MARIO MILER, M.Eng.
E-mail: mmiler@geof.hr
DAMIR MEDAK, Ph.D.
E-mail: dmedak@geof.hr
DRAŽEN ODOBAŠIĆ, M.Eng.
E-mail: dodobas@geof.hr
Faculty of Geodesy, University of Zagreb
Kačićeva 26, 10000 Zagreb, Croatia

Information and Communication Technology
Preliminary Communication
Accepted: Mar. 27, 2013
Approved: Oct. 12, 2013

THE SHORTEST PATH ALGORITHM PERFORMANCE COMPARISON IN GRAPH AND RELATIONAL DATABASE ON A TRANSPORTATION NETWORK

ABSTRACT

In the field of geoinformation and transportation science, the shortest path is calculated on graph data mostly found in road and transportation networks. This data is often stored in various database systems. Many applications dealing with transportation network require calculation of the shortest path. The objective of this research is to compare the performance of Dijkstra shortest path calculation in PostgreSQL (with pgRouting) and Neo4j graph database for the purpose of determining if there is any difference regarding the speed of the calculation. Benchmarking was done on commodity hardware using OpenStreetMap road network. The first assumption is that Neo4j graph database would be well suited for the shortest path calculation on transportation networks but this does not come without some cost. Memory proved to be an issue in Neo4j setup when dealing with larger transportation networks.

KEY WORDS

pgRouting, OpenStreetMap, Dijkstra, benchmark, Neo4j, PostgreSQL

1. INTRODUCTION

The simplest question in routing that needs an answer is: "What is the most effective way to get from here to there?". To answer this question a simple and complex shortest path algorithms are used. Today, the shortest path analysis is used in everyday life, for example in car or personal navigation systems. The problem of the shortest path calculation has been the topic of research for several decades [1–4] but the influence of database system implementation has been rarely considered. With recent rapid development of database technology, there has always been a question: "Will a graph database perform better in the shortest path calculation than a relation database?".

Relational databases are often used for storing transportation network data. Many of them provide navigation models which permit managing networks and perform network operations [5]. Many researchers have been working with such systems and there are many examples and applications of database-based routing systems in the traffic and transportation field. Christopher et al. [6] designed a collaborative route planning system for utility vehicles which relies heavily on route calculation based on underlying transportation network stored in relational databases. Innerebner et al. [7] created a web-based system named ISOGA that uses isochrones to perform geographical reachability analysis over a transportation network stored in a relational database. Zlatanova et al. [5] focused on examining the problem of finding the optimal path for moving objects to avoid moving obstacles using a relational database.

Most of the databases used today are based on the relational model [8] which is a theoretical basis for relational database management systems (RDBMSs). RDBMSs today are dominant for storing various kinds of structured data. These systems were primarily designed for storing data used by business applications. Over the years, the need for storing other types of data and large amounts like spatial, hierarchical and graph data has emerged but technology remained the same. Relational databases proved to be very efficient in storing and querying large amounts of data but they were not adequate for analyzing relations among entities. One of these examples is the shortest path routing algorithm. This algorithm requires a large amount of joins statements which are computationally very ex-

pensive and cannot be used in a standard SQL query [9].

Most of the time we are forced to adapt a data model to fit into the RDBMS structure, and we do not change the technology that would be more suited for this type of data, e.g. network data. Stonebraker et al. [10] changed the way of thinking about data and found that there is *no one-size-fits-all* database solution. Instead, each database problem can be solved with another database solution. This led to a large number of alternative storage systems that are used today. Over the last few years there has been a rapid expansion of new database technologies popularly called NoSQL (Not Only SQL). The primary focus of these technologies is on efficient handling of large amounts of semi-structured data used in the web space. Although such databases and similar systems already exist (e.g. object-oriented databases or XML stores) only in the recent years have they become adopted on the market and in the community. Although NoSQL is a broad term for almost any database that is different in some way from the standard RDBMS, there are several classifications of NoSQL databases. Most of them can be classified as: key-value, document, wide-column, object and graph stores [11–14].

1.1 Graph Database System

The database that has been growing in popularity is a graph database or also called Graph Database System (GDB). The most significant reason for this growth is the importance of graph data structure in the fields of social, information and biological sciences as well as in the development of computer hardware. The graph data structure fits naturally for modelling of world objects, entities and relationships between them [15].

According to Angles et al. [16, p1] the graph database models are defined as "*those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors*". In other words, a graph database is a database management system that is based on graph theory introduced in 1736 [17]. The graph theory uses nodes for storing entities and edges for relationships among them. Graph databases emphasize the relations among entities rather than entities themselves [18]. Any model can be thought of as a representation of reality. A graph model can be thought of as a collection of objects such as people or places and the relationship between them such as "friend" or "living in". Those objects and relationships form a network or a graph [19]. A graph structure is defined as $G = (V, E)$ where $V = \vartheta_1, \vartheta_2, \vartheta_3, \ldots, \vartheta_n$ is a set of vertices and E is a set of edges $E = \varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots, \varepsilon_n$. An edge $\varepsilon_i \in E$ is defined with triple (i, j, ω_i) where $i, j \in V$ and ω_i is a positive real number. A directed edge is defined as $i \rightarrow j$. The graph database systems today are still a nascent technology and in major development. Most of them support directed attributed multi-graph also known as property graph. This allows attaching attributes to every node and relationship on top of graph structure [19]. The property graph is an important prerequisite for calculating the weighted shortest path, e.g. using Dijkstra algorithm.

The most popular shortest path implementation on a relational database is pgRouting which is implemented on top of the PostgreSQL database. Graph databases already have the shortest path implementations in their core. Neo4j is one of the graph database systems used for semi-structured and network-oriented data and has been in production since 2003. It was developed in Java programming language [20] and free for non-commercial use. Neo4j can be used as an embedded or server database. It has started as an embedded database but recently its main usage has been as a server database version and in the future it will continue to develop in that direction.

1.2 Database benchmarks

A database benchmark is a standard set of operations and instructions sent to two or more database systems to evaluate relative and quantitative performance in a controlled experiment [21]. The most common standard RDBMS benchmarks are TCP-(C, H, E) benchmarks [22] but there are also other database benchmarks like Bristlecone or Open Source Development Lab Data Base Test Suite (OSDL-DBTS) [23]. These benchmarks attempt to simulate real-world scenarios mostly in the business domain applications and none of these standard database benchmarks can be used for this research. The business domain data structure is significantly different than the one of the transportation networks. Transportation networks have graph-like data structure.

The spatial database is a database that has been developed and optimized for storing, querying and manipulating 2D and 3D spatial objects (points, lines, polygons, etc.) in geometric space. They are mostly used by geographic information systems (GIS) but can be used in other domain applications where space plays an important factor. Spatial database benchmarks are by construction similar to standard database benchmarks but their main focus is on spatial query and spatial join performance. There are only a few spatial database benchmark projects, especially ones detailed as TCP benchmarks used in standard RDBMS. One of the most recent and versatile benchmarks is *Jackpine* [23]. *Jackpine* is a vector-based spatial database benchmark based on *Bristlcone*

benchmark. It is a flexible benchmark that uses Java Database Connectivity (JDBC) driver implementation and can support almost any database. The author of this benchmark used two approaches, micro and macro. The micro benchmark is used for testing topological relationships, spatial analysis functions and data loading queries. This part of the Jackpine is used to assess database processing and index performance.

To our knowledge there are only a handful of graph database benchmarks and guidelines [9, 18, 24]. Dominguez-Sal et al. [18] propose guidelines for a graph oriented benchmark differing from any standard or spatial database benchmark. According to these authors the proposed tests are: traversal (which includes the shortest path calculation relevant for this research), graph analysis, connected components, communities, centrality measures, pattern matching, graph anonymisation and some application domain generic operation. Queries that they propose must represent the workload of the real environment in application domain. They also noted that the shortest path graph analysis and real time analysis of traffic networks are one of the application domains where graph databases could prove some benefit.

This paper provides a benchmark designed to measure the performance of a Dijkstra shortest path algorithm in a graph database (Neo4j) and relational database (PostgreSQL/pgRouting).

2. RESEARCH METHODOLOGY

The evaluation methodology designed to compare both systems involves an objective benchmark comparison based on system documentation and experience. The following section describes data used for the benchmark and specifies hardware and software used for the system being tested.

2.1 Dataset

The graph data used in this benchmark is based on Austria road dataset from the OpenStreetMap (OSM) project [25]. OSM data model uses nodes (a point with coordinates), segments (directed link between two nodes) and ways (ordered list of segments) to represent the transportation network [26]. Any of these objects can contain tags. Tags are key/value pairs denoting the type and properties of the object. Basically, OSM is a form of a sparse directed property graph but not stored as such in a native form. OSM nodes represent nodes of the graph and ways form relation between those nodes. Tags can be used as property but in our case we use length of a way as a property for weight in the shortest path calculation. Raw OSM data are stored in the XML based format and as such cannot be imported into Neo4j or PostgreSQL database. In order to have consistent data in both databases, OSM data were converted by using osm2po tool [27]. The osm2po creates a graph representation of provided OSM dataset for direct import into PostgreSQL database. After importing data into PostgreSQL, the data were exported into acceptable cvs format for an import into Neo4j database. A small Java program was created that batch imported cvs file into Neo4j database with creating appropriate indexes. The data insertion time metric was not measured because of different data storage models.

The converted data in PostgreSQL were stored as a table with columns: *id*, *source*, *target* and *cost*. These columns represent relations between nodes with cost (length of the path) as a weight. In Neo4j database, each edge has a non-negative weight ω_i equal to the length of the path. Both of these two data models form the same graph of around 630,000 nodes and 750,000 relations.

2.2 Experimental setting

A benchmark is created in such a way that it can be repeated, data used is open and all of the fine tuning is described in the official documentation [28]. For both of the databases the same dataset was used. We used Neo4j enterprise server version 1.7.2 and pgRouting 1.05 on top of PostgreSQL 8.4 database. The recommendations found in official documentation of each database were used for both setups. For Neo4j case, we provided Java Virtual Machine (JVM) maximum amount of memory available on a testing computer and applied gcr cache type. Cache is a fast storage mechanism used to temporarily store data for future need. Neo4j grc cache type is a special type of cache which provides means of assigning a specific amount of memory for loaded nodes and relationship for the purpose of fast insert/lookup operations [28]. During all of our experiments, we monitored available RAM and possible swap hits to disk. Swap disk (also called virtual memory) is temporary space used on a hard disk as RAM. It is used in cases when data do not fit into RAM. It is slow, inefficient and should be avoided.

Tests were performed with cold and hot setup to determine the time needed for a database to load data into memory. The measured cold time is the time required to finish a given set of queries immediately after flushing caches or after reboot. In reality this is achieved by rebooting the system [29]. The hot test run consists of executing the same set of queries as from the cold run, in the same sequence order. The measured hot time is the time required to finish a given set of queries immediately after performing the cold run without flushing any cache or restarting the database [29]. All of the cold tests were executed with cleared disk cache at

operating system level. This is especially important for PostgreSQL database which relies heavily on operating file system cashing. Hot tests were executed right after the cold test with the same queries to assure the use of cached data. Both of the databases were monitored to assure cache hits in hot tests. The cold test begins with database and disk cache completely empty. Empty cache ensures that memory contains no data and that every new test starts independently from the previous one. At least three hot runs were made to assure time stability and that data was cached in running hot test. The client computer, from which the testing was performed, does not cache requests, so all of the tests (cold and hot) were executed with cache empty on the client computer.

A 190 pairs of nodes from the dataset described in the previous section were randomly chosen. These pairs represent locations between which the shortest paths is to be calculated on a given transportation network. The distribution of returned path lengths expressed in the number of nodes is shown in *Figure 1*.

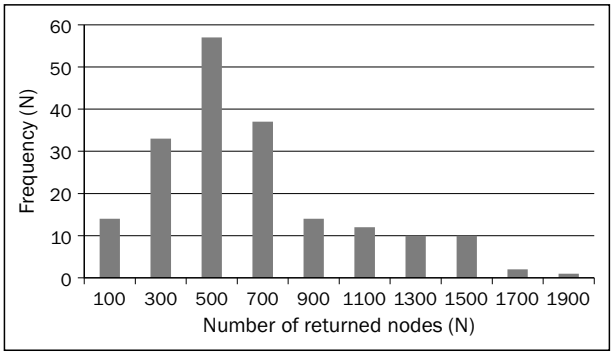

Figure 1 - Distribution of returned path lengths in number of nodes

Our tests were executed with one, two, four, eight and sixteen threads to simulate concurrent requests on the server. In the context of performance testing, thread is the number of concurrent connections to the database or application. Each thread executes its queries completely independently of other threads. We have not been able to test more than 16 threads because of hardware limitation.

Our experiments were conducted on a computer equipped with Intel Core i5 750 at 2.67GHz and 6 GB of RAM and installed Ubuntu Server 10.04 LTS 64-bit operating system with kernel version 2.6.32-28-server and OpenJDK Server VM. For the testing the Apache JMeter was used on a separate computer in the same isolated gigabit network.

3. RESEARCH RESULTS

Based on the official Neo4j documentation [28], JVM gcr cache type is recommended for Neo4j in enterprise environment and as seen in *Figure 2*, our findings proved to be true with Dijkstra shortest path queries. Memory and cache type are two configuration changes that proved to be the most influential ones on the performance of Neo4j routing calculation among all of the recommendations found in Neo4j documentation [28]. Although specified in the documentation as a recommendation flag in production environment, we found that using UseConcMarkSweepGC or UseSerialGC JVM flag slows down Dijkstra shortest path computation by around 30% with one-thread tests. UseParallelGC flag did not have any noticeable effect on computational performance. These flags are part of the mechanism called garbage collector which handles memory in applications written in Java programming language. PostgreSQL and pgRouting on the other hand, did not require any fine tuning to optimize their performance.

All of the presented results do not have swap hits to the disk and maximum available RAM was never achieved in any of our successful tests.

All of the test runs proved to have stable time as can be seen in *Figure 2*. None of the time measurements presented in this paper were averaged in any way. Memory consumption presented in *Figure 3* is the peak (maximum) value from both cold and hot runs.

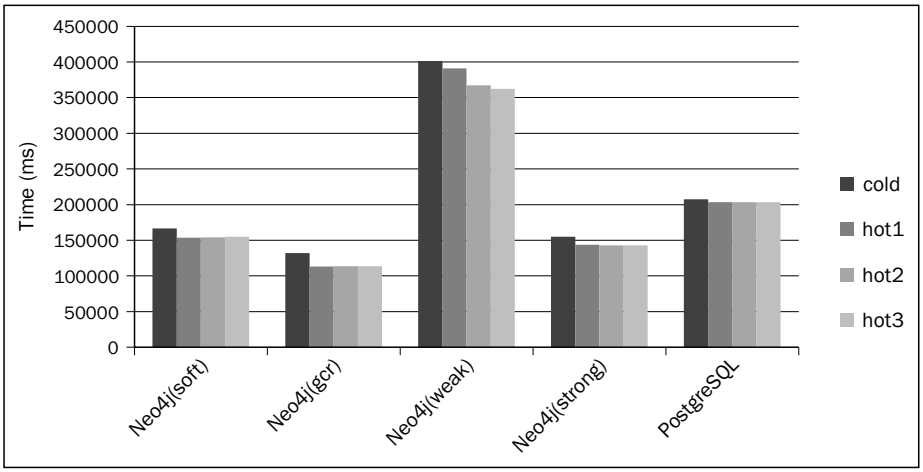

Figure 2 - Duration of cold and three hot tests for Neo4j(by cache type) and PostgreSQL

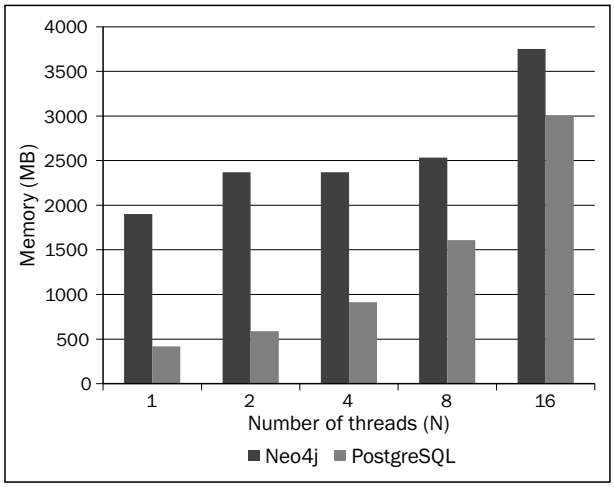

Figure 3 - Peak memory consumption by the number of threads

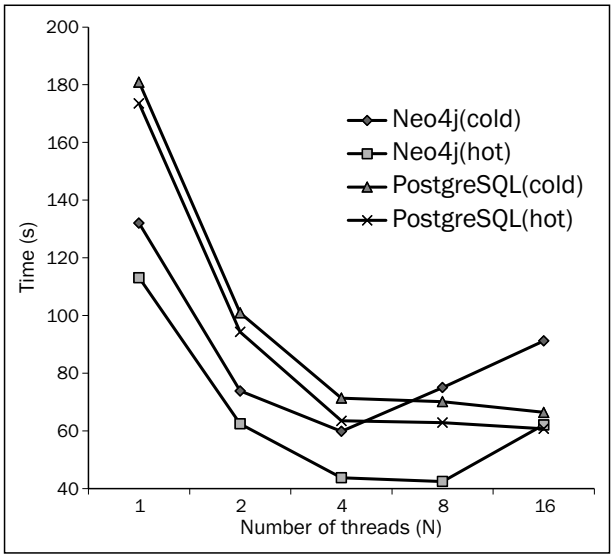

Figure 4 - Time required for cold and hot test to finish for Neo4j and PostgreSQL

Figure 4 summarizes the results of the benchmark for a different number of threads on cold and hot test for Neo4j and PostgreSQL database.

As we can see in *Figure 4*, threads over 16 increase computation time for both of the database systems as well as memory consumption as seen in *Figure 3*. The maximum number of threads depends on the running hardware. It was noted that hot tests, after the first one, did not change, as presented in *Figure 2*. This demonstrates that all of the data were loaded into the memory and none were released after the query finished.

Figure 5 shows constant time between requests for PostgreSQL calculation in contrast to variable time for Neo4j. This means that it would take the same time to load data for a query between two neighbouring nodes or for a full graph traversal, plus the time needed for calculating the shortest path.

4. DISCUSSION

The first assumption is that if some graph data structure is stored in a graph database, then the shortest path calculations must perform better because the data are stored in their native form. We demonstrated that graph database is 30% to 35% faster in calculation of the shortest path in transportation use case but uses more memory, from 20% to 75%, depending on the number of used threads.

The first full graph database performance analysis was presented in paper [9] which implemented queries of HPC Scalable Graph Analysis Benchmark v1.0 (HPC-SGAB) [24]. HPC-SGAB was designed by several leading researchers from academia and industrial companies. In their paper, the authors tested the performance of four graph database systems on a synthetic generated graph: Neo4j, Jena, HypergraphDB and DEX. The test was composed of four kernels: edge and node insertion performance, measuring time needed to find a set of edges that meet a condition, measuring time spent on building a subgraph and traversal performance over the whole graph. The result analysis of this paper was the basis for graph database of choice for our research. The results showed that DEX and Neo4j were the most efficient GDBs at the time of writing.

In contrast to databases, the routing can also be done with specialized in-memory routing engines. Currently, there are several in-memory routing engines available such as osm2po, Open Source Routing Machine (OSRM) or GraphHopper. Comparing the special-

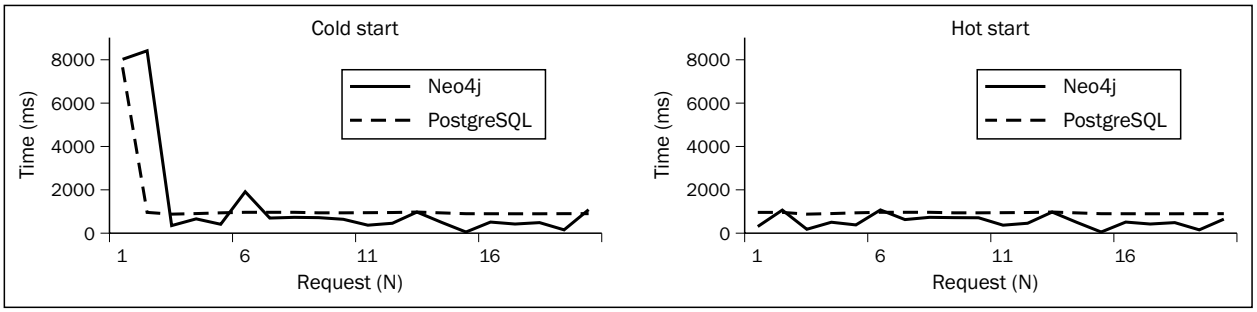

Figure 5 - Cold and hot test for first 20 requests, one thread

ized in-memory routing engines with full-fledged database system would not yield fair comparison because such routing solutions do not provide any of the database management system abilities which provide substantial performance overhead. PostgreSQL and Neo4j are general purpose database management systems and their main purpose is not routing engine but managing and storing of structured and semi-structured data. Although pgRouting is also in-memory routing engine it is still based on data provided by the underlying database management system. pgRouting does not retrieve graph data structure directly from memory for each shortest path calculation as in-memory routing engines do. It relies heavily on the underlying database for data retrieval and manipulation.

Dijkstra algorithm was chosen because it was the only algorithm implemented on both solutions. Because both of the database systems have their source code openly available, it can be seen that they both have implemented very similar Dijkstra algorithms. The obvious difference in implementation is because Neo4j uses graph traversal queries whereas pgRouting uses pure mathematical calculation. Our assumption was that GDB would have more algorithm implementations for the shortest path calculation. Both of the tested systems also provide A* shortest path algorithm, but it cannot be tested because its implementation in Neo4j is only available for the embedded database version. Using the embedded version of the database would provide inconsistent results as it would be directly tied to the application. Another thing that has to be taken into account is that Neo4j Dijkstra implementation is closely tied to the Neo4j traversal speed and is still in major development. On the other hand, pgRouting depends on PostgreSQL only for data retrieval, not for the calculation and in most cases of use does not benefit from database indexes. The developers of Neo4j announced bidirectional traversal algorithm which will greatly increase the speed of calculation. Both database systems have the source code openly available, and it could be easily concluded that both use the same Dijkstra implementation.

At the beginning of our testing, we wanted to use default configuration for both of the database systems, but that was not possible in case of Neo4j. Neo4js default configuration proved to be insufficient for any Dijkstra shortest path calculation on any of our datasets because it is optimized for smaller localized traversal queries, which have smaller memory requirements. The shortest path algorithms usually use full graph traversal queries, which heavily depend on the available memory. Our findings are that none of the recommendations for PostgreSQL had any noticeable impact for the pgRouting performance. Another thing that has to be noted is that Neo4j server communicates with clients via REST API over HTTP. This creates small overhead on performance but only to the delivery of the results from the server, not on the calculation of the shortest path. This overhead is minor comparing to the time for the calculation of the shortest path and was ignored in our test case.

For every query (concurrent or not) that uses pgRouting function, PostgreSQL database will load the data into the memory, compute the shortest path and release data from memory. For example, if two queries are sent at the same time on the same dataset, every query will take its own data into memory and compute the shortest path, and release it after the computation. There is no memory sharing with pgRouting function and memory consumption will be doubled in that case. The number of possible concurrent queries on PostgreSQL depends only on the size of the data and available memory. Neo4j and pgRouting function have different approach in handling of loading data into the memory. Neo4j only loads data required for the current query and that data is shared between requests, if the data is still in the memory.

As seen in the previous section, data used for our test are from the Austria OSM dataset. Tests with larger road networks, e.g. road network from Germany, the Netherlands or Italy OSM dataset led to the lack of memory. Another problem that was encountered during testing is infinite traverse loops on some OSM datasets, e.g. the Czech OSM dataset. This is due to the implementation of Dijkstra algorithm in Neo4j. Any position in the graph may be revisited. This is contrary to the default option for other Neo4j traversal queries, which is that no node in the entire graph may be visited more than once. On some specific graph layouts this creates infinite loops during Dijkstra shortest path calculation. Until the writing of this paper this was still default for the Dijkstra implementation in Neo4j.

5. CONCLUSION

The graph database management systems are not routing engines and are not suitable for full graph traversal, which is used in the shortest path calculations. Their primary purpose is local graph traversal based on the property graph in cases of use such as social networks, fraud detection, recommendation engines, resource authorization, or computer network management. Although in most cases Neo4j outperforms pgRouting, the Neo4j “greed” for memory has to be considered. This is especially important for large transportation networks. If the memory is not an issue, then graph database is the right choice for the shortest path calculation.

MARIO MILER, dipl. ing.
E-mail: mmiler@geof.hr
Dr. sc. **DAMIR MEDAK**
E-mail: dmedak@geof.hr
DRAŽEN ODOBAŠIĆ, dipl. ing.
E-mail: dodobas@geof.hr
Geodetski fakultet, Sveučilište u Zagrebu
Kačićeva 26, 10000 Zagreb, Hrvatska

SAŽETAK

USPOREDBA UČINKOVITOSTI ALGORITMA NAJKRAĆEG PUTA U GRAF I RELACIJSKOJ BAZI PODATAKA NA PROMETNOJ MREŽI

U području geoinformatike i prometnih znanosti, najkraći put izračunava se na graf podatkovnoj strukturi od kojih se uglavnom sastoje cestovne i prometne mreže. Ovi podaci se često pohranjuju u razne sustave baza podataka. Mnoge aplikacije koje koriste prometne mreže zahtijevaju izračun najkraćeg puta. Cilj ovog istraživanja je usporediti učinkovitost Dijkstra algoritma najkraćeg puta u PostgreSQL (koristeći pgRouting) i Neo4j graf baze u svrhu određivanja razlike u brzini izračuna najkraćeg puta. Ispitivanje je izvršeno na prosječnom računalu koristeći OpenStreetMap cestovnu mrežu. Unatoč tome što se Neo4j graf baza smatra pogodnom za izračun najkraćeg puta na prometnim mrežama, činjenica je da i takav način ima cijenu. Dokazano je da u Neo4j bazi, memorija računala može biti problem, posebno kod velikih prometnih mreža.

KLJUČNE RIJEČI

pgRouting, OpenStreetMap, Dijkstra, testiranje, Neo4j, PostgreSQL

REFERENCES

[1] **Dreyfus, S.E.**: *An Appraisal of Some Shortest-Path Algorithms*. Operations Research. 1969 May 1;17(3):395–412

[2] **Golden, B.L.**: *Shortest Path Algorithms: A Comparison*. Massachusetts Institute of Technology, Operations Research Center; 1975

[3] **Cherkassky, B.V.**, **Goldberg, A.V.**, **Radzik T.**: *Shortest paths algorithms: theory and experimental evaluation*. 1994 Jan 23;516–25

[4] **Cormen, T.H.**, **Leiserson, C.E.**, **Rivest, R.L.**, **Stein, C.**: *Introduction to Algorithms*, Third Edition. BOOK. The MIT Press; 2009. p. 1312

[5] **Zlatanova, S.**, **Baharin, S.S.K.**: *Optimal Navigation of First Responders Using DBMS*. 3rd International Conference on Information Systems for Crisis Response and Management 4th International Symposium on GeoInformation for Disaster Management. 2008;541–54 606

[6] **Tuot, C.**, **Obradovic, D.**, **Fichter, F.**, **Dengel, A.**: *CRUV - A Collaborative Route-Planning System for Utility Vehicles*. 2010;

[7] **Innerebner, M.**, **Böhlen, M.**, **Gamper, J.**: *ISOGA: a system for geographical reachability analysis*. Liang SHL, Wang X, Claramunt C, editors. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. pp. 180–9

[8] **Codd, E.F.**: *A relational model of data for large shared data banks*. 1970. MD computing computers in medical practice. ACM; 1970;15(3):162–6.

[9] **Dominguez-Sal, D.**, **Urbón-Bayes, P.**, **Giménez-Vañó, A.**, **Gómez-Villamor, S.**, **Martinez-Bazán, N.**, **Larriba-Pey, J.**: *Survey of graph database performance on the hpc scalable graph analysis benchmark*. Web-Age Information Management. Springer; 2010;37–48

[10] **Stonebraker, M.**, **Cetintemel, U.**: *"One Size Fits All": An Idea Whose Time Has Come and Gone*. 21st International Conference on Data Engineering ICDE05. Ieee; 2005;0(Icde):2–11.

[11] **Strauch, C.**: *NoSQL Databases*. Lecture Notes Stuttgart Media. Hochschule der Medien, Stutgart; 2010;1–8.

[12] **Orend, K.**: *Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer*. Architecture. Citeseer; 2010. p. 100.

[13] **Tiwari, S.**: *Professional NoSQL*. 1 edition. Wrox; 2011.

[14] **Sadalage, P.J.**, **Fowler, M.**: *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 1 edition. Addison-Wesley Professional; 2012.

[15] **Ciglan, M.**, **Averbuch, A.**, **Hluchy, L.**: *Benchmarking Traversal Operations over Graph Databases*. 2012 IEEE 28th International Conference on Data Engineering Workshops. IEEE; 2012. p. 186–9.

[16] **Angles, R.**, **Gutierrez, C.**: *Survey of graph database models*. ACM Computing Surveys. ACM; 2008;40(1):1–39.

[17] **Euler, L.**: *Solutio problematis ad geometriam situs pertinentis*. Commentarii Academiae Scientiarum Imperialis Petropolitanae. Nature Publishing Group; 1736;8:128–40.

[18] **Dominguez-Sal, D.**, **Martinez-Bazan, N.**, **Muntes-Mulero, V.**, **Baleta, P.**, **Larriba-Pay, J.L.**: *A discussion on the design of graph database benchmarks*. 2010 Sep 13;25–40.

[19] **Rodriguez, M.A.**, **Neubauer, P.**: *Constructions from Dots and Lines*. Science. American Society for Information Science and Technology; 2010;X(X):35–41.

[20] NeoTechnology. *The Neo database - A Technology Introduction*. 2006; Available from: http://dist.neo4j.org/neo-technology-introduction.pdf

[21] **Seng, J-L.**, **Yao, S.B.**, **Hevner, A.R.**: *Requirements-driven database systems benchmark method*. Decision Support Systems. 2005 Jan 1;38(4):629–48.

[22] *TCP. Transaction Processing Performance Council* [Internet]. 2011 [cited 2011 Sep 21]. Available from: http://www.tpc.org/default.asp

[23] **Ray, S.**, **Simion, B.**, **Demke, Brown A. Jackpine**: *A benchmark to evaluate spatial database performance*. Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE; 2011. p. 1139–50.

[24] **Bader, D.A.**, **Feo, J.**, **Gilbert, J.**, **Kepner, J.**, **Koester, D.**: *Hpc scalable graph analysis benchmark*. Citeseer. Citeseer; 2009;(HiPC 2005):1–10.

[25] **Coast, S.**: *OpenStreetMap project* [Internet]. 2013. Available from: http://www.openstreetmap.org/

[26] **Ramm, F.**, **Topf, J.**: *Towards a New Data Model for OpenStreetMap* [Internet]. 2007 p. 1–42. Available from: http://www.remote.org/frederik/tmp/towards-a-new-data-model-for-osm.pdf

[27] **Moeller, C.**: *osm2po* [Internet]. 2011. Available from: http://osm2po.de/

[28] NeoTechnology. *The Neo4j Manual v1.7.2* [Internet]. NeoTechnology; 2012. Available from: http://docs.neo4j.org/chunked/1.7.2/

[29] **Naphtali Rishe, A.V.**: *A Benchmarking Technique for DBMSs with Advanced Data Models*.