Neo4j's BFS and DFS Evaluation in GDS and APOC Libraries with SPL Feature Models

Hazim Shatnawi
The George Washington University
Washington, D.C., USA
hazim.shatnawi@gwu.edu

Jamil Saquer
Missouri State University
Springfield, MO, USA
jamilsaquer@missouristate.edu

***Abstract*— This paper evaluates the performance of Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms in Neo4j using the Graph Data Science (GDS) and the Awesome Procedures on Cypher (APOC) libraries. We benchmark these algorithms on feature models of varying complexity to assess their efficiency and scalability. Our findings indicate that GDS significantly outperforms APOC in terms of traversal times, particularly for large and complex graphs. This study underscores the importance of algorithm optimization in graph databases and provides insights into practical applications and future directions for improving feature model management in software product lines.**

***Index Terms*—Neo4j; graph databases; traversal algorithms in graph data science; directed acyclic graphs; software product lines;**

I) INTRODUCTION

Software Product Lines (SPLs) enable software reuse by sharing features across related systems to meet market or mission needs. Feature models (FMs) capture these features and their dependencies, managing commonality and variability [1]. As SPLs grow, their complexity increases, requiring efficient management and analysis techniques.

Neo4j, with its native graph storage and querying capabilities, is ideal for encoding and analyzing FMs [2]. Neo4j's advanced graph algorithms in the APOC [3] and GDS [4] libraries enhance performance and scalability for SPL management.

This paper evaluates Neo4j's GDS and APOC libraries for traversing directed acyclic graphs (DAGs), used to represent FMs in SPLs. We compare the performance of Neo4j's GDS and APOC libraries in traversing DAGs of varying complexity. DFS and BFS are two famous algorithms often used to traverse graphs, which we concentrate on in this article. We utilize FMs that represent SPLs as a prime example of graph representation in Neo4j. These models serve as an excellent illustration due to their ability to scale from small to very large sizes while encapsulating complex relationships. This characteristic allows us to effectively demonstrate the capabilities of Neo4j in handling both the simplicity and intricacy inherent in graphs that represent FMs and their interconnections.

We represent FMs as DAGs due to their natural fit with hierarchical and complex dependency structures, ensuring clear parent-child relationships without circular dependencies [2]. Following Feature-Oriented Domain Analysis (FODA), using DAGs simplifies modeling, focusing on hierarchical and dependency relationships [5]. Representing FMs as DAGs provides a structured, efficient approach to managing feature dependencies such as requires and excludes dependencies, with cycles managed through detection and traversal algorithms, ensuring integrity and clarity in performance evaluation.

DOI reference number: 10.18293/SEKE2024-110

II) NEO4J

We initially used a relational database for FMs but faced scalability issues [6]. Transitioning to Neo4j improves performance and scalability for SPLs. Neo4j's nodes, properties, labels, and relationships efficiently manage data. This encoding technique optimizes data management, improving performance, scalability, and complexity [7]. This shift to graph-based structures optimizes performance and management, advancing SPL research and applications.

Neo4j's architecture handles connected data efficiently, advancing SPL research and applications. Its native graph storage outperforms relational databases on interconnected data with many relationships, facilitating automated feature model creation and product generation [2, 7].

Neo4j includes APOC and GDS libraries for complex data operations. APOC offers procedures for data transformations and pathfinding, enhancing graph operations. Integrating APOC and GDS extends data manipulation abilities, vital for academic and enterprise use. Our experiments use these libraries to assess their effectiveness in analyzing SPL feature models in Neo4j.

III) EXPERIMENTAL SETUP AND METHODOLOGY

This section details our methodology for analyzing FMs in Neo4j using the APOC and GDS libraries. We assess how model complexity affects BFS and DFS traversal performance. These algorithms explore the model's structure, evaluating effectiveness in both APOC and GDS contexts.

Cypher queries are executed via the Neo4j Python driver. Large datasets are ingested using Cypher commands, with the Batch Importer Technique and CSV files supporting incremental loading to minimize memory use. We use a Cypher algorithm discussed in [2] to ensure that DAGs remain cycle-free.

Our experimental procedure involves nested loops over database encodings, FMs, and performance tests, calculating

averages and statistics for each test. The parameters are:

- B = the collection of database encodings
- FM = the collection of feature models
- PT = the collection of time-based feature model performance tests
- N = the number of repetitions of the test runs

The following pseudocode, expressed in terms of the above parameters, outlines the experimental procedure:

```
PROCEDURE (DB , FM , PT , N) :
  For each db in DB :
    For each fm in FM :
      Repeat N times :
        For each pt in PT :
          Perform pt , measuring time to complete
        Compute and record statistics for this run which
        includes mean , minimum , maximum for
        each  pt
```

Figure 1. Experimental Procedure

Table 1 lists the number of features, relationships, requires, excludes, and graph depths for the nine FMs in our experiments.

Table 1. Graphs' Parameters for Experiments

FM	#Feature	#Relations	Depth	#Require	#Exclude
1	19	20	4	1	1
2	500	485	12	72	91
3	1000	980	15	169	161
4	2000	2008	16	331	333
5	5000	5200	16	246	504
6	10000	10000	21	683	726
7	18000	13671	20	1140	1604
8	30000	29999	24	1524	1542
9	42000	37212	26	1484	1234

IV) EXPERIMENTAL RESULTS AND DISCUSSION

We compare APOC and GDS performance across graph sizes, from small to complex. Tables 2-5 show DFS and BFS results for different FM sizes, where time is in seconds.

Table 2. Experimental Results of using DFS from GDS

Nodes	Min	Max	Mean
20	0.000878	0.027010	0.004065
500	0.105626	0.141981	0.123033
1000	0.226812	0.311963	0.266546
2000	0.222220	0.318001	0.266530
5000	0.743833	0.935618	0.846392
10000	3.540471	3.853090	3.675441
18000	5.699261	6.437443	5.964832
30000	7.589752	11.903438	11.496000
42000	10.783620	15.021226	12.012990

Table 3. Experimental Results of Using BFS from GDS

Nodes	Min	Max	Mean
20	0.000948	0.029171	0.004390
500	0.114076	0.153340	0.132875
1000	0.244957	0.336919	0.287869
2000	0.239598	0.343441	0.287852
5000	0.803340	1.010467	0.914103
10000	3.823709	4.161338	3.969476
18000	6.155201	6.952438	6.442019
30000	8.196932	12.855713	9.040000
42000	11.245310	16.222966	11.880000

Table 4. Experimental Results of using DFS from APOC

#FM	Min	Max	Mean
20	0.000998	0.030011	0.004517
500	0.117362	0.157757	0.136703
1000	0.252013	0.346625	0.296162
2000	0.261435	0.357648	0.308444
5000	0.875098	1.05955	0.955658
10000	3.930412	4.281211	4.079411
18000	6.332512	7.152714	6.627591
30000	10.126215	16.356986	14.513201
42000	19.745435	25.907372	21.290300

Table 5. Experimental Results of using BFS from APOC

#FM	Min	Max	Mean
20	0.000982	0.023704s	0.003481
500	0.119620	0.184382	0.142963
1000	0.281596	0.370896	0.318995
2000	0.256398	0.399774	0.346087
5000	0.984398	1.646392	1.159850
10000	4.051471	6.238928	4.721993
18000	6.351740	10.146643	7.850616
30000	15.568654	17.351475	16.508593
42000	19.920761	31.602018	23.123314

Fig. 2 shows a plot of the running time of DFS in GDS versus APOC for different FM sizes. Likewise, Fig. 3 shows a plot of the running time of BFS in GDS versus APOC for different FM sizes.

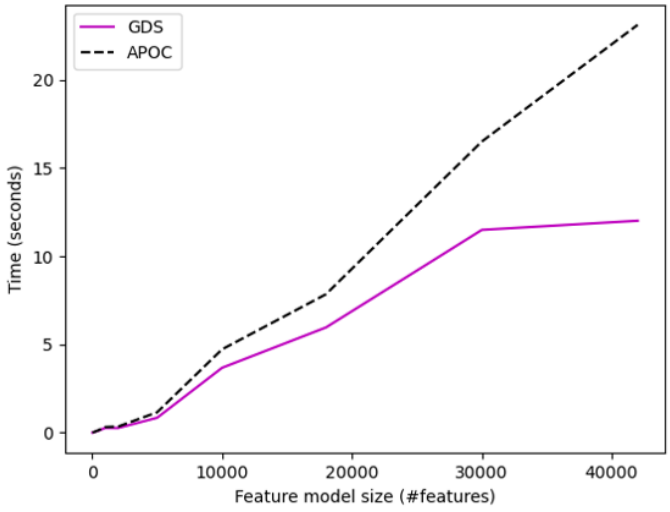

Figure 2. GDS versus APOC in Utilizing DFS in our Experiment.

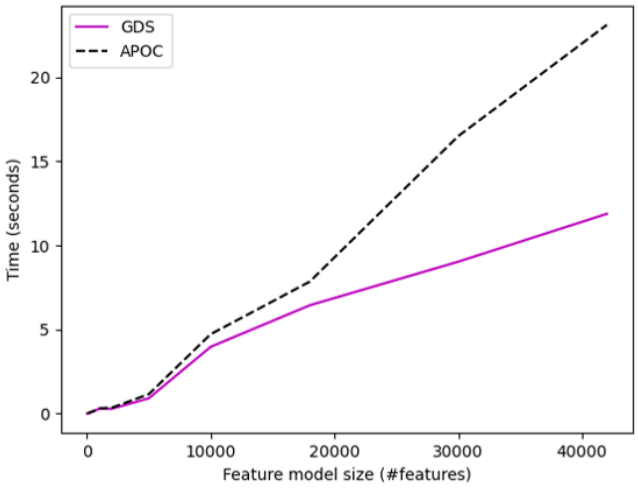

Figure 3. GDS versus APOC in utilizing BFS in our Experiment.

As the results demonstrate, in the case of small graphs, the traversal times are both minimal and nearly similar, making it challenging to draw a definitive conclusion. The following points highlight the key aspects of the comparison:

- **Consistency and Scalability.** Both APOC and GDS traversal times increase with the number of nodes, but GDS consistently outperforms APOC when FMs grow in size and complexity. For instance, APOC DFS time for 42,000 nodes is 21.3 seconds compared to 12.0 seconds for GDS. Similarly, APOC BFS times are 23.1 seconds versus 11.9 seconds for GDS. This demonstrates that while both systems scale predictably, GDS is more efficient. For example, it takes around half the time of APOC when traversing the feature model with 42000 nodes and 37212 relationships.
- **Impact of Depth and Constraints.** Traversal times increase with greater depth and more constraints. A model with a depth of 26 shows slower traversal times. A simpler model with a depth of 4 and minimal constraints, exhibits faster traversal times. APOC demonstrates consistent performance with increasing node counts but generally slower traversal times than GDS in large and complex FMs.

In summary, while both APOC and GDS handle depth and constraints effectively, GDS's consistent performance and faster traversal times make it the more efficient choice for large-scale and complex models.

V) Conclusion

The analysis of DFS and BFS traversal times using GDS and APOC in Neo4j reveals several key insights. 1) GDS Superiority: GDS consistently outperforms APOC in traversal times, particularly in large and complex FMs. This is attributed to GDS's optimized algorithms, making it the preferred choice for handling extensive and intricate graph structures. 2) Scalability and Efficiency: Both GDS and APOC demonstrate scalability, but GDS offers more efficient management of depth and constraints, resulting in faster traversal times. This efficiency is crucial for practical applications where performance and speed are paramount.

References

[1] K. Czarnecki and U. W. Eisenecker, Generative programming: methods, tools, and applications, Boston, MA, USA: Addison-Wesley, 2000.

[2] H. Shatnawi and J. Saquer, "Encoding feature models in Neo4j graph databas," in *2024 ACM Southeast Conference (ACMSE 2024)*, Marietta, GA, USA, 2024.

[3] Neo4j, Inc., "APOC user guide," 2024. [Online]. Available: https://neo4j.com/docs/apoc/current/.

[4] Neo4j, Inc., "Graph data science user guide," 2024. [Online]. Available: https://neo4j.com/docs/graph-data-science/current/.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, 1990.

[6] H. Shatnawi and C. Cunningham, "Mapping SPL Feature Models to a Relational Database," in *Proceedings of ACM SouthEast Conference (ACMSE 2017)*, Kennesaw, GA, USA, 2017.

[7] J. A. M. Stothers and A. Nguyen, "Can Neo4j replace PostgreSQL in healthcare?," *AMIA Joint Summits on Translational Science*, vol. 2020, p. 646–653, 2020.