Summary on
Graph Patterns Indexes: their Storage and Retrieval

Jaroslav Pokorný, Michal Valenta, 2018. Association for Computing Machinery,
doi.org/10.1145/3282373.3282374

This article reviews existing graph pattern indexes, discussed the possibilities of structure-based graph indexing in GDBs and subsequent design and implementation of a graph index for the selected Graph database management system (GDBMS). The index is organized in a hash table and stored in the different database other than the graph database, enabling create/use/update operations. Analyses are done on Neo4j database engine, with the indexes stored on MapDB for comparison between queries with/without indexes.

Background:

The article firstly presents the background information, explaining what is graph database and GDBMS, how is the graph pattern match query working, and the limit of existing indexing method.

1) Graph database (GDB) is constructed on graph theory, with nodes, properties, edges, emphasizing on entities and relationships between entities.
2) GDBMS shows great effectiveness in traversing the graph from several root nodes, however suffer a lack of consistence.
3) Pattern match query is a fundamental processing requirement, that searches over the graph G to check existence of a pattern graph in G.
4) Indexing methods have been designed for pattern matching query, but most subgraph isomorphism algorithms are based on a backtracking method. Current indexes implemented in GDBMS are mostly value indexes, while Structure-based indexes can be more helpful.

Review:

The article then summarize related works on graph pattern and indexing in details, to better explain how different types of indexes used in GDBMS, what is the limitation, what is the focusing problems.

1) Value-based indexing: composite index is efficient but limited for definition on property keys combination, mixed index can be used for any lookups, node-centric index is local index structure built individually per node.
2) Value index in GDBMS: Neo4j use Cypher to create indexes on one or more properties for all nodes. Sparksee5 uses B+-trees and compressed bitmap indexes to store nodes and edges. Titan6 supports graph indexes and node-centric indexes.
3) structure-based index: the design of structure-based index is to extract and index structure properties, using filter to search for candidates for the query for improved efficiency.
4) Research on structure-based index: research focusing on improving substructure features, to enable better filtering power that reduces number of candidate subgraphs for saving time. Types included path-based index, subgraph-based index, and spectral methods.

Solution:

The article explains their solution on how they deal with the indexing and storage for graph patterns.

1) The graph pattern index is stored in external database, MapDB here is used.
2) Both node and edge-Ids is used in order to support multigraph structure indexing

Analysis:

Finally, the article focuses on implementing and testing the solution, to show the effectiveness of their index implementation, with the comparison with another implementation of pattern indexes, to draw a conclusion of their improvement.

1) Tests are done within the environment of GraphAware framework9 on Neo4j.
2) Query performance:
    a) a graph G as a 100,000-node community network with 500,000 edges is used for testing
    b) the query time with indexes is more than 100 times faster than without
    c) the index creation uses 78s/3min/4h for High-level/low-level/none cache
    d) in index created take 77KB
3) DML operations:
    a) Update on index is done in the same transection of a DML statement
    b) Tests done on creating index, creating a relationship, deleting a relationship, and deleting a node, to measure the response time
    c) Comparison between no-index and with-index shows improved response time over 100 times, indicating that the maintenance on such small-size index is much cheaper than the speedup query.
    d) Comparison among this-index and another structure-based indexing published before shows reduced response time.
    e) The results indicate that their approach is promising and suitable for further research especially with conjunction of optimizer indexes usage improvement.