

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- (WebCrawlerService.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;
}

Backend > src > main > java > com > info6205 > webcrawler > service > WebCrawlerService.java > Language Support for Java(TM) by Red Hat > WebCrawlerService > static
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) {
76         threadPool.shutdown();
77         try {
78             >         catch (InterruptedException e) {
79
80                 performanceTracker.endTracking();
81                 // Print blacklisted and low priority URLs
82                 System.out.println(x:"*****Blacklisted URLs:*****");
83                 blacklistedUrlsList.forEach(System.out::println);
84
85                 System.out.println(x:"*****Low Priority URLs:*****");
86                 lowPriorityUrlsList.forEach(System.out::println);
87
88             }
89             return calculatePageRankResponse();
90         }
91     }
92
93     protected Map<String, Object> calculatePageRankResponse() {
94         List<String> nodes = neo4jService.getNodes();
95         Map<String, List<String>> graph = neo4jService.getGraph();
96         Map<String, Double> pageRanks = pageRankCalculator.computePageRank(nodes, graph);
97         neo4jService.updatePageRank(pageRanks);
98
99         List<Map<String, Object>> resultData = pageRanks.entrySet().stream()
100            .map(entry -> {
101                String url = entry.getKey();
102                Double rank = entry.getValue();
103                int inDegree = (int) nodes.stream().filter(node -> graph.getOrDefault(node, new ArrayList<>()).contains(url))
104                    .collect(Collectors.toList());
105
106                return Map.of(k1:"url", url, k2:"rank", rank, k3:"in_degree", inDegree);
107            })
108            .sorted(Comparator.comparingDouble((Map<String, Object> m) -> (Double) m.get(key:"rank")).reversed()) // Sort by
109            .collect(Collectors.toList());
110
111         return Map.of(
112             k1:"status", v1:"success",
113             k2:"timestamp", ZonedDateTime.now().toString(),
114             k3:"total_urls_crawled", nodes.size(),
115             k4:"data", resultData
116         );
117     }
118
119     public int calculatePriority(String url) {
120
121         // Ignore blacklisted URLs
122         for (String blacklistedUrl : BLACKLISTED_URLS) {
123             if (url.contains(blacklistedUrl)) {
124                 synchronized (blacklistedUrlsList) {
125                     if (!blacklistedUrlsList.contains(url)) {
126                         blacklistedUrlsList.add(url);
127                     }
128                 }
129                 return Integer.MAX_VALUE;
130             }
131         }
132
133         // Ignore dark web URLs
134         if (url.contains(s:".onion")) {
135             return Integer.MAX_VALUE;
136         }
137
138         if (url.matches(regex:"^\\.(jpg|jpeg|png|gif|bmp|mp4|avi|mkv|mov|wmv|flv|webm)$")) {
139             synchronized (lowPriorityUrlsList) {
140                 if (!lowPriorityUrlsList.contains(url)) {
141                     lowPriorityUrlsList.add(url);
142                 }
143             }
144             return 20;
145         }
146
147         You, 3 days ago * Ignoring blacklisted and low-priority URLs in C...
148         if (url.contains(s:"facebook.com") || url.contains(s:"twitter.com") || url.contains(s:"instagram.com") || url.contains(s:
149             synchronized (blacklistedUrlsList) {
150                 if (!blacklistedUrlsList.contains(url)) {
151                     blacklistedUrlsList.add(url);
152                 }
153             }
154         }
155         return 20;
156     }
157 }
```

2.2 Code Output-

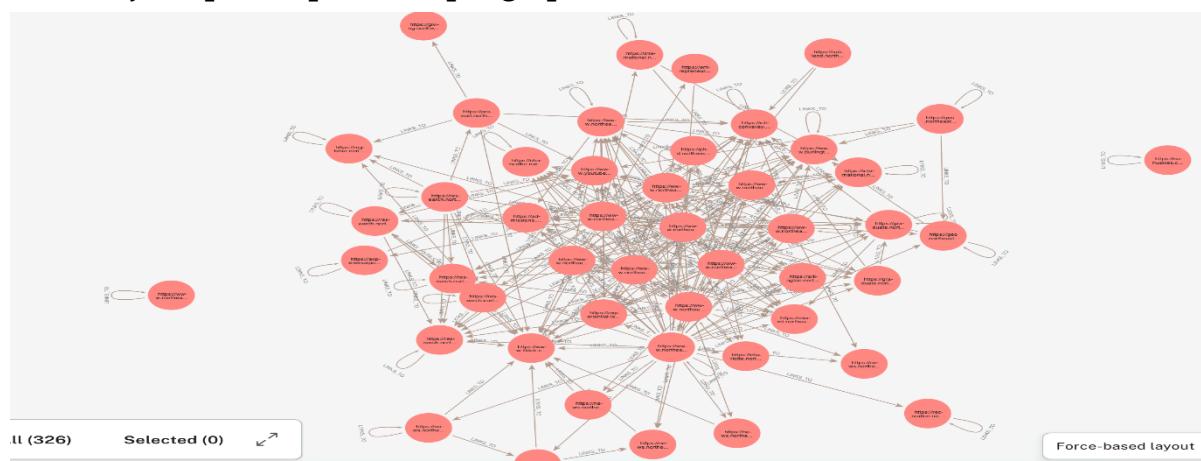
IDE Screenshot showing the terminal output of the WebCrawler application:

```

2024-12-02T17:47:50.521-05:00 INFO 34864 --- [webcrawler] [nio-8080-exec-4] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2024-12-02T17:47:50.626-05:00 INFO 34864 --- [webcrawler] [nio-8080-exec-4] c.l.w.service.WebCrawlerService : Starting web crawl with starting URL: https://www.northeastern.edu/
***** Processing URL ***** https://www.northeastern.edu/
***** Processing URL ***** https://siliconvalley.northeastern.edu/
***** Processing URL ***** https://chancellor.northeastern.edu/
***** Processing URL ***** https://research.northeastern.edu/sustainability/
***** Processing URL ***** https://recreation.northeastern.edu/
***** Processing URL ***** https://news.northeastern.edu/2024/11/22/spirit-airlines-bankruptcy-filing/?utm_source=northeastern.edu&utm_medium=website&utm_campaign=northeastern.edu+homepage&utm_id=northeastern.edu+homepage
***** Processing URL ***** https://news.northeastern.edu/2024/11/26/harold-woods-northeastern-basketball/?utm_source=northeastern.edu&utm_medium=website&utm_campaign=northeastern.edu+homepage&utm_id=northeastern.edu+homepage
***** Processing URL ***** https://miami.northeastern.edu/
***** Processing URL ***** https://research.northeastern.edu/ug-research/
***** Processing URL ***** https://news.northeastern.edu/back-to-school-2024/
***** Processing URL ***** https://international.northeastern.edu/
***** Processing URL ***** https://twitter.com/Northeastern
***** blacklisted URLs:*****
https://www.instagram.com/northeastern/
https://nu.outsystemsenterprise.com/fsd/
https://www.tiktok.com/@northeasternu
https://www.youtube.com/user/Northeastern
https://www.linkedin.com/school/northeastern-university/
https://www.facebook.com/northeastern/
https://twitter.com/Northeastern
https://twitter.com/intent/tweet?text=Meet team-minded Harold Woods, the ?cool uncle? behind Northeastern's early success in men's basketball https://news.northeastern.edu/2024/11/26/harold-woods-northeastern-basketball/ via Northeastern Global News
https://www.reddit.com/submit?url=https://news.northeastern.edu/2024/11/26/harold-woods-northeastern-basketball/&title=Meet team-minded Harold Woods, the ?cool uncle? behind Northeastern's early success in men's basketball
https://www.facebook.com/northeastern
https://www.instagram.com/northeasternglobalnews
https://www.facebook.com/sharer.php?url=https://news.northeastern.edu/2024/11/26/harold-woods-northeastern-basketball/
https://twitter.com/northeastern
https://www.youtube.com/user/Northeastern/featured
https://twitter.com/IanathNU
https://www.linkedin.com/shareArticle?url=https://news.northeastern.edu/2024/11/26/harold-woods-northeastern-basketball/&title=Meet team-minded Harold Woods, the ?cool uncle? behind Northeastern's early success in men's basketball
https://www.instagram.com/northeasternrec/?utm_source=ig_embed&utm_campaign=loading
https://www.tiktok.com/@northeasternrec?refer=creator_embed

(total_urls_crawled=11, status=success, data=[{"in_degree=2, rank=0.1754377103600576, url=https://international.northeastern.edu/}, {"in_degree=2, rank=0.1754377103600576, url=https://recreation.northeastern.edu/}, {"in_degree=2, rank=0.1754377103600576, url=https://chancellor.northeastern.edu/}, {"in_degree=3, rank=0.17543771036005748, url=https://research.northeastern.edu/sustainability/}, {"in_degree=3, rank=0.17543771036005748, url=https://research.northeastern.edu/heALTH/}, {"in_degree=1, rank=0.12280676349346335, url=https://siliconvalley.northeastern.edu/}, {"in_degree=3, rank=2.640122117794483E-6, url=https://www.northeastern.edu/}, {"in_degree=1, rank=5.111460093020647E-7, url=https://news.northeastern.edu/2024/11/26/harold-woods-northeastern-basketball/?utm_source=northeastern.edu&utm_medium=website&utm_campaign=northeastern.edu+homepage&utm_id=northeastern.edu+homepage}, {"in_degree=1, rank=5.111460093020647E-7, url=https://miami.northeastern.edu/}, {"in_degree=1, rank=5.111460093020647E-7, url=https://washington.northeastern.edu/}], timestamp=2024-12-02T17:51:57.994665100-05:00[America/New_York]]
```

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



2.4 API response (Postman)-

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8080/api/crawler/start?startUrl=https://www.northeastern.edu/`. The response status is 200 OK, time taken is 1m 5.58s, and size is 2.48 KB. The response body is a JSON object:

```
1 "timestamp": "2024-11-29T22:04:20.354279200-05:00[America/New_York]",  
2 "total_urls_crawled": 21,  
3 "status": "success",  
4 "data": [  
5     {  
6         "url": "https://siliconvalley.northeastern.edu/",  
7         "in_degree": 5,  
8         "rank": 0.34229651573187436  
9     },  
10    {  
11        "url": "https://research.northeastern.edu/sustainability/",  
12        "in_degree": 6,  
13        "rank": 0.26073660838891094  
14    },  
15    {  
16        "url": "https://recreation.northeastern.edu/",  
17        "in_degree": 2,  
18        "rank": 0.23123086278697003  
19    },  
20    {  
21        "url": "https://chancellor.northeastern.edu/",  
22        "in_degree": 3,  
23        "rank": 0.09205976671523351  
24    }]
```

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-INF06205.git
```

Navigate to Directory:

```
cd backend
```

Create a .env file in /backend

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

```
NE04J_URL=neo4j+s://9d807b45.databases.neo4j.io  
NE04J_USERNAME=neo4j  
NE04J_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:

The screenshot shows the Eclipse IDE interface with the following details:

- TEST EXPLORER:** Shows a tree view of test cases under the Java category. Tests include TestControllerTest, WebCrawlerControllerTest, Neo4jServiceTest, PageRankCalculatorTest, WebCrawlerServiceTest, and WebcrawlerApplicationTests.
- PROBLEMS:** Shows 6 errors.
- OUTPUT:** Displays the command-line output of the tests. It includes logs from Spring Boot (INFO messages) and the test runner (JUnit).
- DEBUG CONSOLE:** Shows the Java Single Debug configuration.
- TERMINAL:** Shows the command-line interface.
- TEST RESULTS:** Shows the results of the tests, indicating 1 test run, 0 failures, 0 errors, and 0 skipped.
- PORTS:** Shows port information.
- COMMENTS:** Shows comments.
- GITLENS:** Shows git-related information.
- POSTMAN CONSOLE:** Shows Postman console output.
- Test Runner for Java:** Shows the results of the test runner, including the successful execution of the testCreateNode_Success() method.

4.1 Neo4j Tests

4.1.1 TestCreateNode()

The screenshot shows the Eclipse IDE interface with the following details:

- TEST EXPLORER:** Shows a tree view of test cases under the Java category. Tests include TestControllerTest, WebCrawlerControllerTest, Neo4jServiceTest, PageRankCalculatorTest, WebCrawlerServiceTest, and WebcrawlerApplicationTests.
- PROBLEMS:** Shows 6 errors.
- OUTPUT:** Displays the command-line output of the tests. It includes logs from Spring Boot (INFO messages) and the test runner (JUnit).
- DEBUG CONSOLE:** Shows the Java Single Debug configuration.
- TERMINAL:** Shows the command-line interface.
- TEST RESULTS:** Shows the results of the tests, indicating 1 test run, 0 failures, 0 errors, and 0 skipped.
- PORTS:** Shows port information.
- COMMENTS:** Shows comments.
- GITLENS:** Shows git-related information.
- POSTMAN CONSOLE:** Shows Postman console output.
- Test Runner for Java:** Shows the results of the test runner, indicating success for the testCreateNode_Success() method.

4.1.2 TestAddEdge_Sucess()

The screenshot shows the Eclipse IDE interface with the following details:

- TEST EXPLORER:** Shows a tree view of test cases under the Java category. Tests include TestControllerTest, WebCrawlerControllerTest, Neo4jServiceTest, PageRankCalculatorTest, WebCrawlerServiceTest, and WebcrawlerApplicationTests.
- PROBLEMS:** Shows 6 errors.
- OUTPUT:** Displays the command-line output of the tests. It includes logs from Spring Boot (INFO messages) and the test runner (JUnit).
- DEBUG CONSOLE:** Shows the Java Single Debug configuration.
- TERMINAL:** Shows the command-line interface.
- TEST RESULTS:** Shows the results of the tests, indicating 1 test run, 0 failures, 0 errors, and 0 skipped.
- PORTS:** Shows port information.
- COMMENTS:** Shows comments.
- GITLENS:** Shows git-related information.
- POSTMAN CONSOLE:** Shows Postman console output.
- Test Runner for Java:** Shows the results of the test runner, indicating success for the testAddEdge_Success() method.

4.1.3 TestGetNodes_Sucess()

The screenshot shows the VS Code interface with the 'TEST EXPLORER' view open. The 'TEST RESULTS' tab is selected. The output window displays the following test results:

```
%TESTC 1 v2
%STTRE2,com.info6205.webcrawler.service.Neo4jServiceTest,true,1,false,1,Neo4jServiceTest,[engine:junit-jupiter]/[class:com.info6205.webcrawler.service.Neo4jServiceTest]
%STTRE3,testGetNodes_Success(com.info6205.webcrawler.service.Neo4jServiceTest),false,1,false,2,testGetNodes_Success(),,[engine:junit-jupiter]/[class:com.info6205.webcrawler.service.Neo4jServiceTest]/[method:testGetNodes_Success()]
%TESTS 3,testGetNodes_Success(com.info6205.webcrawler.service.Neo4jServiceTest)
%TESTE 3,testGetNodes_Success(com.info6205.webcrawler.service.Neo4jServiceTest)
%RUNTIME1656
```

4.1.4 TestUpdatePageRank_Success()

The screenshot shows the VS Code interface with the 'TEST EXPLORER' view open. The 'TEST RESULTS' tab is selected. The output window displays the following test results:

```
%TESTC 1 v2
%STTRE3,testUpdatePageRank_Success(com.info6205.webcrawler.service.Neo4jServiceTest),false,1,false,2,testUpdatePageRank_Success(),,[engine:junit-jupiter]/[class:com.info6205.webcrawler.service.Neo4jServiceTest]/[method:testUpdatePageRank_Success()]
%TESTS 3,testUpdatePageRank_Success(com.info6205.webcrawler.service.Neo4jServiceTest)
%TESTE 3,testUpdatePageRank_Success(com.info6205.webcrawler.service.Neo4jServiceTest)
%RUNTIME1556
```

4.2. PageRankCalculatorTest

4.2.1 TestComputePageRank()

The screenshot shows the VS Code interface with the 'TEST EXPLORER' view open. The 'TEST RESULTS' tab is selected. The output window displays the following test results:

```
%TESTC 1 v2
%STTRE2,com.info6205.webcrawler.service.PageRankCalculatorTest,true,1,false,1,PageRankCalculatorTest,[engine:junit-jupiter]/[class:com.info6205.webcrawler.service.PageRankCalculatorTest]
%STTRE3,testComputePageRank(com.info6205.webcrawler.service.PageRankCalculatorTest),false,1,false,2,testComputePageRank(),,[engine:junit-jupiter]/[class:com.info6205.webcrawler.service.PageRankCalculatorTest]/[method:testComputePageRank()]
%TESTS 3,testComputePageRank(com.info6205.webcrawler.service.PageRankCalculatorTest)
%TESTE 3,testComputePageRank(com.info6205.webcrawler.service.PageRankCalculatorTest)
%RUNTIME108
```

4.2.2 TestGetOutDegree()

4.3. WebCrawlerServices

4.3.1 TestStartCrawl()

The screenshot shows the Visual Studio Code interface with several open panes:

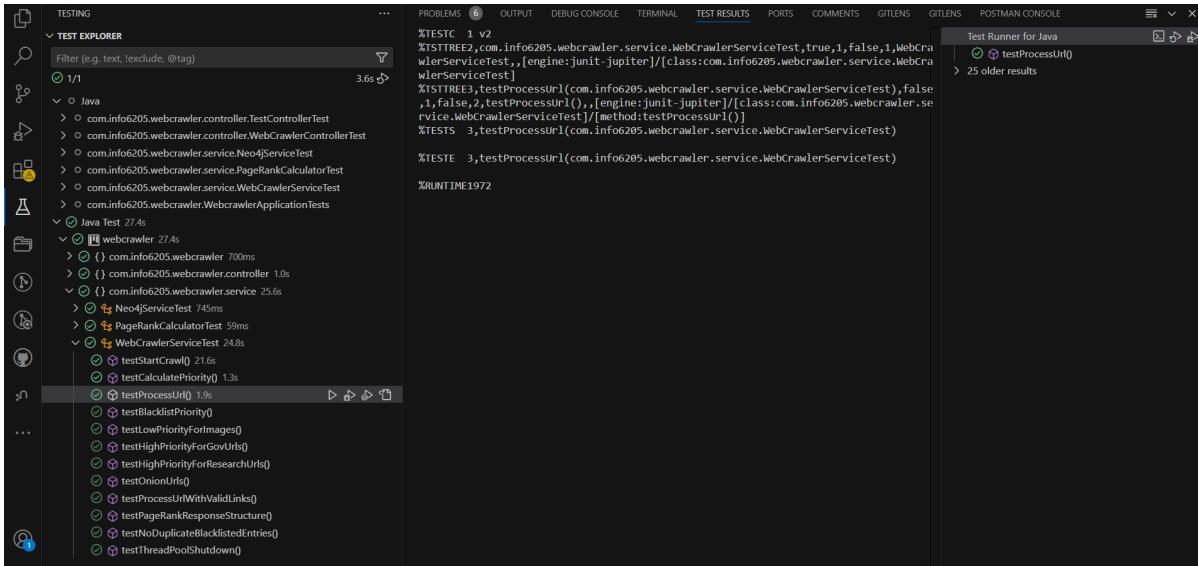
- TEST EXPLORER**: Shows a tree view of test cases. The root node is expanded, showing Java tests, Java test (242), and a detailed view of the webcrawler service test (21.6s). This detailed view includes methods like testStartCrawl(), testCalculatePriority(), and testProcessUrl().
- PROBLEMS**: Shows 6 errors.
- OUTPUT**: Shows the output of the test runner.
- DEBUG CONSOLE**: Shows the output of the debugger.
- TERMINAL**: Shows the output of the terminal.
- TEST RESULTS**: Shows the results of the tests.
- PORTS**: Shows network port information.
- COMMENTS**: Shows code comments.
- GITLENS**: Shows git lens information.
- GITLENS**: Shows git lens information.
- POSTMAN CONSOLE**: Shows the output of the Postman console.
- Test Runner for Java**: A sidebar showing the current test configuration: `testStartCrawl()`. It also lists 23 older results.

4.3.2 TestCalculatePriority()

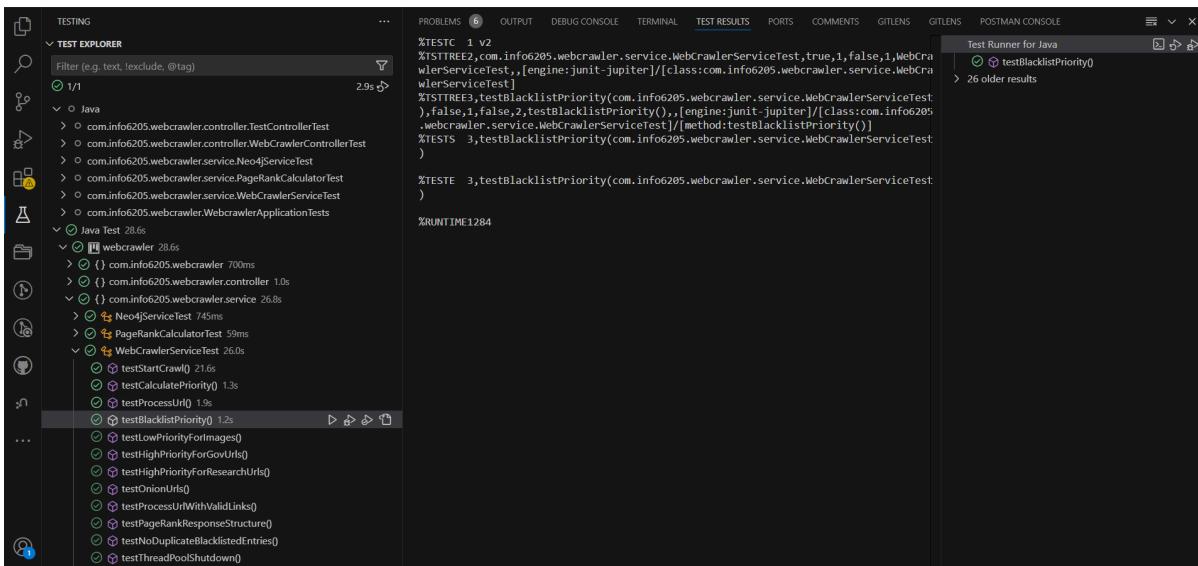
The screenshot shows the Visual Studio Code interface with the following details:

- SIDE BAR:** Includes icons for File, Edit, View, Insert, Search, Find, Replace, Go To, and Help.
- TEST EXPLORER:** Shows a tree view of test cases. A search bar at the top says "Filter (e.g. text, exclude, @tag)".
 - Java:** Contains tests for TestControllerTest, WebCrawlerControllerTest, Neo4jServiceTest, PageRankCalculatorTest, and WebCrawlerServiceTest.
 - Java Test:** Contains a single test named "25.ms".
 - webcrawler:** Contains tests for 700ms, controller (1.0s), service (23.7s), Neo4jServiceTest (745ms), PageRankCalculatorTest (59ms), and WebCrawlerServiceTest (22.9s). The WebCrawlerServiceTest node has several child tests listed below it.
- TEST RESULTS:** The active tab, showing the results of the last run:
 - TESTS:** 6 total, 6 passed, 0 failed, 0 skipped.
 - TEST LIST:** A list of 24 older results, with the most recent being "testCalculatePriority" from "com.info6205.webcrawler.service.WebCrawlerServiceTest".
- PROBLEMS:** Shows 6 errors.
- OUTPUT, DEBUG CONSOLE, TERMINAL:** Standard VS Code toolbars.
- GIT LENS:** Shows file history and blame information.
- POSTMAN CONSOLE:** Shows API requests and responses.
- TEST RUNNER FOR JAVA:** A separate panel showing the results of the Java test run.

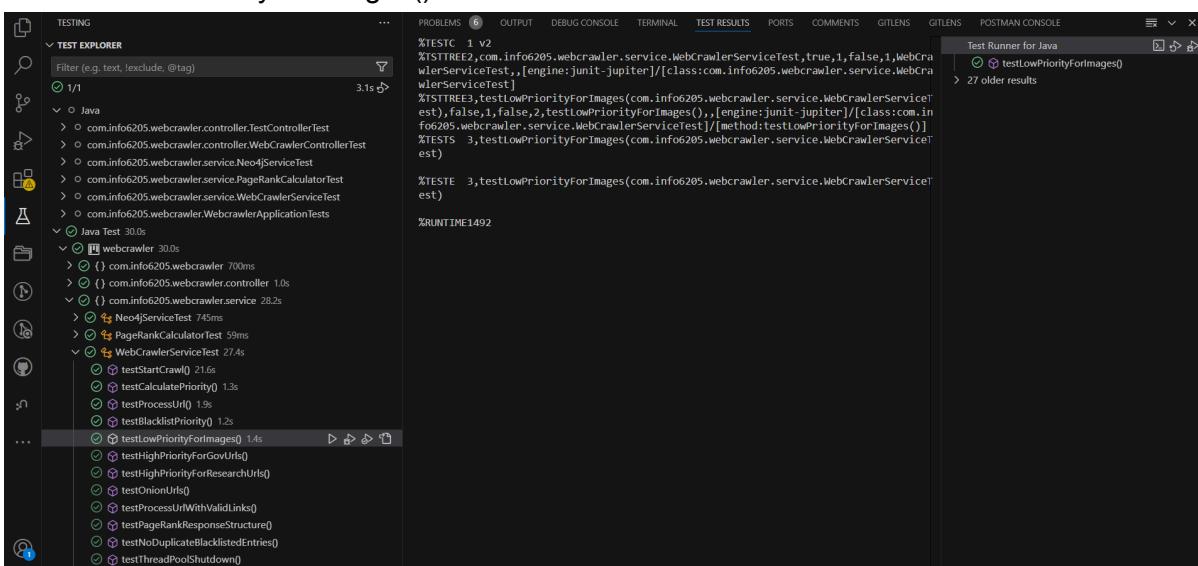
4.3.3 TestProcessURL()



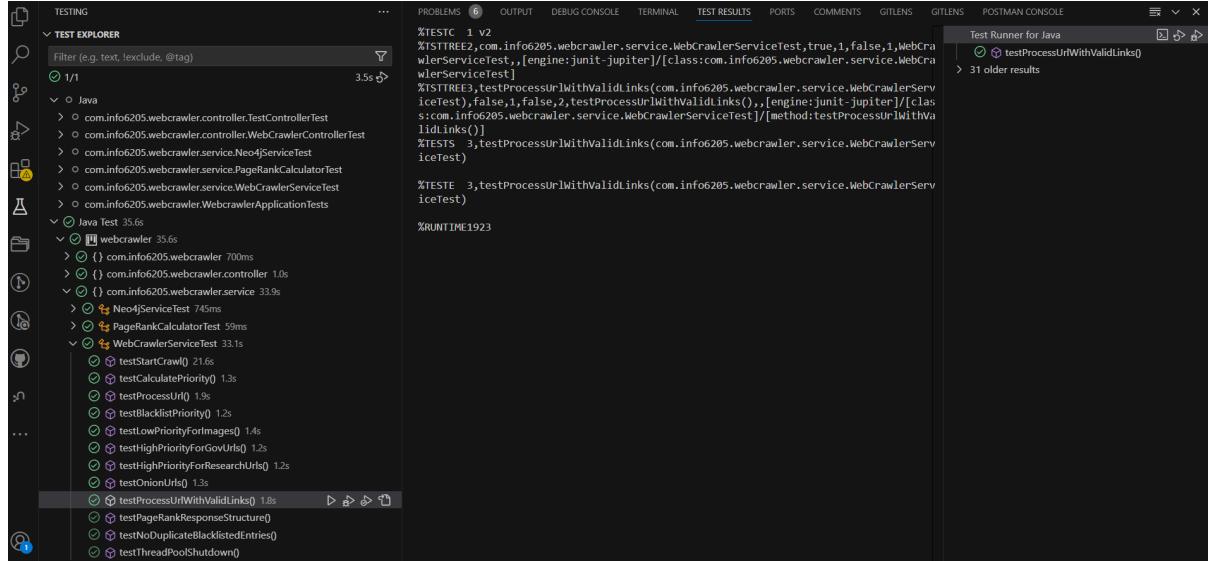
4.3.4 TestBlackListPriority()



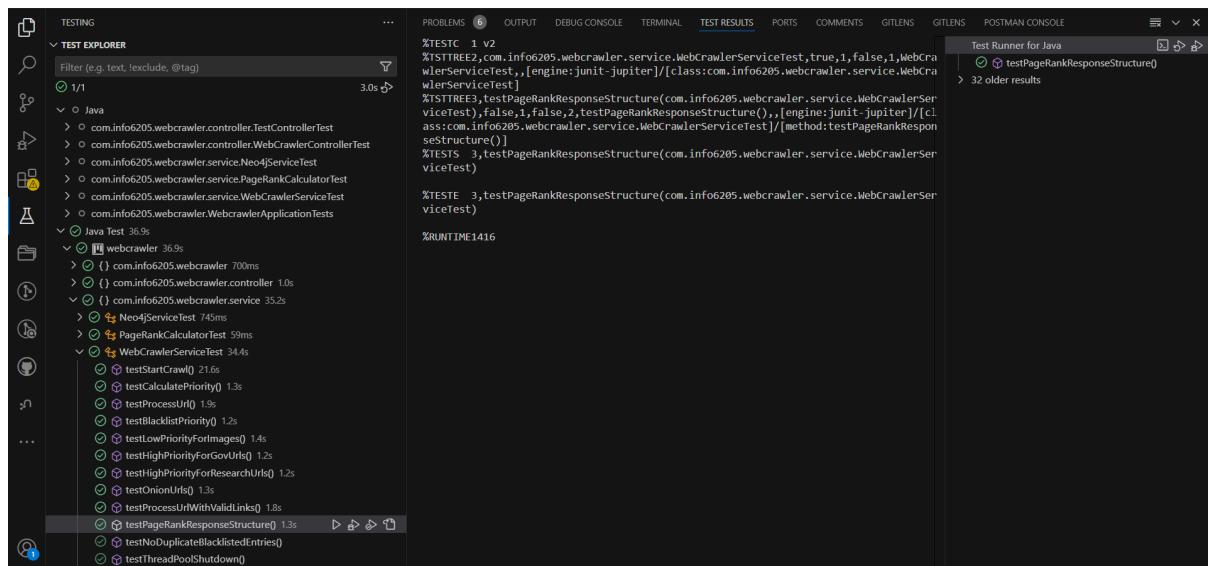
4.3.4 TestLowPriorityForImages()



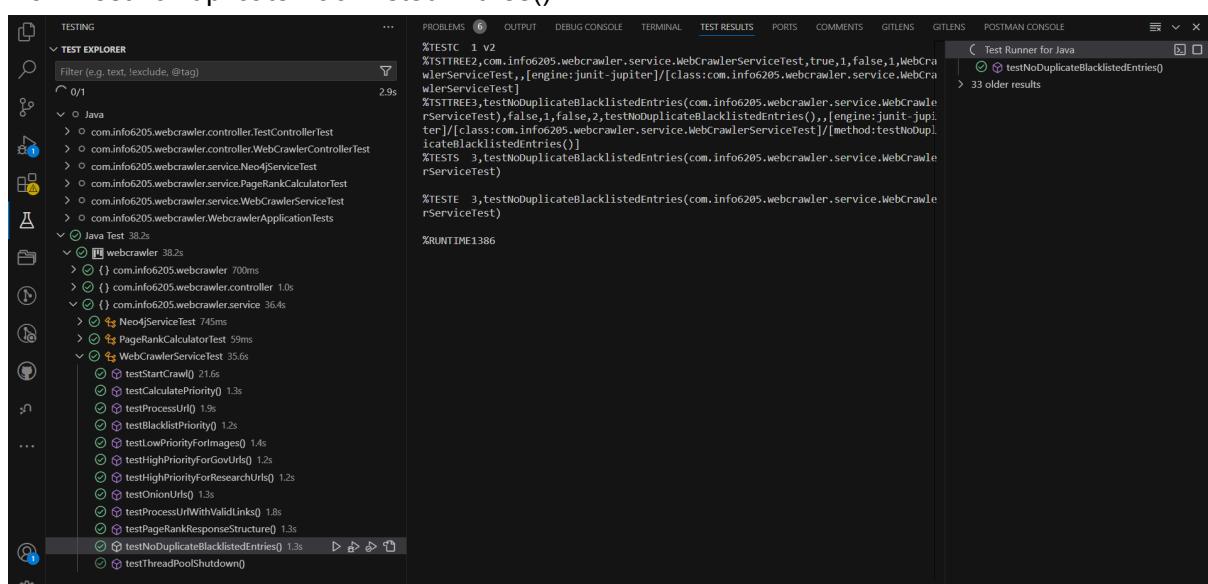
4.3.5 TestProcessURLWithValidLinks()



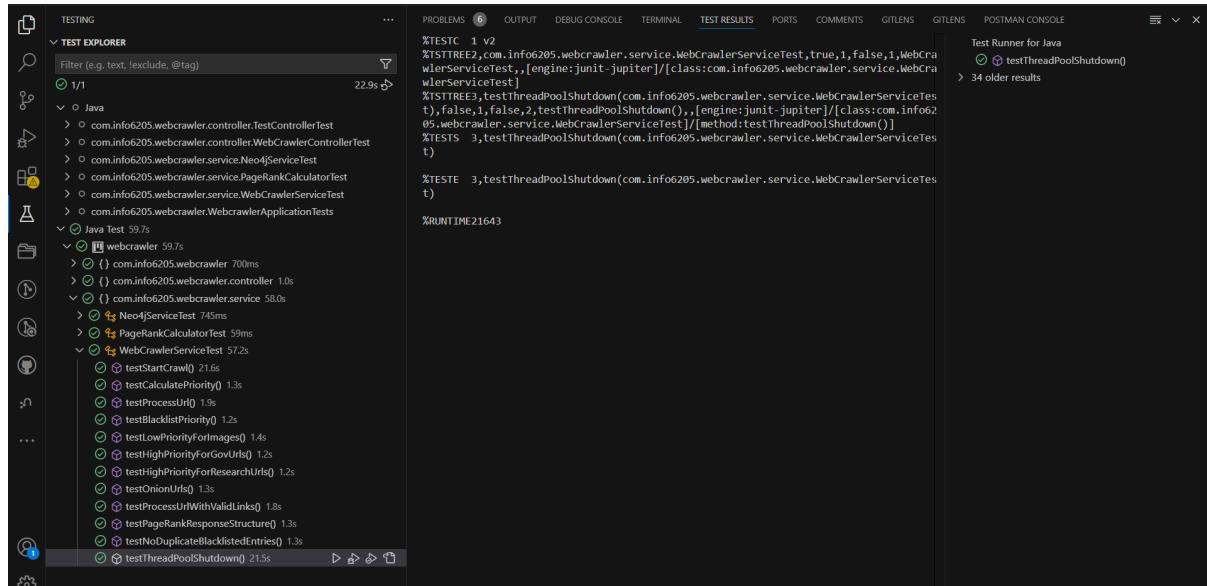
4.3.6 TestPageRankResponseStructure()



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

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

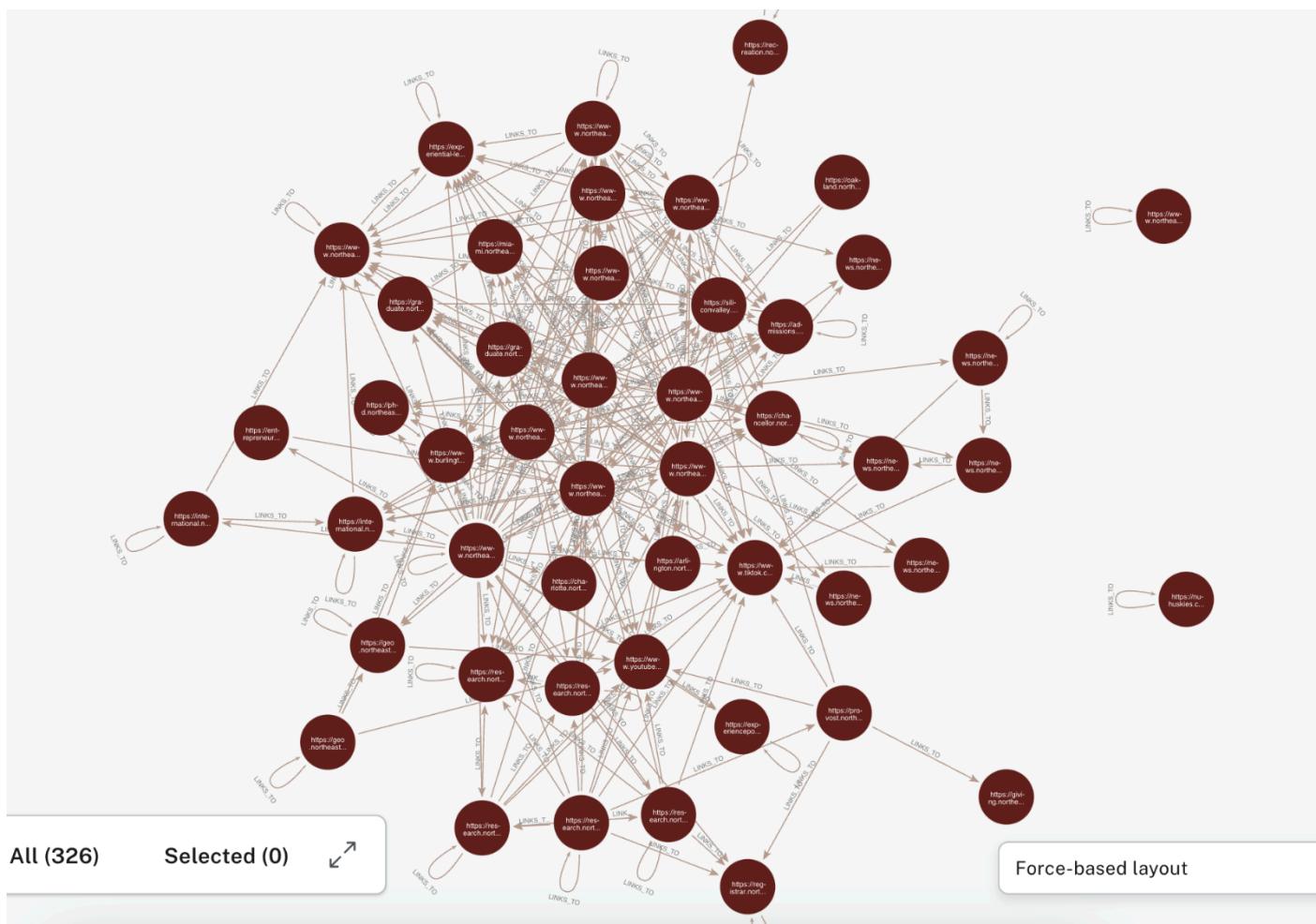
Here are the same results, tabulated.

StartTimestamp	StartUrl	ThreadPoc	MaxDepth	TotalUrlsC	CrawlDura	UrlsPerSec	PeakMemoryUsedMB
2024-12-02T00:52:05.915695Z	https://www.northeastern.edu	2	2	51	465886	0.11	62
2024-12-02T01:37:02.128431Z	https://www.northeastern.edu	2	3	51	456454	0.11	65
2024-12-02T01:51:04.539164Z	https://www.northeastern.edu	2	4	51	458618	0.11	68
2024-12-02T02:12:04.739304Z	https://www.northeastern.edu	2	5	51	472200	0.11	65
2024-12-02T02:25:04.98994Z	https://www.northeastern.edu	4	2	51	235077	0.22	76
2024-12-02T02:32:04.221376Z	https://www.northeastern.edu	4	3	51	221376	0.23	73
2024-12-02T02:39:04.233710Z	https://www.northeastern.edu	4	4	51	233710	0.22	76
2024-12-02T02:45:04.234324Z	https://www.northeastern.edu	4	5	51	234324	0.22	70
2024-12-02T03:04:08.123057Z	https://www.northeastern.edu	8	2	51	123057	0.41	96
2024-12-02T03:15:08.122178Z	https://www.northeastern.edu	8	3	51	122178	0.42	94
2024-12-02T03:20:08.123859Z	https://www.northeastern.edu	8	4	51	123859	0.41	92
2024-12-02T02:50:08.124860Z	https://www.northeastern.edu	8	5	51	124860	0.41	99
2024-12-02T03:25:016.98994Z	https://www.northeastern.edu	16	2	51	98994	0.52	142
2024-12-02T03:29:016.123637Z	https://www.northeastern.edu	16	3	51	123637	0.41	114
2024-12-02T03:33:016.131466Z	https://www.northeastern.edu	16	4	51	131466	0.39	145
2024-12-02T03:37:032.67836Z	https://www.northeastern.edu	32	2	51	67836	0.75	247
2024-12-02T03:47:048.335451Z	https://www.northeastern.edu	48	5	51	335451	1.49	219

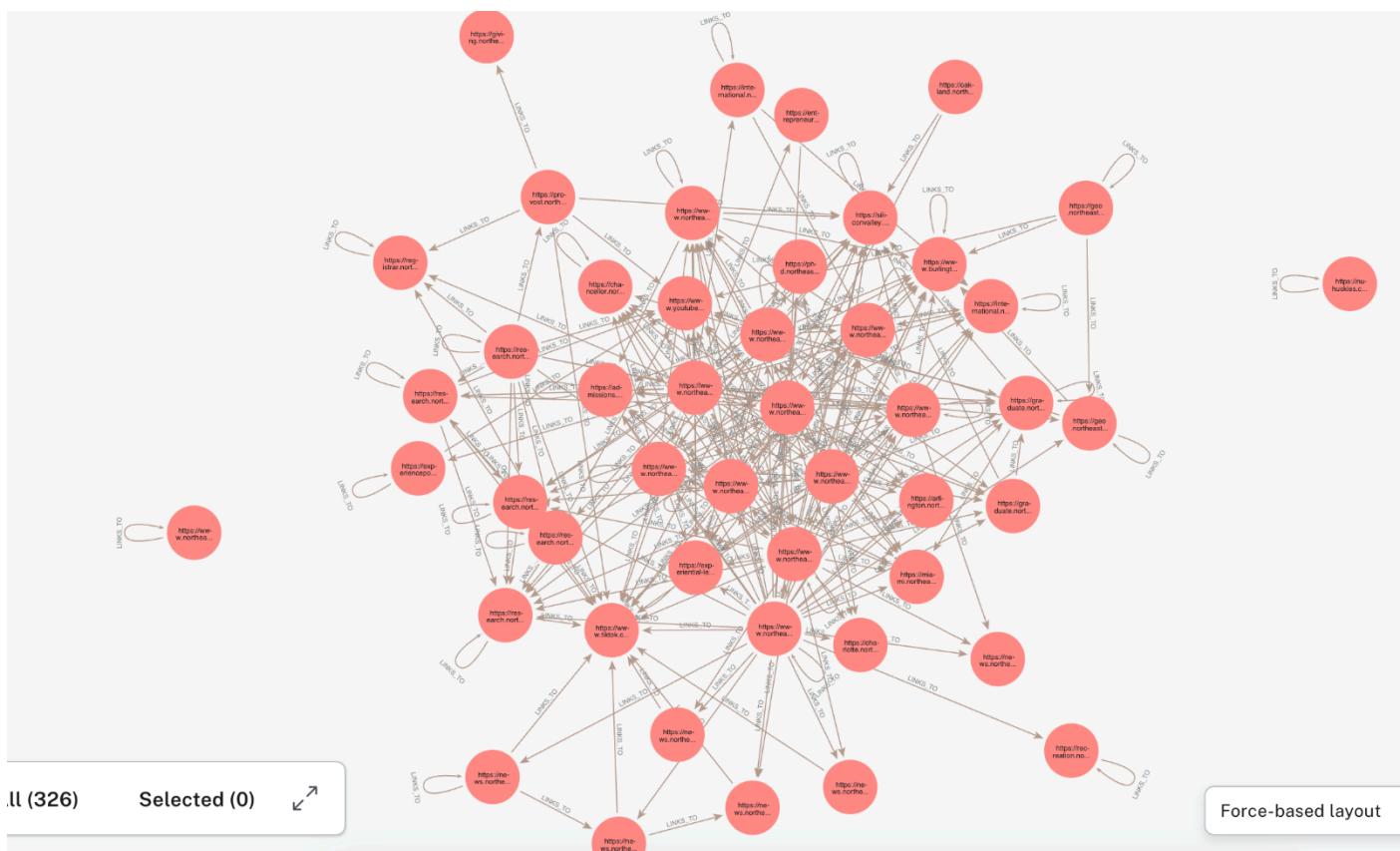
Variations in Graph based on Thread and Depth size

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

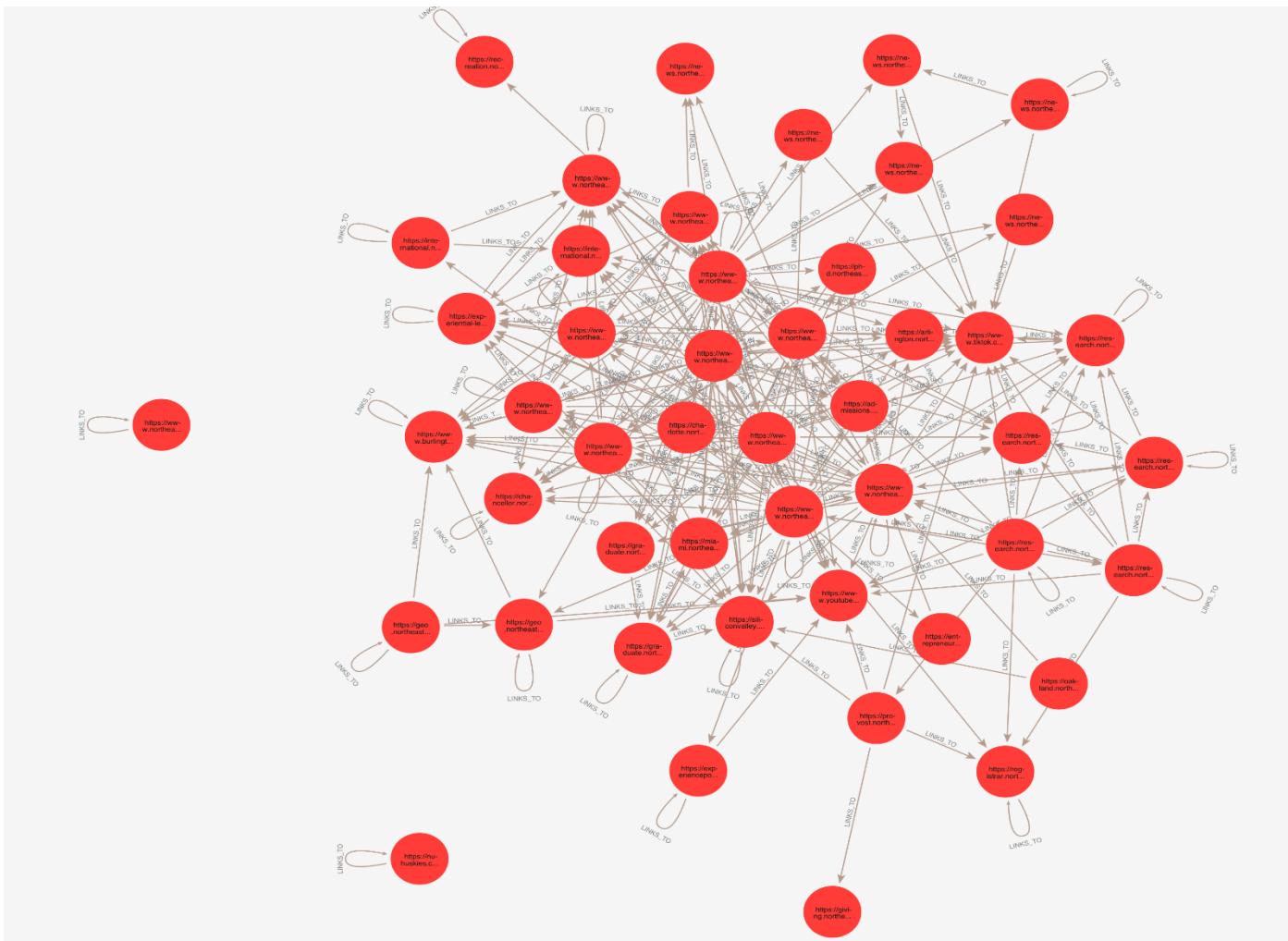
T-2 D-2



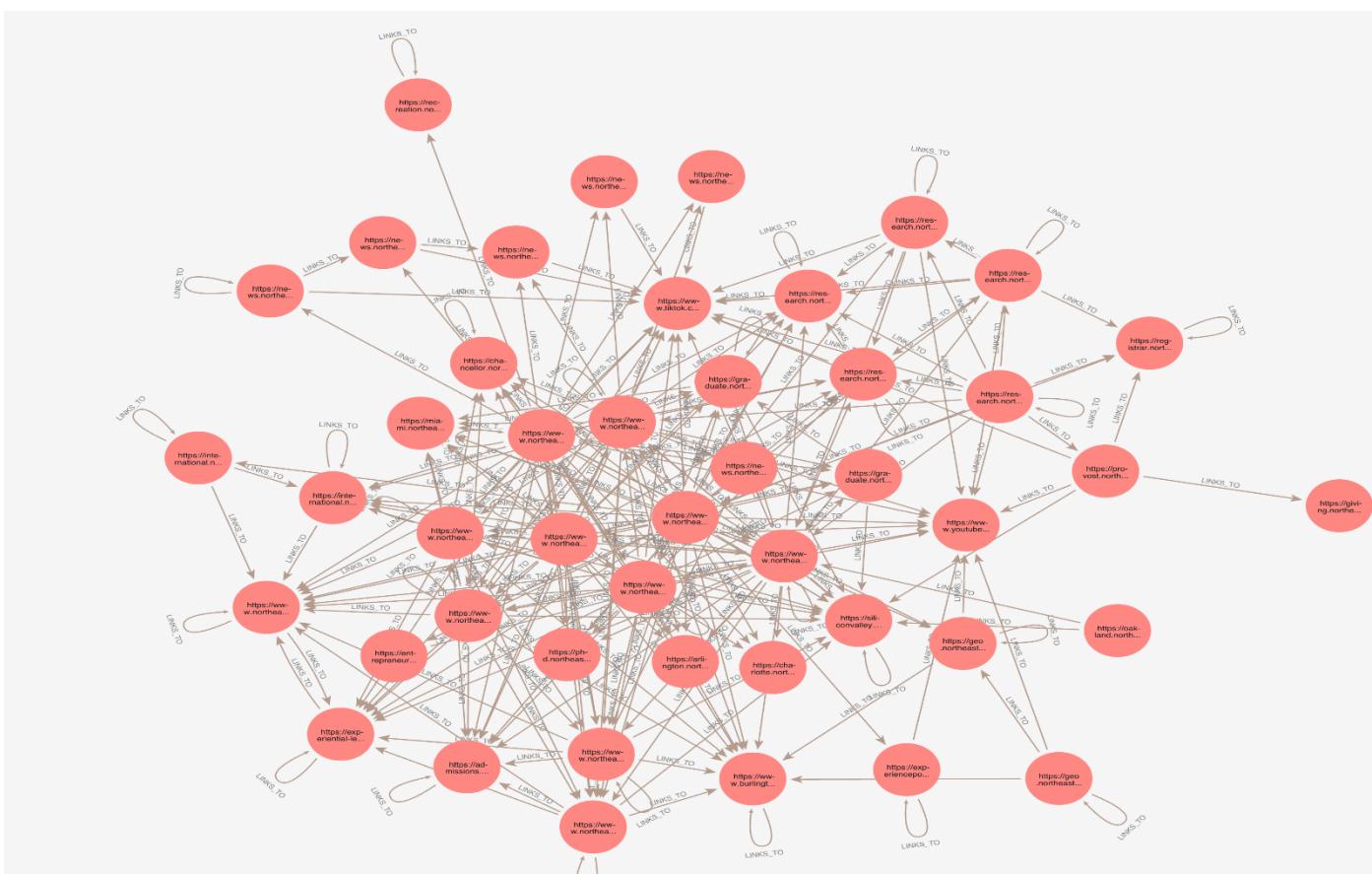
T-2 D-3



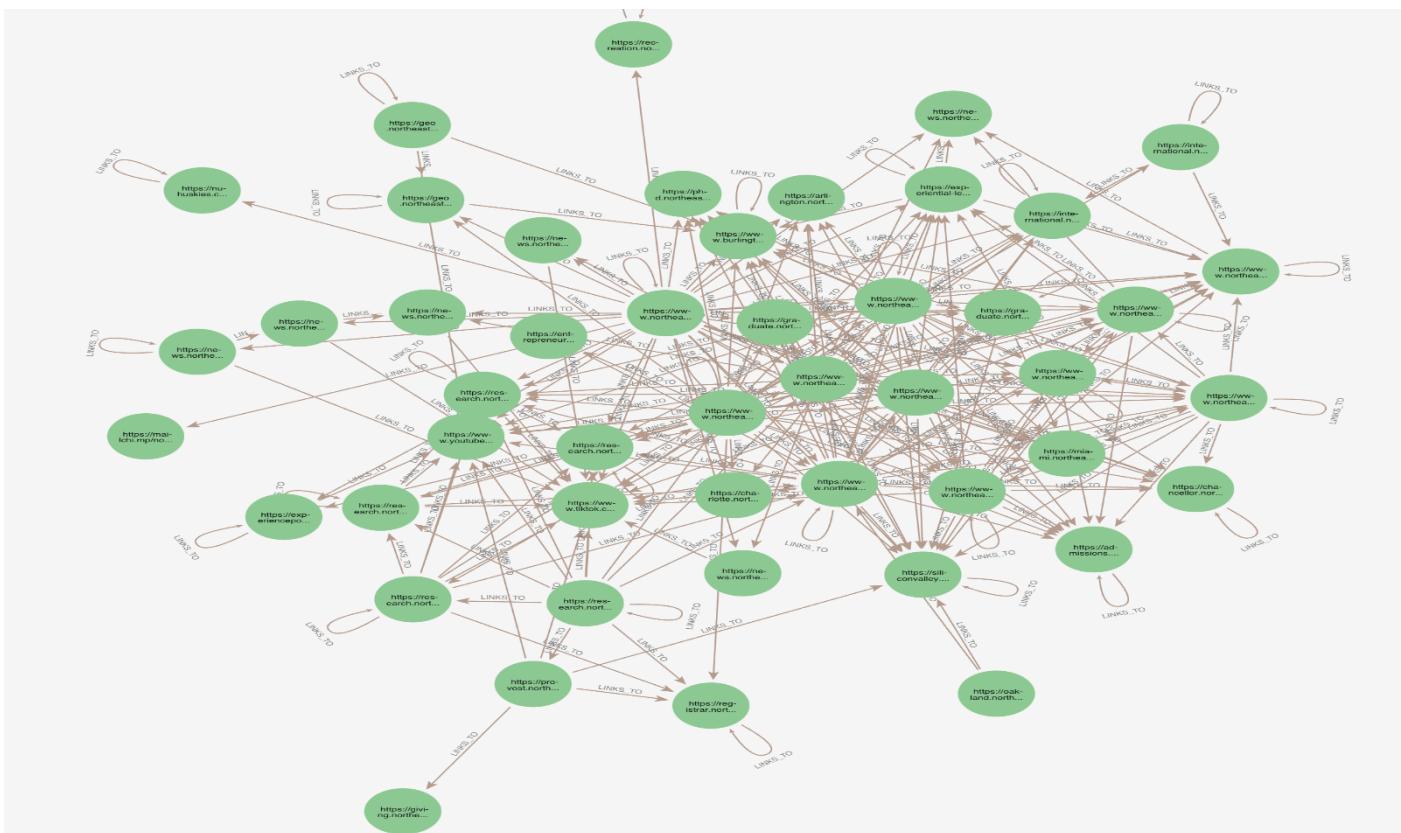
T-2 D-4



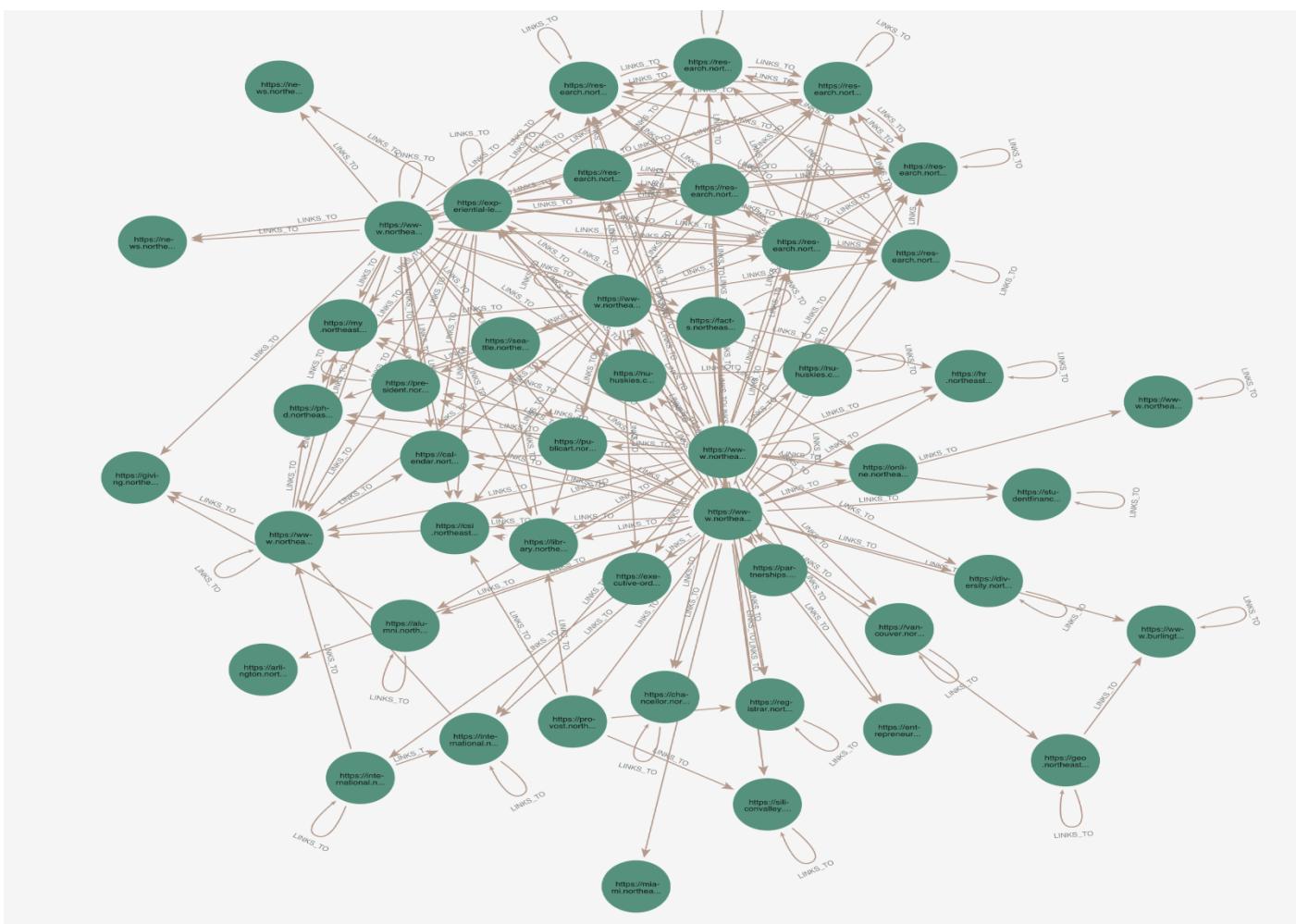
T-2 D-5



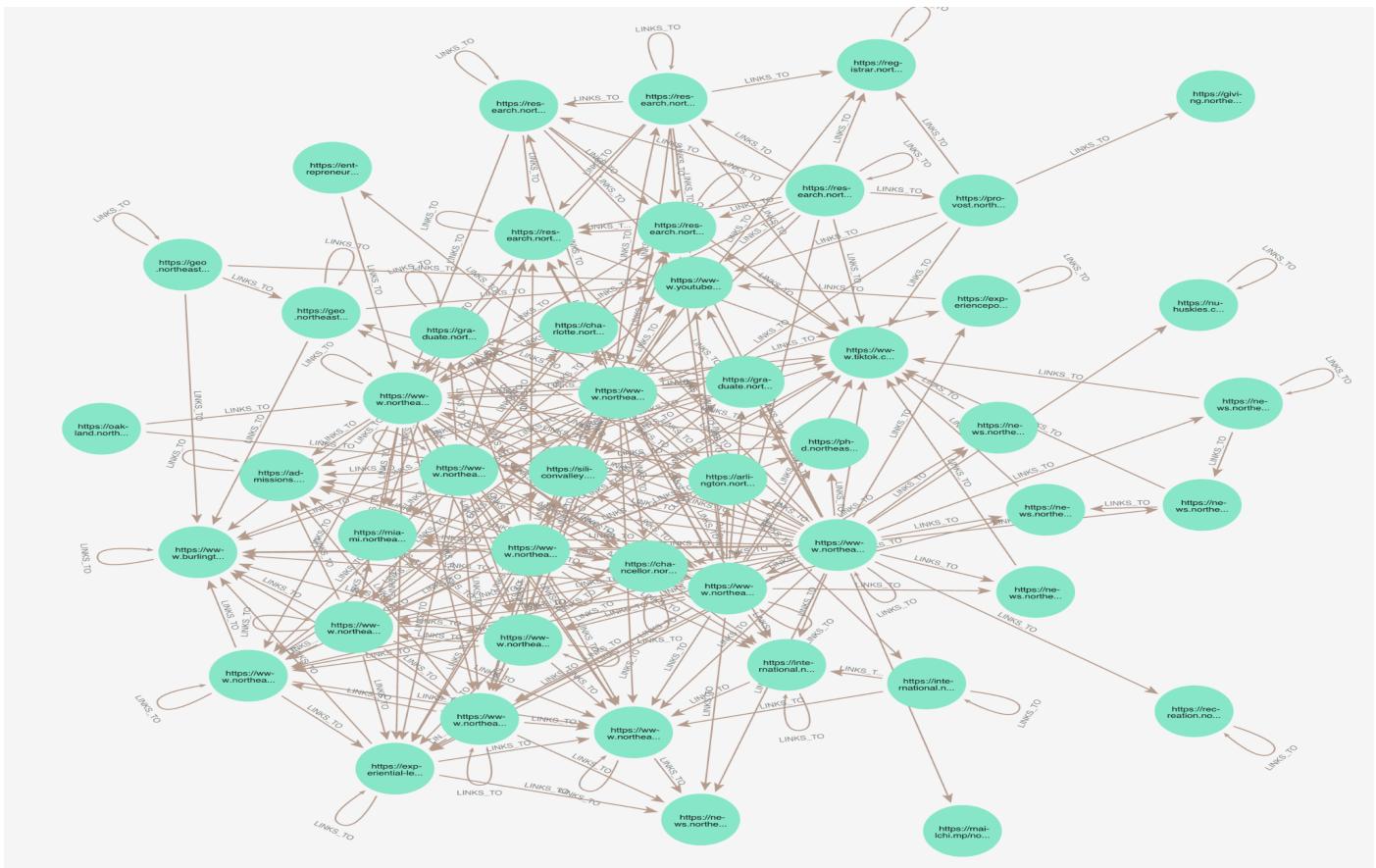
T-4 D-2



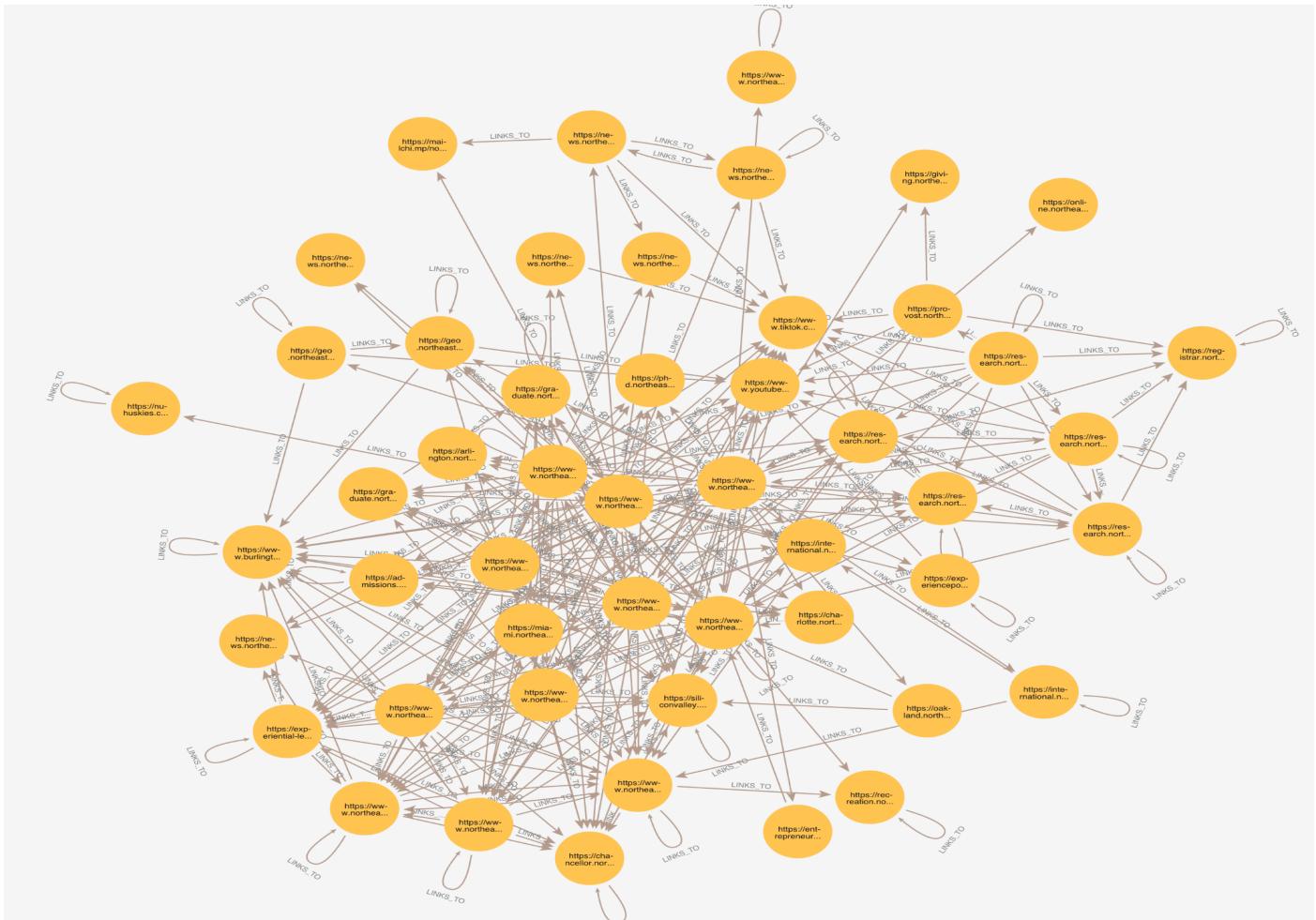
T-4 D-3



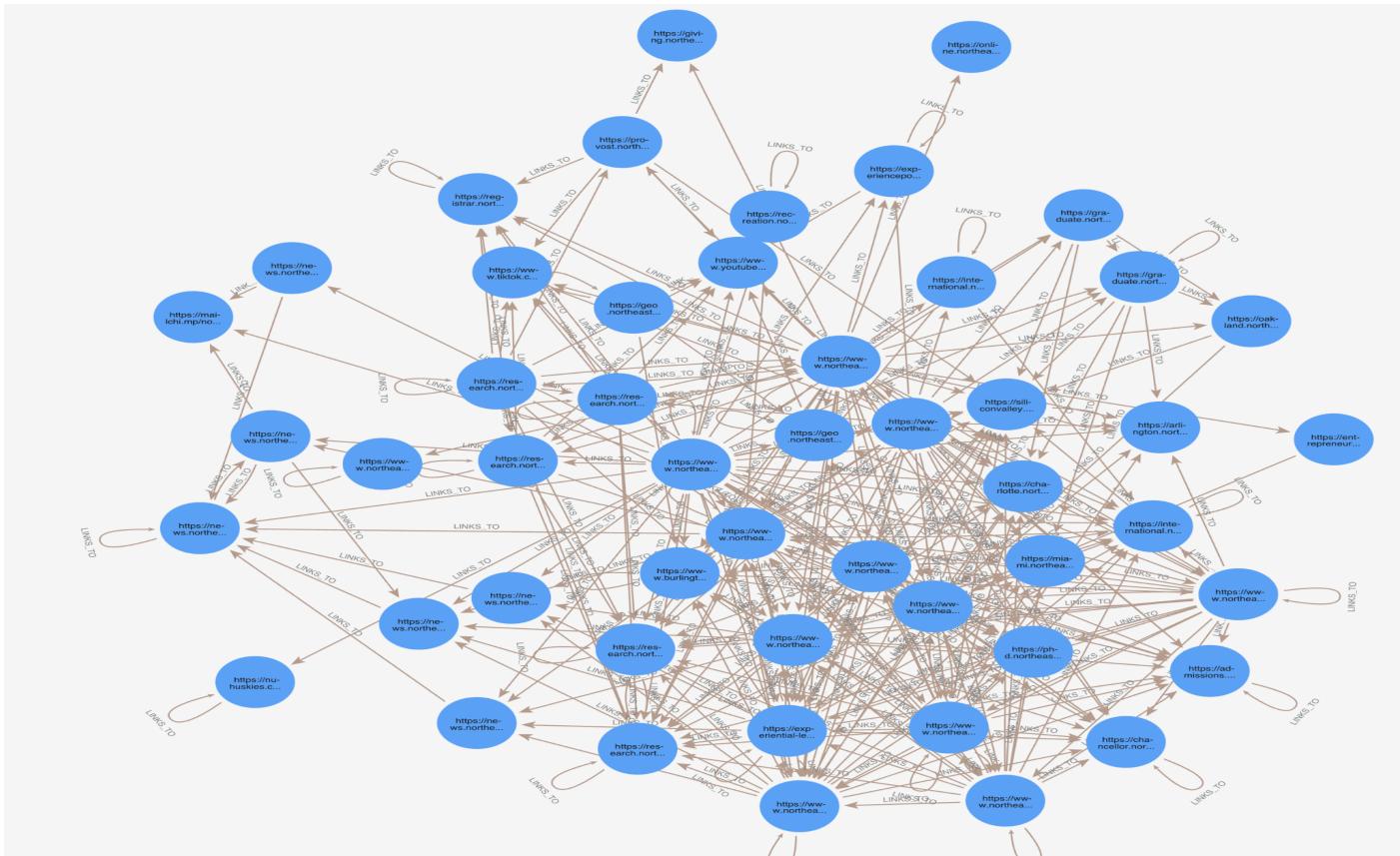
T4-D5



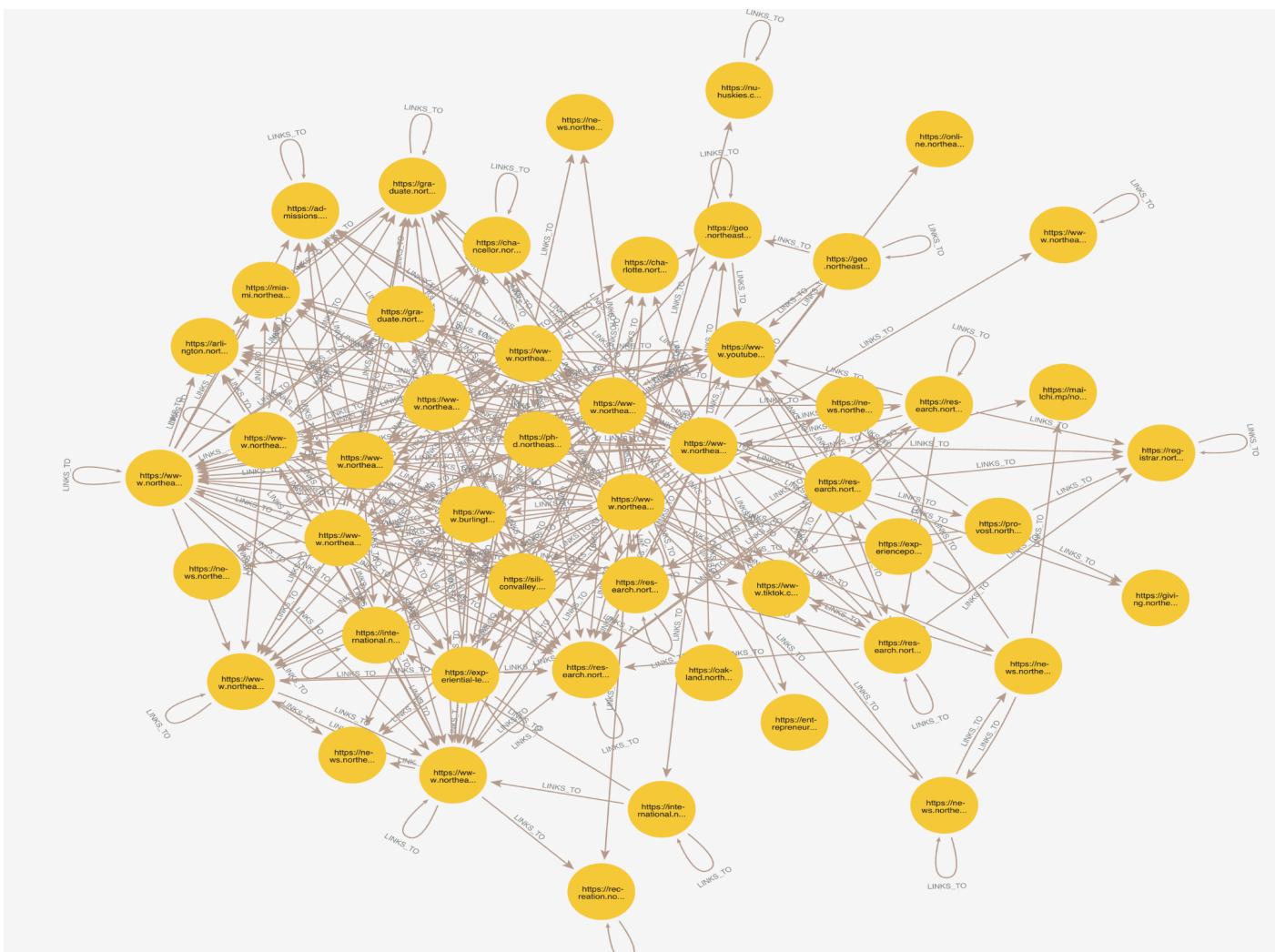
T8-D5



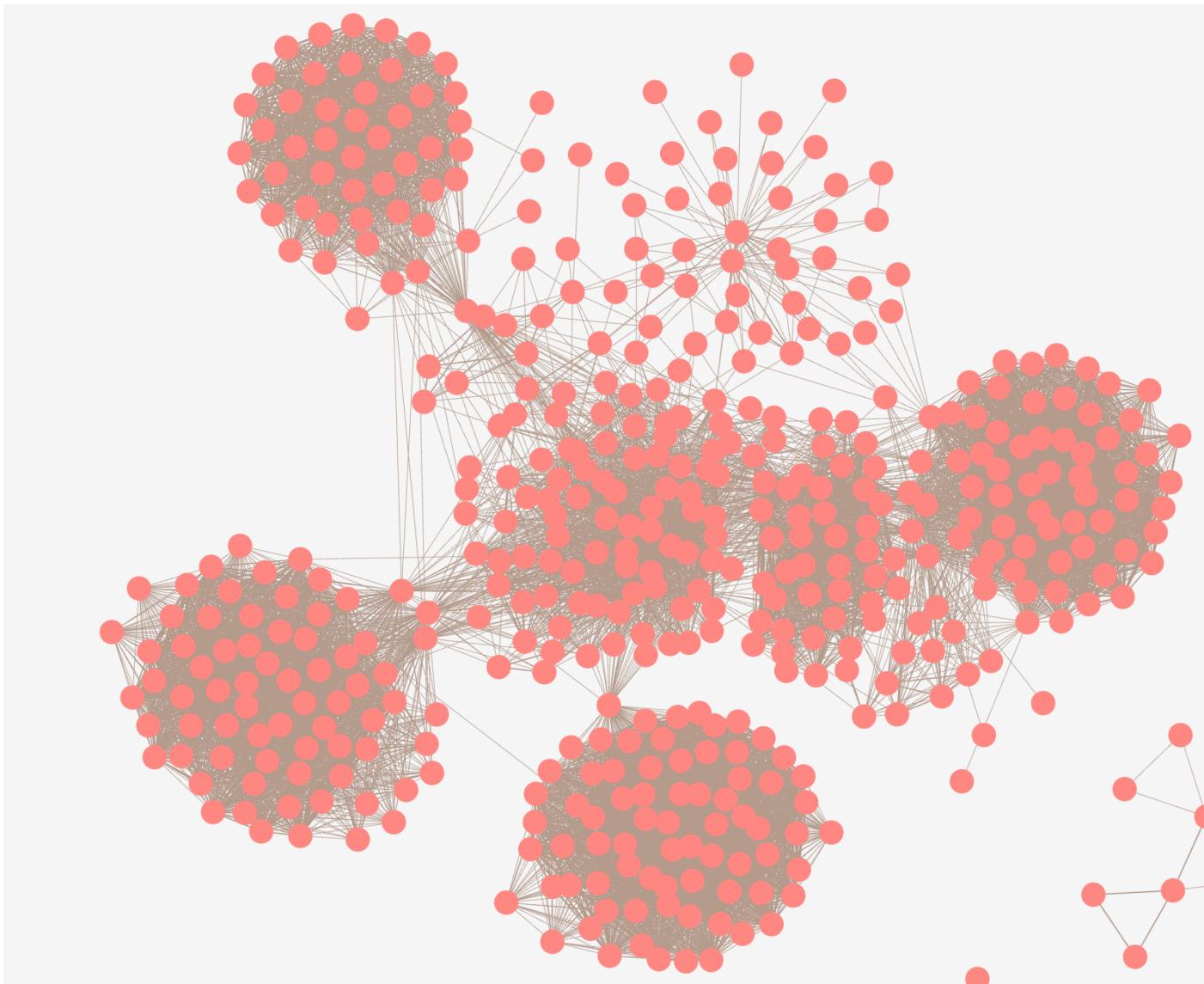
T16-D5



T32-D5

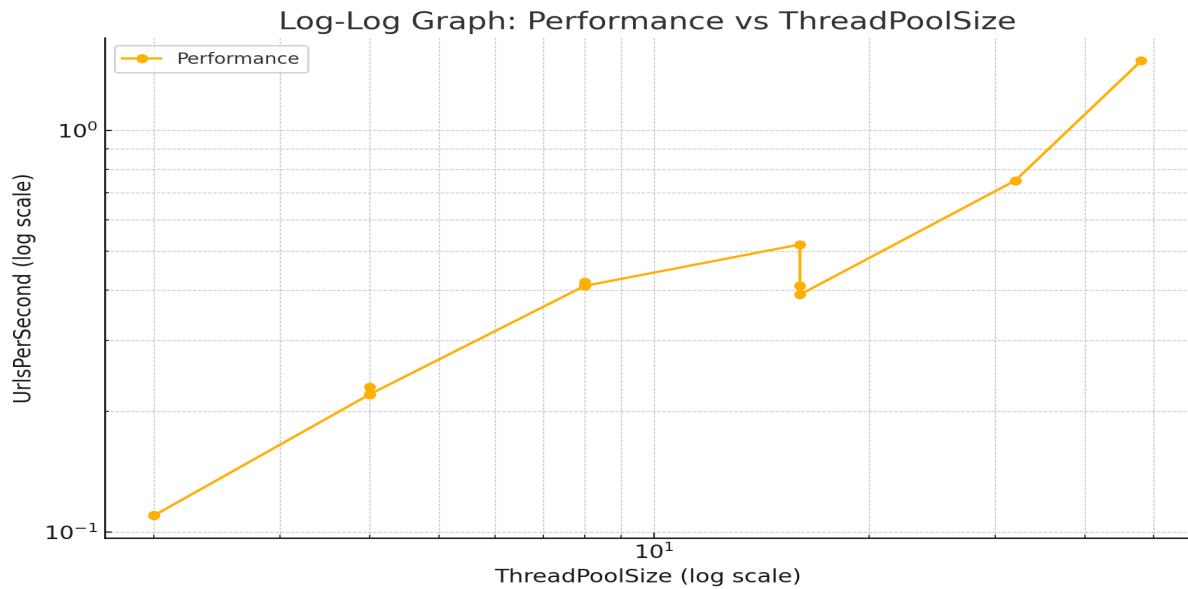


T48-D5 (this is our largest graph with ~500 URLs visited (cut off manually))



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:

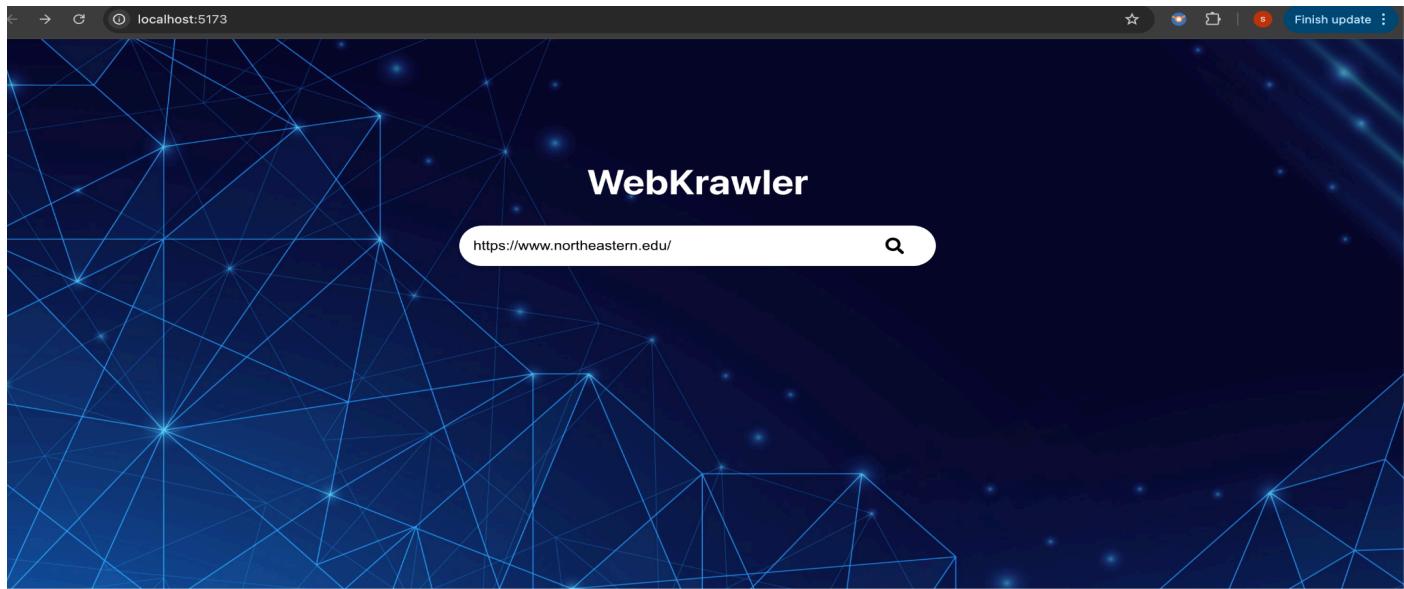
- Spring Boot Starter Web: Provides a starter for building web applications, including RESTful APIs, using Spring Boot.
- Neo4j OGM Core: Object Graph Mapping (OGM) library for interacting with Neo4j databases.
- Neo4j Java Driver: Official Java driver for connecting to Neo4j databases.
- Jsoup: Library 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.



A screenshot of a web browser window titled 'localhost:5173/results'. The background is the same network graph. A central modal window is open, showing the 'Crawled URLs' tab selected. It displays a list of URLs with their ranks and in-degrees:

URL	Rank	In-Degree
https://recreation.northeastern.edu/	0.1182050831675692	68
https://www.nulondon.ac.uk/	0.09103020536792787	112
https://registrar.northeastern.edu/	0.04365506862823522	52
https://roux.northeastern.edu/	0.03686618915482293	52
https://chancellor.northeastern.edu/	0.031303563848941486	41

A screenshot of a web browser window titled 'localhost:5173/results'. The background is the same network graph. A central modal window is open, showing the 'Black-listed URLs' tab selected. It displays a list of URLs:

https://www.creativebookmark.com/
https://www.ifupdate.org
https://www.ad-tracker.example
https://www.popup-ads-site.net
https://www.bannerads.org
https://nu.outsystemsenterprise.com/FSD/
https://www.instagram.com/northeastern
https://www.northeastern.edu/charlotte/