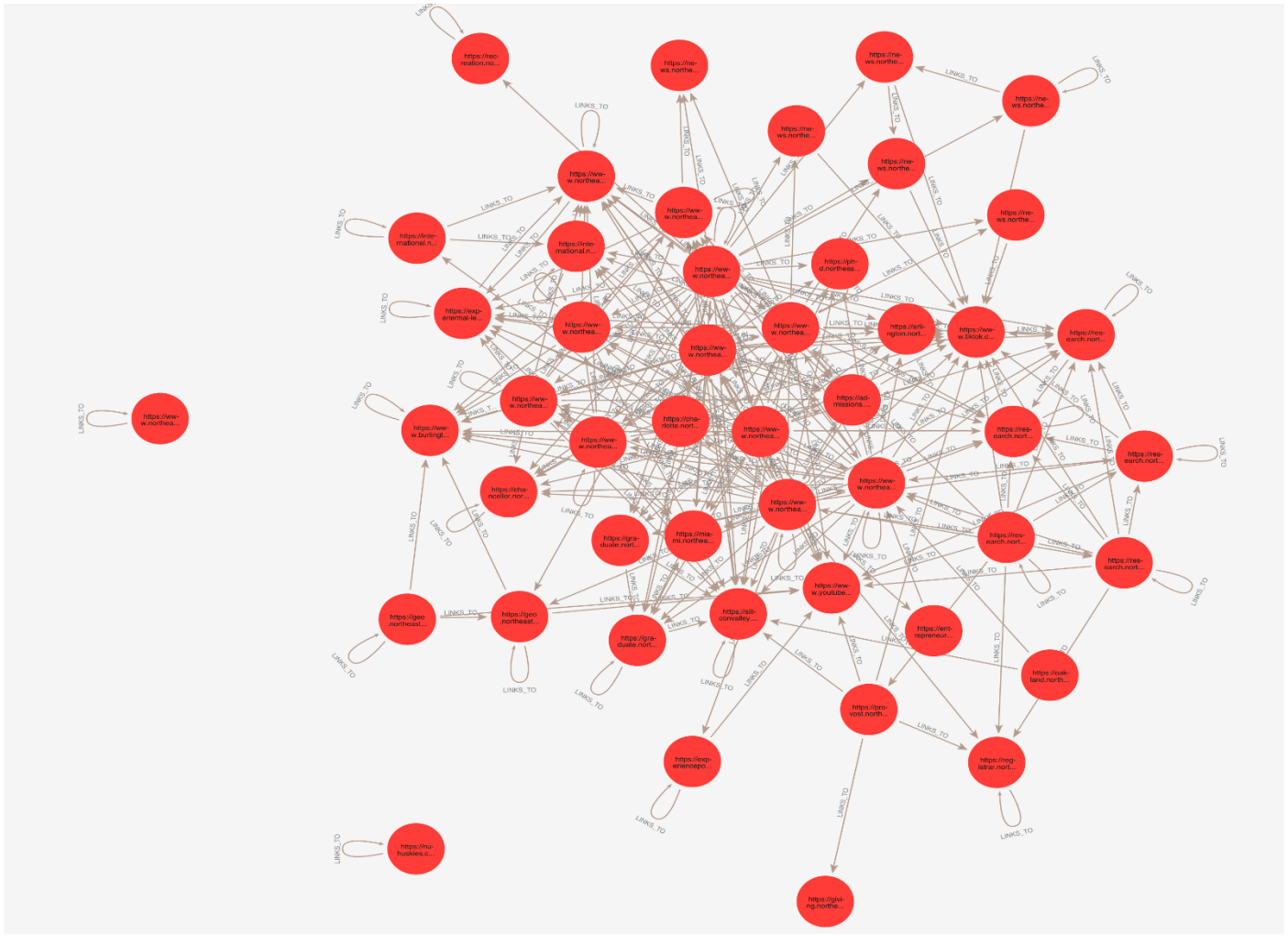
T-2 D-2



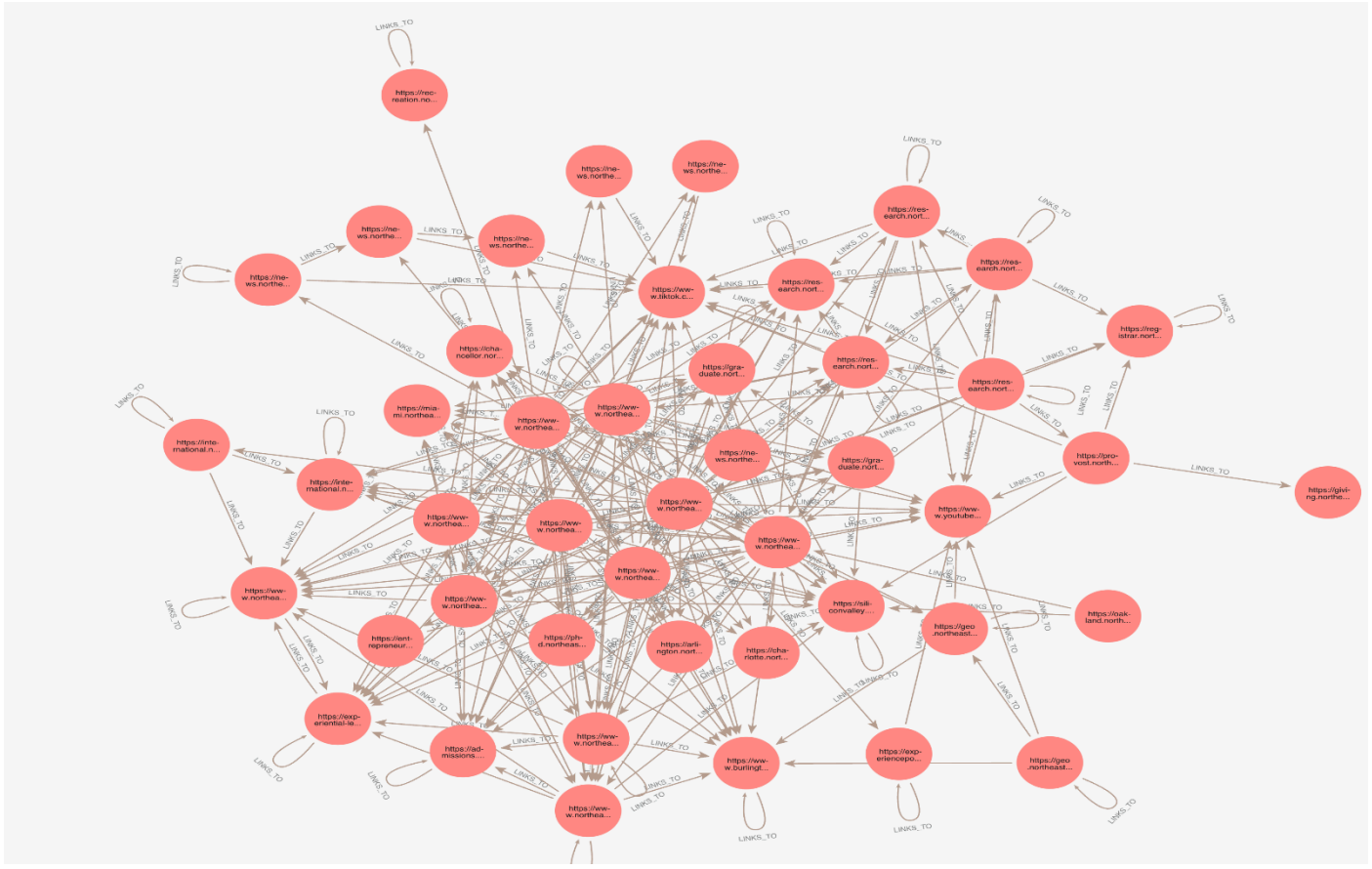
T-2 D-3



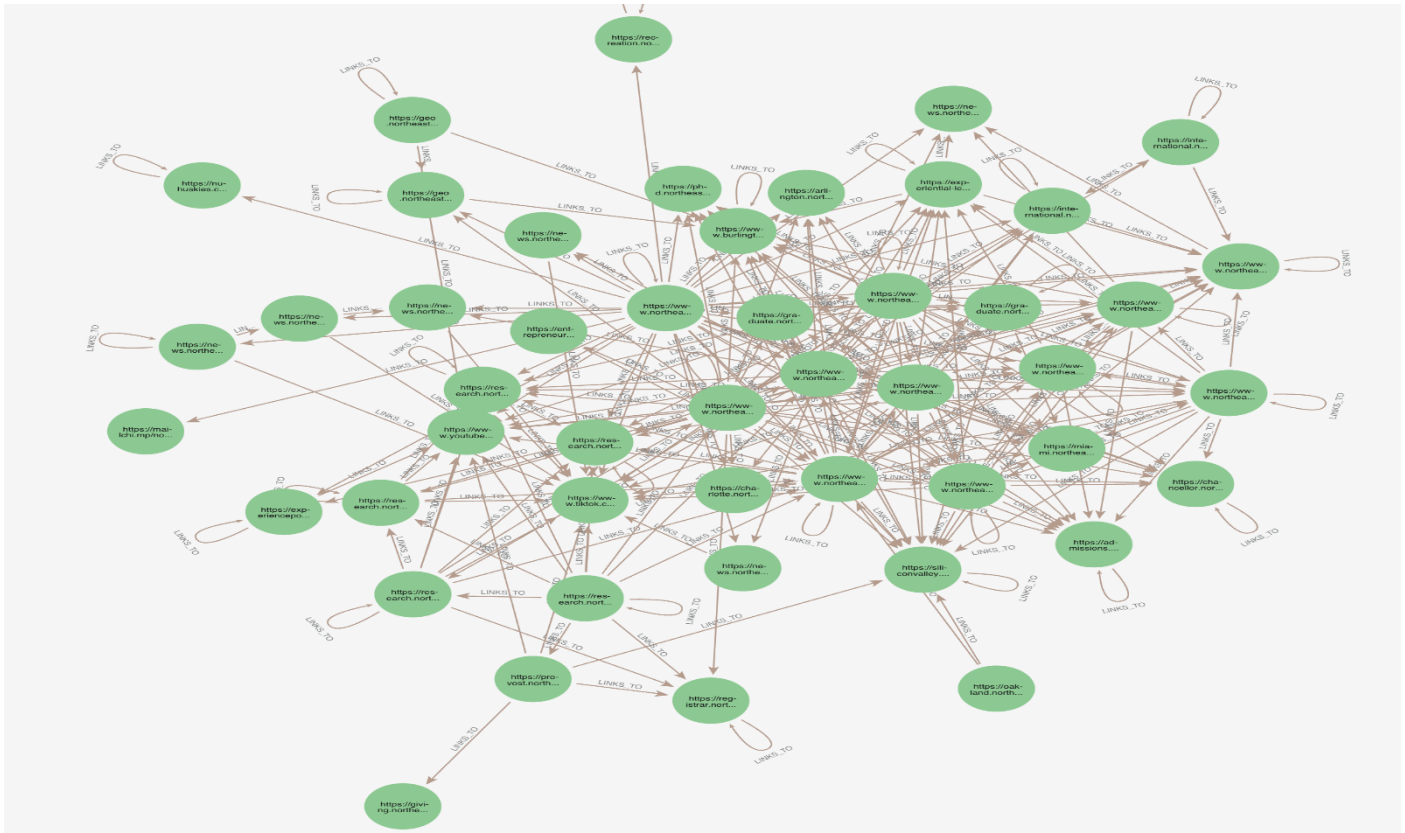
T-2 D-4



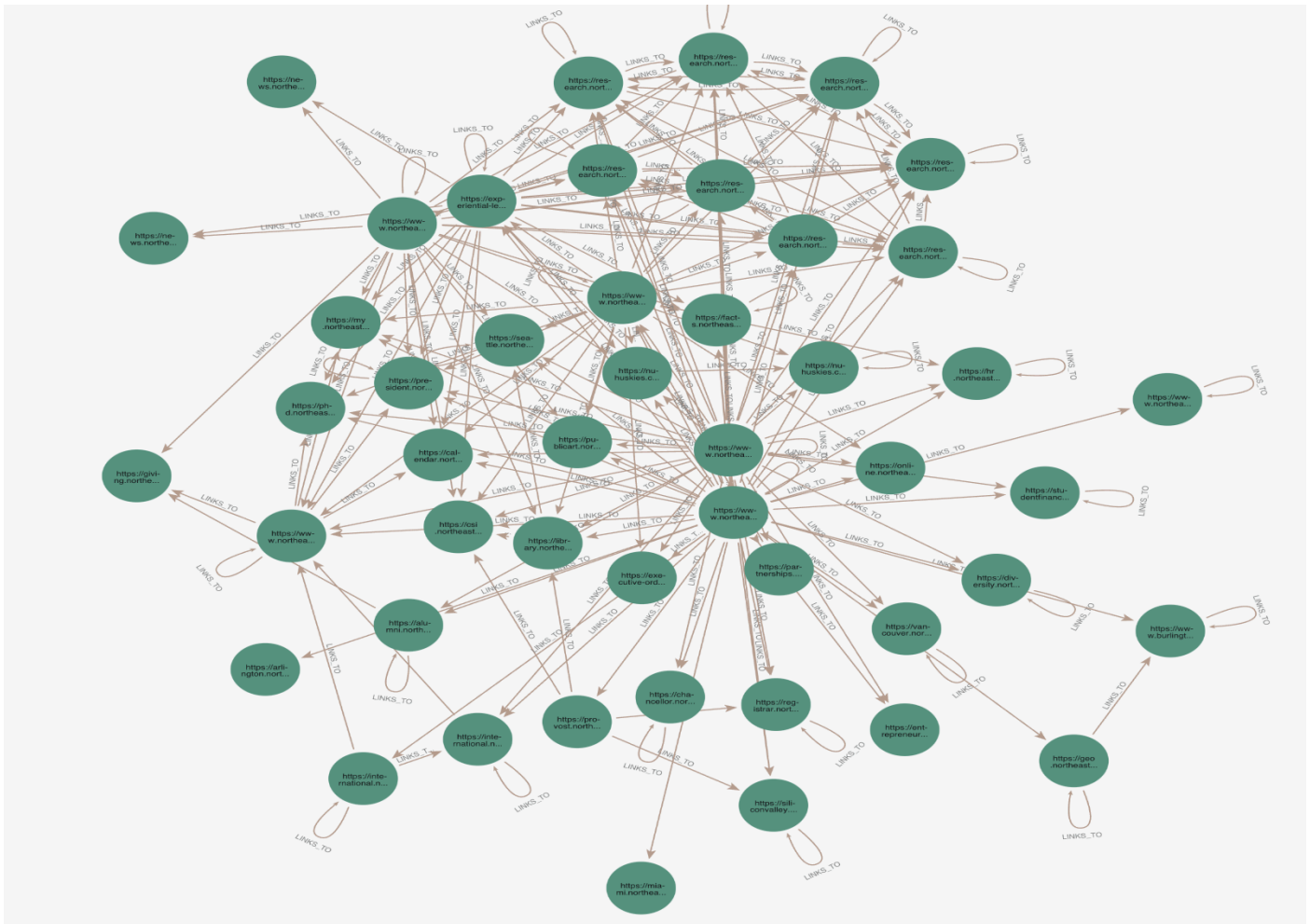
T-2 D-5



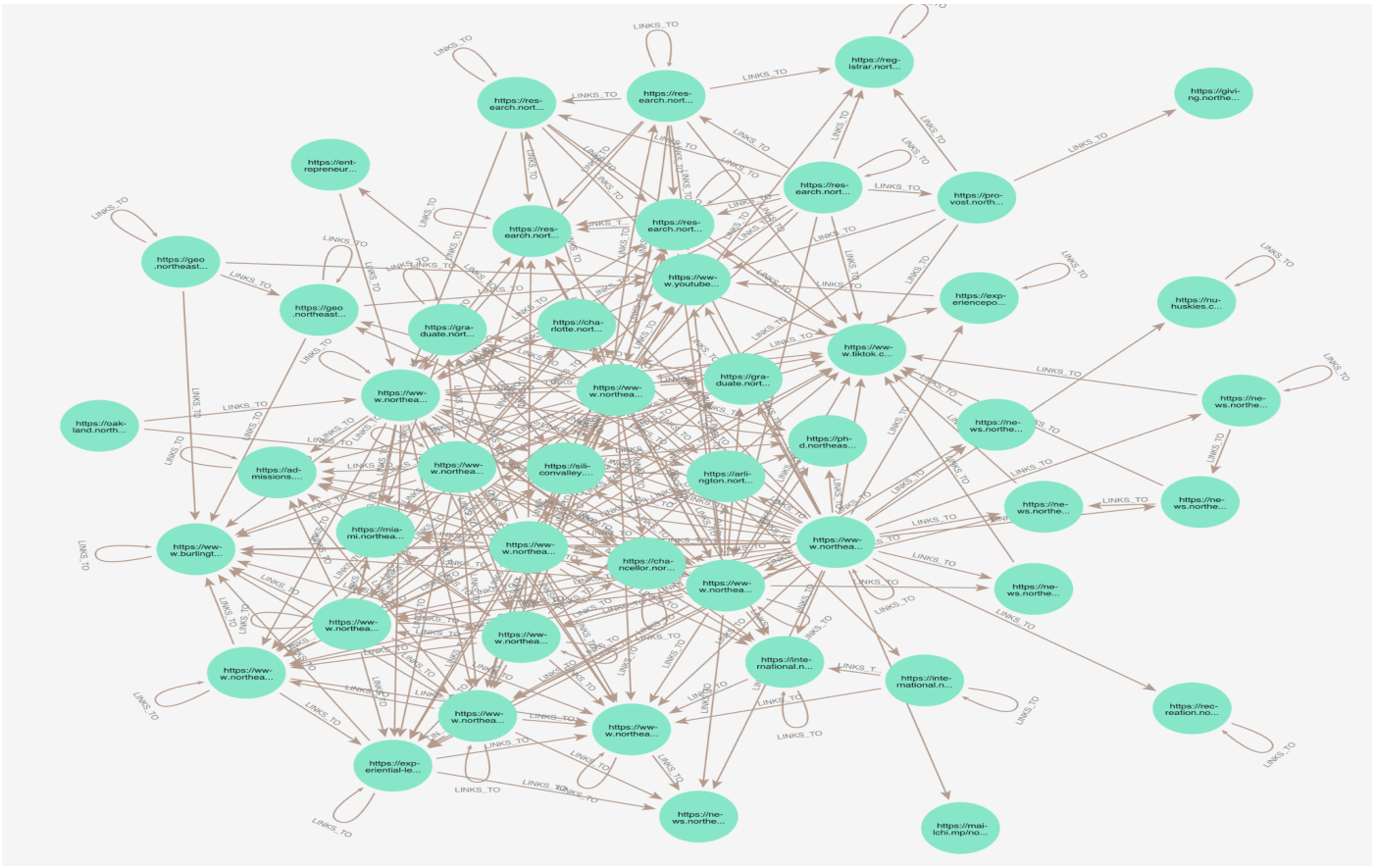
T-4 D-2



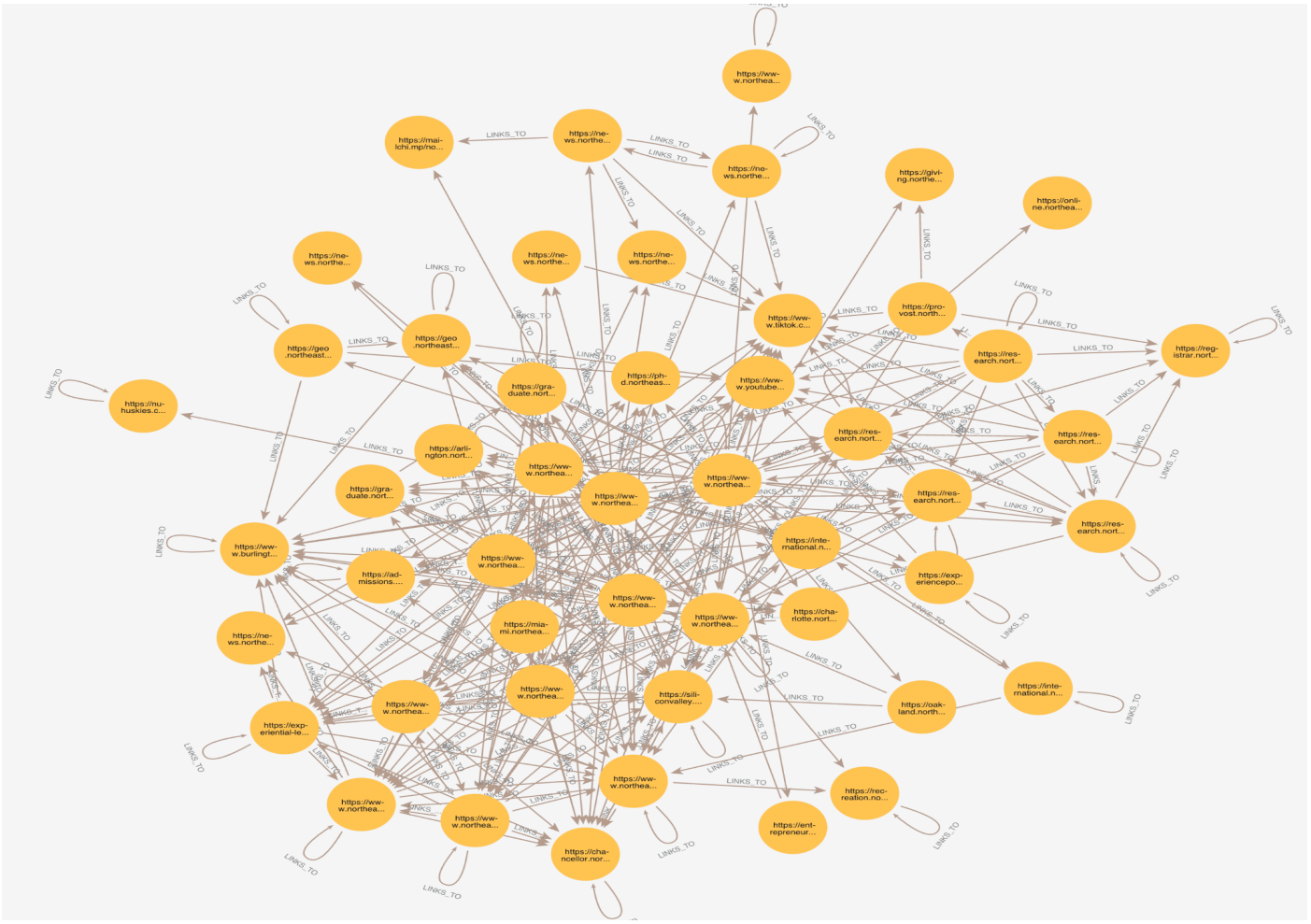
T-4 D-3



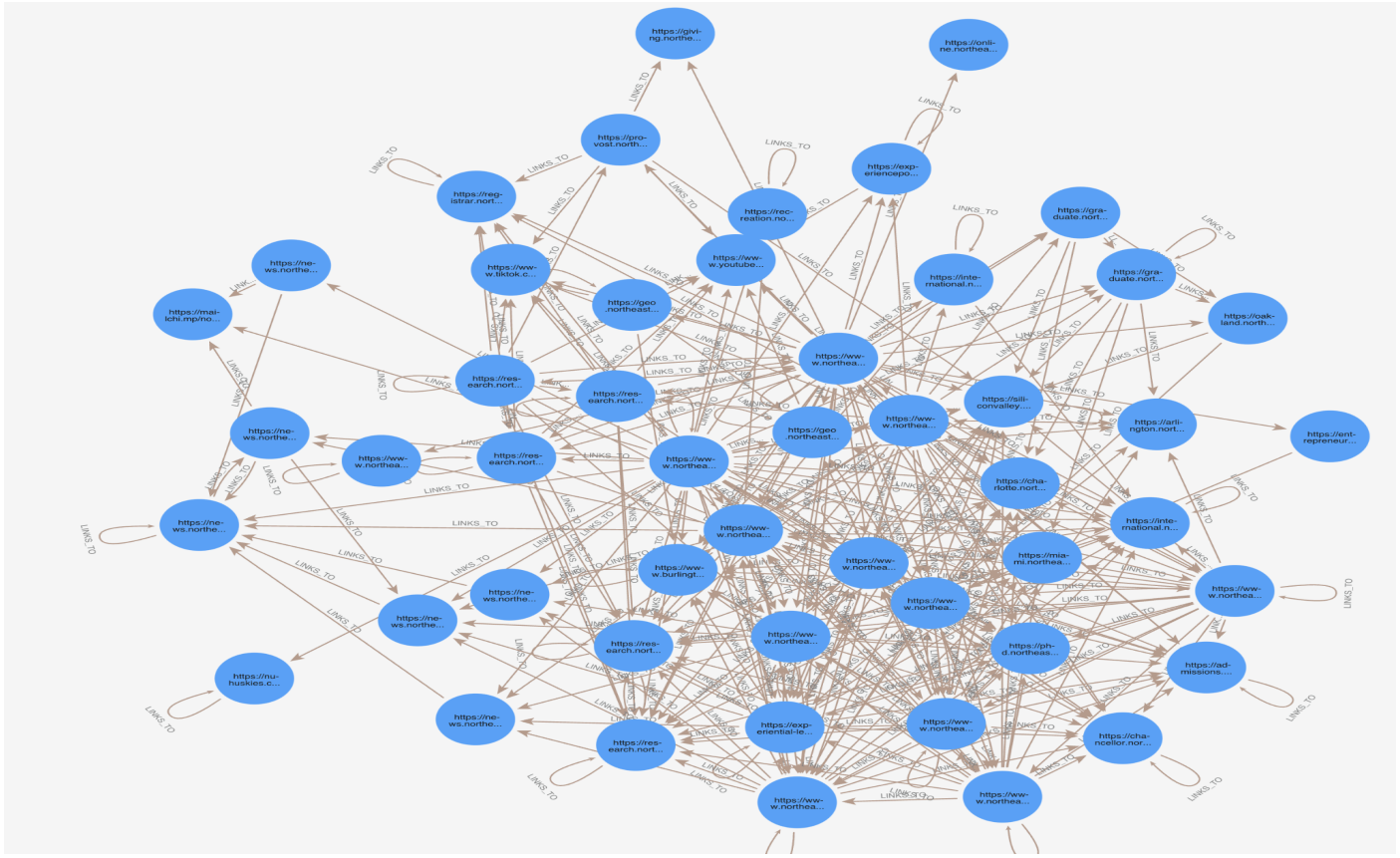
T4-D5



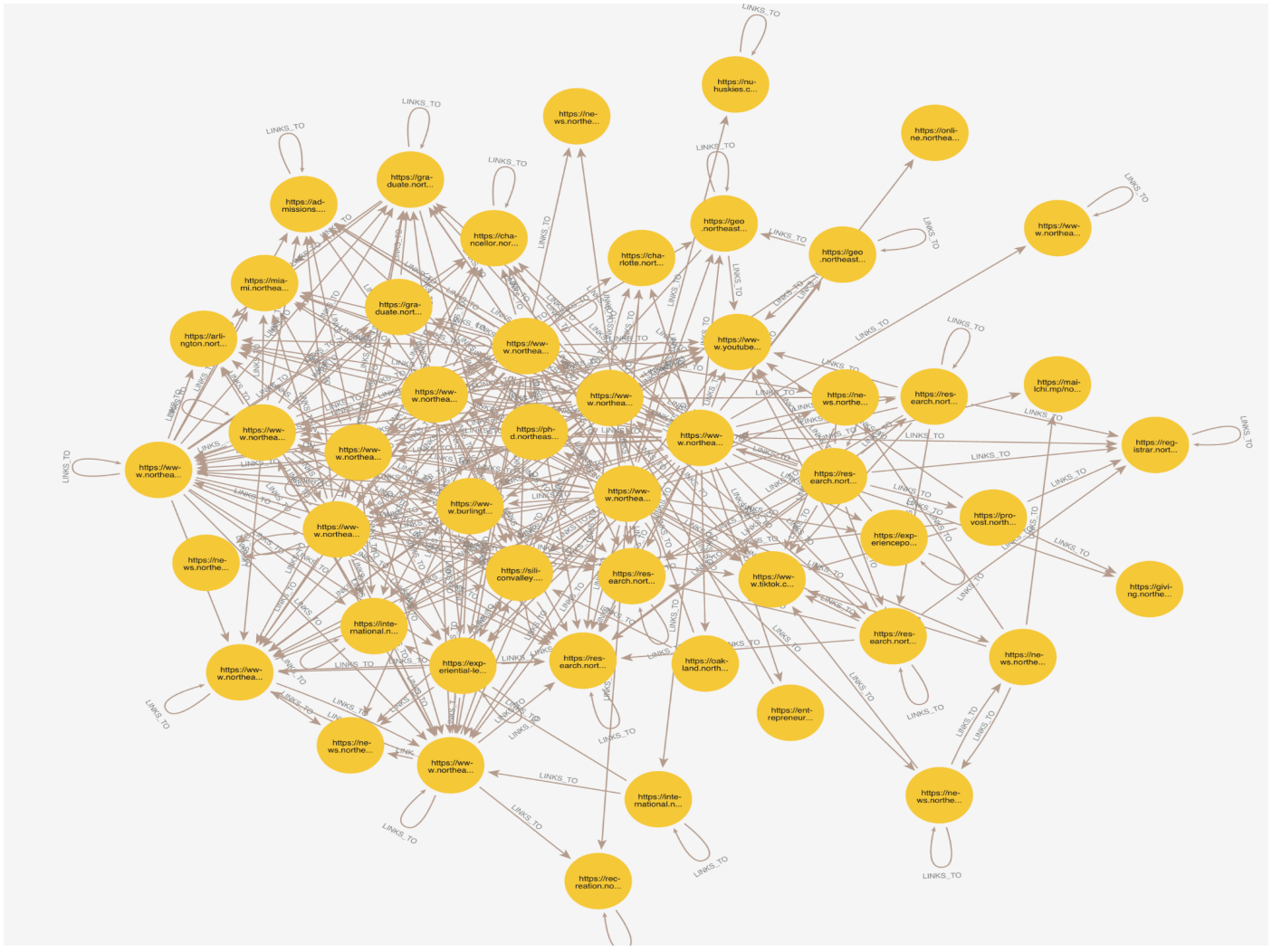
T8-D5



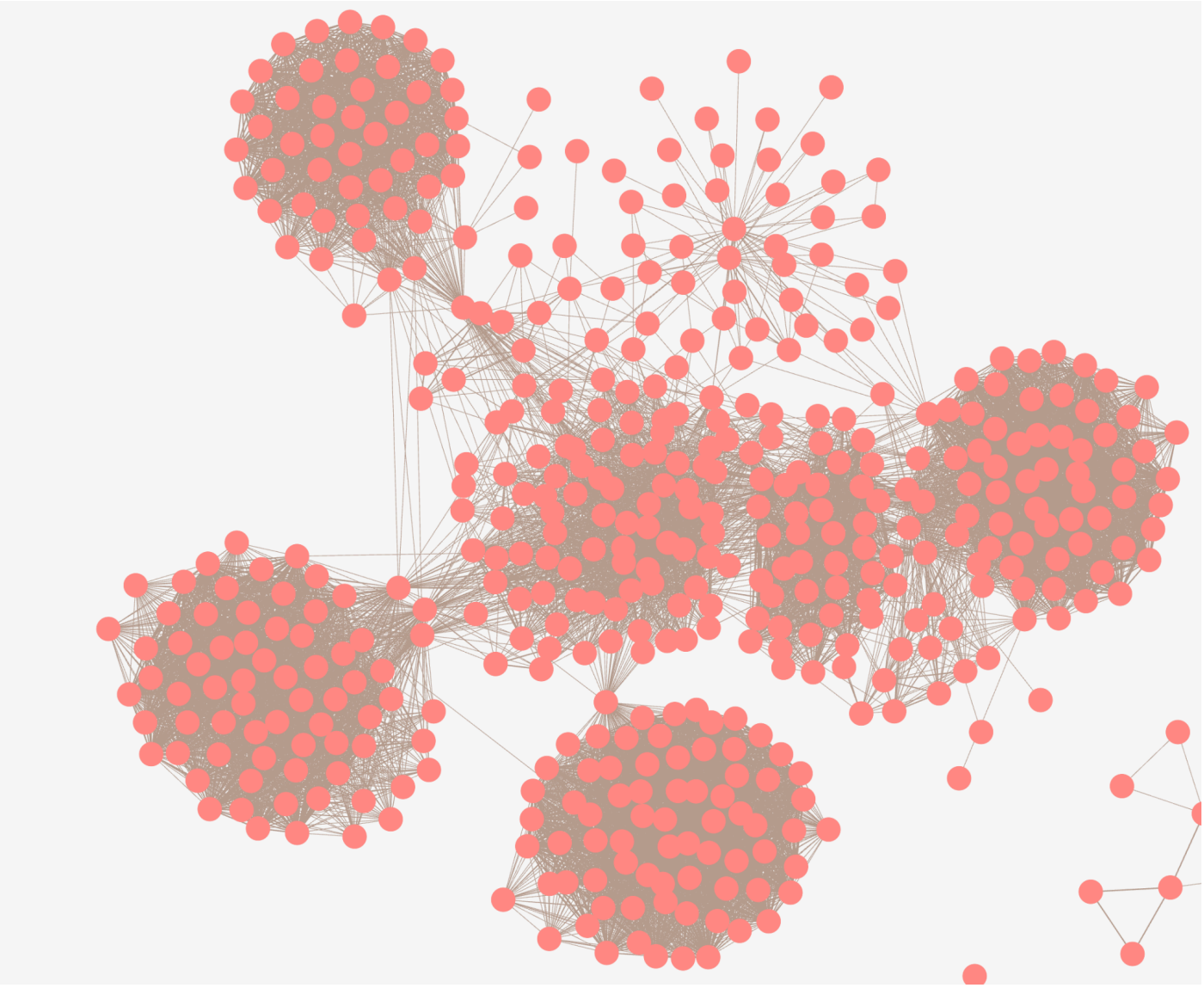
T16-D5



T32-D5

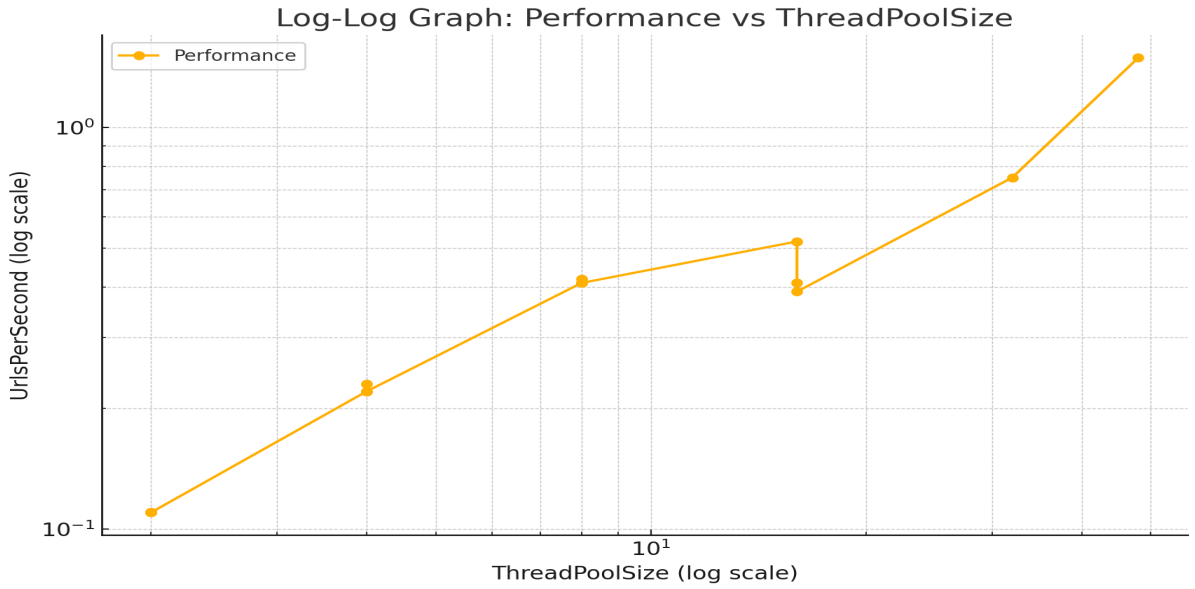


T48-D5 (this is our largest graph with ~500 URLs visited (cut off manually)



We can observe that graphs become progressively denser with increasing crawl depth, as more nodes and edges are added to represent the deeper exploration of the network. Similarly, increasing the thread pool size accelerates the process of graph densification by allowing parallel traversal. The densest and most comprehensive graphs are produced with a combination of high depth and large thread pool sizes, though this comes at the cost of higher resource consumption. Balancing these parameters is key to efficient and effective crawling.

Log-Log plotting



The log-log graph above illustrates the relationship between the thread pool size and the number of URLs crawled per second (UrlsPerSecond) during your web crawler's performance benchmarking.

6. Observations & Conclusion

1. ThreadPoolSize:
Represents the number of threads available for crawling tasks.
As the thread pool size increases (e.g., 2 → 4 → 8 → 16 → 32 → 48), the crawl duration (CrawlDurationMs) generally decreases, leading to an increase in URLs crawled per second (UrlsPerSecond).
Memory usage (PeakMemoryUsedMB) increases significantly with larger thread pools due to higher parallelism and context switching overhead.

2. MaxDepth:
Represents how deep into the link structure the crawler can traverse.
At higher depths (e.g., 3 → 4 → 5), crawl durations tend to increase because of additional URLs being processed or added to the task queue.
Memory usage also increases with depth as more tasks are retained in memory.

3. Performance Trends:
Small Thread Pools (2-4 threads):
UrlsPerSecond is low (around 0.11–0.23), indicating slower processing due to sequential execution and potential network I/O bottlenecks.
Memory usage is modest, but crawl durations are long.

4. Larger Thread Pools (16-32 threads):
Performance improves significantly (UrlsPerSecond rises to 0.75 with 32 threads), showing better utilization of resources.
Memory usage increases sharply (e.g., 247 MB for 32 threads), highlighting the cost of parallelism.

5. Extreme Thread Pools (48 threads):
Maximum performance is achieved (UrlsPerSecond = 1.49), but at the expense of high memory usage (219 MB). There may be diminishing returns due to network bandwidth constraints or thread contention.

Overall, the crawler's scalability improves with thread pool size but faces diminishing returns due to network constraints, rate limiting, and increasing memory usage at higher depths. Optimization opportunities include reducing redundant processing, balancing depth and parallelism, and batching database interactions to enhance performance.

6. Rank vs In-Degree:
The resultant list is sorted based on the **rank**, while this **rank** is closely related to **in-degree**, it is not the same.

As we can see it's possible for a lower in-degree URL to have a higher rank, this is due to the **PageRank** algorithm that assigns weight to the URLs, so a URL with low in-degree but high quality of in-degree **(meaning the URLs linking to this URL are themselves high ranking)** will result in a higher rank compared to another URL with high in-degree but low quality of in-degree.

7. Libraries used:

- Spring Boot Starter Web: Provides a starter for building web applications, including RESTful APIs, using Spring Boot.
- Neo4j OGM Core: Object Graph Mapping (OGM) library for interacting with Neo4j databases.
- Neo4j Java Driver: Official Java driver for connecting to Neo4j databases.
- Jsoup: Library for parsing and manipulating HTML documents.
- Guava: A set of core libraries for Java by Google, offering utilities like collections, caching, and concurrency.
- Spring Dotenv: Supports loading environment variables from .env files in Spring Boot applications.
- Spring Boot Starter Test: Provides testing support for Spring Boot applications, including JUnit and Spring testing utilities.
- Mockito Core: Library for mocking objects in unit tests.
- Apache Commons CSV: A library for parsing and writing CSV files.

8. GUI

Cd frontend> ‘npm install’ and then ‘npm run dev’;
Go to : http://localhost:5173/
Enter:https://www.northeastern.edu/ in WebKrawler Search box.