Meth Hunter: an Extension of Automated Intelligence Systems via Graph Database

Mark Blair
CEO & CIO
DAGIR Co.
Erie, Pennsylvania

Yunkai Liu
Associate Professor
Computer and Information Science Department, Gannon University
Erie, Pennsylvania
Email: liu006 [AT] gannon.edu

Theresa M. Vitolo
Associate Professor
Computer and Information Science Department, Gannon University
Erie, Pennsylvania

Abstract— **Automated intelligence analysis was born into a skeptical community dealing with high stakes dilemmas, where often outcomes are measured in loss of human life. The massive amount of data require analysts to be masters of identifying "indicators" intuitively foreshadowing intelligence targets and masters of modeling the indicators in analysis systems. The paper explores the application of graph theory through graph databases to determine how the index-free adjacency of graph databases can be exploited to improve processing time. The study utilized the Meth Hunter, an analytic machine designed to identify methamphetamine conspirators by analyzing pseudoephedrine purchase records as a case study. The graph database (Neo4j) was compared to a SQL relational database (WAMP). Neo4j demonstrated superior performance in identifying and retrieving relationships between data points. Neo4j successfully demonstrated the ability to implement such a strategy and extend the horizons of traditional data mining systems.**

Keyword—Automated intelligence; big data; graph database; SQL database;

I. INTRODUCTION

Data analytics allow analysts to absorb, digest, and synthesize inordinately larger bodies of information, all with the complication of extensive process time. Further, when modeling analytic targets, an analyst must attempt to recreate the real-world target structure as accurately as possible in the model.

Research in the area of automated analytics is important to demonstrate the capabilities gained and to bolster confidence in Structured Analytic Techniques (SATs) [3] by a partially skeptical intelligence community.

The paper uses the example of analyzing the logs derived from pseudoephedrine sales collected under the Combat Methamphetamines Epidemic Act (CMEA). Most meth labs (e.g., the most popular synthesis methods) require large amounts of pseudoephedrine to operate. Multiple individuals (referred to as "Smurfs") make multiple purchases of pseudoephedrine-based products to acquire the necessary quantities of the pseudoephedrine. Data associated with these Smurfs are signatures of social networks linked by a conspiracy to produce meth hidden in massive data bodies. The Meth Hunter is an analytic system analyzing CMEA data to find indicators of Smurfs using a relational database management system (RDBMS). Due to the nature of the highly connected target data, the system can take more than 61+ hours to conduct a full analysis on as few as 90,000 records. The paper explores the possibility of improving the processing runtime and of bolstering the analytic capabilities of the Meth Hunter by employing a graph database management system (GDBMS). The proposed system uses a GDBMS-RDBMS hybrid system.

II. METH HUNTER DESIGN AND METHODOLOGY

The Meth Hunter is comprised of a php application accessing a locally hosted WAMP server v. 2.0h. The CMEA requires records of all purchases of pseudoephedrine-based products be maintained and made available to law enforcement professionals for the purposes of interdicting methamphetamine producing conspiracies or more commonly termed, "meth rings". Most methamphetamine production methods require the key ingredient pseudoephedrine (PSE). The CMEA limits the amount of PSE an individual may purchase to 3.6 grams a day and 9 grams a month. Persons purchasing in excess of these limits can be charged with a precursor violation (possession of the chemical precursor of an illicitly manufactured drug). In order to acquire enough PSE, meth rings utilize multiple individuals to make multiple, small quantity purchases of PSE-based products. These individuals, known as "Smurfs," rarely purchase in excess of the legally allowed limits. Smurfs from the same meth ring represent a social network; their organized behavior leaves a signature in the data providing vital intelligence to law enforcement professionals. The Meth Hunter is specifically designed to identify these individuals and their meth rings by detecting those signatures.

The challenge facing the Meth Hunter is that no single data point provides any indication of suspicion. Indication

of suspicion lies in the relationships between data points. Any single data point is simply the record of a purchase and the personal, legally-required information of the buyer. Only through collation and analysis of multiple data points can any indication of involvement in a meth conspiracy be determined.

A. Indicators

Heuer and Pherson define the following: "Indictors are observable phenomena that can be periodically reviewed to help track events, spot emerging trends, and warn of unanticipated changes. An indicators list is a pre-established set of observable or potentially observable actions, conditions, facts, or events whose simultaneous occurrence would argue strongly that a phenomenon is present or is very likely to occur" [3]. The Meth Hunter's foundation is most similar to an expert system based on the indicators used by law enforcement professionals to identify meth conspirators. These indicators are primarily for identifying individual meth conspirators not the organized cells. Meth Hunter developers (intelligence analysts and technologists) extended this analytic framework to identify more indicators of both individuals and networks (meth rings).

B. Individual Indicators

Several indicators of a single individual's purchasing behavior can increase the probability about participation in a meth conspiracy.

- Frequency of purchases
- Total number of purchases
- Amount of PSE purchased
- More than one purchase from more than one dispenser on a single day
- Distance from residence to dispenser
- Use of multiple, different, forms of identification

C. Conspiracy Indicators

Conspiracy indicators predict instances where multiple individuals conduct organized behavior to achieve a common goal. Conspiracy indicators impart a stronger prediction than individual indicators because conspiracy indicators have fewer competing hypotheses. For instance, an individual who purchases PSE products frequently in excess of recommended doses could be simply abusing PSE as a recreational drug or as an alertness or study aid. However, such individuals have few reasons to organize their behavior to purchase pseudoephedrine. In fact, aside from activities associated with illicit/clandestine chemistry no other reason is identified for individuals to organize in such a manner. Artifacts creating a competing hypothesis normally involve an intricate set of environmental constraints, such as: factory workers working 3rd shift, and buying PSE at a dispenser across the street from the factory during their break. These types of artifacts are easy to detect and confirm. Conspiracy indicators include:

- CoBuys
- Purchasing routes
- Shared addresses
- Shared last names
- Consistent Product/Quantity/Stores

D. CoBuys

A CoBuy is an instance where two individuals purchase PSE-based products within a short time period of each other. For instance, if Johnny buys PSE at 12:00 pm and Sally buys PSE at 12:09 pm, a CoBuy would be noted. Anyone may be coincidently standing behind Johnny in line or buying PSE on the other side of town around the time of Johnny. For this reason, a single CoBuy between two individuals has no indication; however, when two individuals participate in a CoBuy on multiple occasions, these events quickly exceed the probability of happenstance and random synchronization. Two competing hypotheses can explain this reoccurring relationship: 1) the individuals are engaging in an organized conspiracy or 2) some environmental constraint has exerted its influence over the individuals. The latter is a much less likely explanation. These competing hypotheses are easy tested by investigators to determine whether the individuals merit further investigation.

To understand the indicative power of CoBuys imagine two individuals: Johnny living in California and Sally living in Virginia. Johnny and Sally purchase PSE-based products regularly. Curiously, Johnny and Sally always purchase PSE products within 15 minutes of each other despite living on opposite sides of the country and having a four-hour difference in time zones. One explanation of this behavior says Sally and Johnny have or had been given the same schedule. Another might be Johnny and Sally receive phone calls from an individual directing them when to purchase. Other explanations are possible, but require an intricate and complex explanation. While this anomaly is not an absolute indictment of guilt in some illicit conspiracy, it does merit further investigation.

Of course, CoBuys made in a smaller geographic area show a stronger indication of cooperation. What if Johnny and Sally always purchased at the same store within 10 minutes of each other? What if Johnny and Sally did this in multiple stores a day and even crossed state lines to purchase more? CoBuy indication is dynamic. CoBuys at

the same store are more powerful than CoBuys at different stores. Smurfs will often share a vehicle to drive to different dispensers. CoBuys where the participants buy the same product are more powerful than CoBuys where the individuals buy different products.

CoBuys are by nature a relationship between two purchases. The accumulation of CoBuys is a relationship between two buyers (purchasers). In order to develop a relationship between two buyers each CoBuy in the data must be found and recorded resulting in a timely process, theoretically requiring n(n-1)/2 operations. The term "theoretically" is used above because WAMP optimization processes are opaque to the developer. WAMP uses filtering or similar processes to reduce the number of necessary operations. However, experimental results seem to indicate SQL queries do indeed perform n(n-1)/2 operations. CoBuy queries resemble the following SQL code:

```
SELECT a.purchaser_id, b.purchaser_id
FROM raw_data AS a, raw_data AS b
WHERE a.purchaser_id != b.purchaser_id
AND a.purchase_date < b.purchase_date
AND b.purchase_date – a.purchase_date < 10 MINUTES
```

E. Purchasing Routes

As mentioned above Smurfs often share the same vehicle to drive between PSE dispensers. Purchasing routes can be identified in two computationally expensive manners. 1) If a graph is developed where nodes/vertices are dispensers and links/edges are drive-times between, then a series of purchases and the time between them can be compared to the dispenser containing those locations. The development of the dispenser graph is computationally expensive, as the drive-time between every pair of dispensers must be calculated (n^2 drive-times). However, the graph need only be created once. 2) Individuals making purchases on the same day regularly can be isolated. A dispenser list is then created based on the purchase locations. Then drive-times between those locations are calculated and compared to the time intervals of the purchases. In either case, the process is so computationally expensive it restricts running the the associated queries on full data sets until individuals have been identified as suspicious. Then, the computations can be conducted on smaller data sets to minimize execution time.

F. Other Attributes and Indicators

Since most PSE-based product sales are legitimate, multiple indicators and evidence offer more credibility Many smaller relational indicators are often present in the data substantiating the PSE-abuse premise. For instance, it is often the case that husbands and wives, brothers and sisters, etc. are involved in the same conspiracy. Additionally, individuals may share the same residence. Meth cooks tend to be picky and want a specific product. The Meth Hunter has actually identified Smurfs based on their habit of returning one PSE product in favor of another. A battery of "subtle" indicators can strengthen more solid indicators, such as CoBuys or frequency of purchase. These subtle indicators are collected and augment the large analytic picture.

III. Extending Meth Hunter's Capabilities

Meth Hunter developers had to address the known behavior of individuals consciously thwarting the analytical efforts of law-enforcement. A modified version of the statistical approach, the Birthday Problem, was used. However, the mathematical equation was complicated and hard to understand. Prosecutors and agents feared the complexity of the analysis would be too difficult to convince a jury of guilt. Additionally, the analysis was too complex and specialized to standardize and automate in the Meth Hunter. Graph theory and graph database management systems were examined as an analytical paradigm to extend the Meth Hunter's capabilities

Given the time it requires the current Meth Hunter to calculate CoBuys (n (n-1)/2 operations), a RDBMS cannot identify these triangular relationships in an acceptable time frame; however, the Neo4j GDBMS provides a much more realistic time frame to conduct such analysis.

Cypher, Neo4j's query language, offers the function "MATCH CIRCULAR." The function identifies paths starting and ending at the same node. The function allows for the identification of the relationship type between each node; thereby ignoring all but the target edges (i.e. time between). Then, viable edges/links (i.e., WHERE timeBetween < 10) are filtered. Further filtering will prevent returning every permutation of a triangular relationship (i.e., date1 < date2 < date3). The viability of such a query was tested and detailed below. The experiment shows the power of graph theory to extend the capabilities of the Meth Hunter and current analytic computer systems in general.

IV. Experimental Design

The first experiment focuses on the CoBuy query to demonstrate any advantage Neo4j imparts to the Meth Hunter in terms of processing time. The second experiment is tests Neo4j's ability to identify triangular relationships in a realistic time frame. All experiments are conducted with real-world data.

The decision to use real-world data instead of a control dataset was made for several reasons. First, when importing

real-world data, inevitably erroneous data points due to processing mistakes such as in parsing or improperly entered data points will occur. The experiment seeks to test the technology's real-world performance and therefore, must operate over real-world data. Former designs of the Meth Hunter were abandoned specifically for their lack of robustness for dealing with improperly formed data. Second, real-world data has many characteristics, nuances, and anomalies not found in experimental data.

A Java application was created to run the SQL (RDBMS) and Cypher (GDBMS) queries to identify CoBuys. The queries were compared in terms of execution time and the returned result sets. Execution times were measured from within the Java application. This approach measures the amount for time it took to receive the data, as this more accurately represents actual wait time. The process was executed multiple, consecutive times with an increasingly-sized dataset.. Least squares methodology was used to project the difference in performance out to even large data sets.

The testing application followed a simple procedure. First, a number of records were transferred from one table in the RDBMS to the experimental table. Second, a graph was built based on the rows in the experimental table. Each row was converted to a subgraph containing four nodes: (1) A date node containing a date time group connects via a PURCHASED_ON relationship, to a purchased node, (2) the purchased node containing a record (row) identifier, (3) a purchaser (buyer) node containing a driver license number, and (4) a store (dispenser) node containing a store ID, connected to the purchase node via PURCHASED_BY and PURCHASED_AT relationships respectively. Next the Java application queried an ordered list of date, placing into an array. If the difference between to date-nodes' date time group was less than 60 minutes a TIME_BTWN relationship (edge) was built between them with the property of the exact difference (in minutes) between the two date nodes.

Efficiency or optimization in creating the graph was not included in the scope of the project. The real goal was to compare the capability of the different platforms executing the queries, not the graph-production speed. That said, this process was the most time consuming portion of the trials; the process was built for robustness not efficiency. Each node to be created was first queried to see if it exists before creating a new node. If it exists, no new node was created. Rather, links were created from the incoming record nodes to the existing node. A duplicate record would produce nothing.

Once the graph was created two queries were run: A RDBMS SQL query and a GDBMS Cypher query. Both queries returned purchases made within 10 minutes of each other. Both queries returned the same data: (1) the record ID of the first and second purchases, (2) the date of the first and second purchases, and (3) the time between the first and second purchases. The Java application produced two documents for each CoBuy. The first document captures the execution time of the respective CoBuy queries and a list of each identified CoBuy. The second document is a collection of metadata for each interval query in a trial. This metadata includes: the number of data points in the trial, the execution time of the RDBMS and GDBMS queries, the number of CoBuys identified, the execution time per CoBuy, the number of objects in the RDBMS (rows) and the GDBMS (nodes) databases, the ratio of RDBMS and GDBMS objects, and the number of relationships in the GDBMS.

Each trial design was repeated four times in a series of four independent trials. A trial consists of 10 interval queries where data is added incrementally to both databases. CoBuy queries are run on the databases between the data-addition intervals and appropriate data is recorded before the next increment of data is uploaded. For instance, several trials consist of ten increments of 1000 data points. Each trial was run four times, each starting with empty databases.

Although join tables may have improved the RDBMS on subsequent queries, the design would have required multiple query operations, one to enter new CoBuys into the join table and a second one to pull records joined by the join table. RDBMS advocates may argue that a join table is akin to the time between relationships created in the GDBMS; however, join tables do not provide the flexibility nor the ability to handle novel queries. For instance, a join table created to store pairs of data points related by time intervals less than 10 minutes apart will not be suitable for determining purchases less than 5 minutes or 15 minutes apart.

All queries conducted are "novel" meaning no results or abbreviated results have been stored from previous queries for subsequent processing. However, one last design issue relating to performance merits discussion. The version of Neo4j used (1.9.2) does not contain the ability to index nodes and/or relationships via Cypher. (Version 2.0 and above do.) Indexing on earlier versions is possible using a REST API, but not through the JDBC utilizing Cypher. While indexing will likely have a minimal effect on the GDBMS performance, it could make a difference in the performance of the RDBMS.

The queries utilized by the RDBMS and GDBMS are as follows:

SQL:

```
SELECT a.rd_id AS record1, b.rd_id AS record2,
a.purch_date AS dtg1, b.purch_date AS dtg2,
b.purch_date - a.purch_date AS time_btwn
FROM raw_data AS a, raw_data AS b
```

```
WHERE a.purchaserID != b.purchaserID
AND a.purch_date < b.purch_date
AND b.purch_date - a.purch_date < 1000
```

Cypher:

```
Start r=rel(*) MATCH pur1-[p1:PURCHASED_ON]
->date1-[r:timeBtwn]->date2
<-[p2:PURCHASED_ON]-pur2
 WHERE has(r.TIME_BTW)
 AND r.TIME_BTW < 10
 RETURN pur1.rd_id, pur2.rd_id, date1.purch_date,
date2.purch_date, r.TIME_BTW
```

The raw test data included 26,311 real-world data points. Each trial used a portion of these data points for testing.

Trial 1: 100-1000 queried at 100 data point intervals starting at first record.

Trial 2: 1000-10,000 queried at 1000 data point intervals starting at first record.

Trial 3: 1000-10,000 queried at 1000 data point intervals starting at 10,001st record.

Trial 4: 1000-20,000 queried at 1000 data point intervals starting at first record.

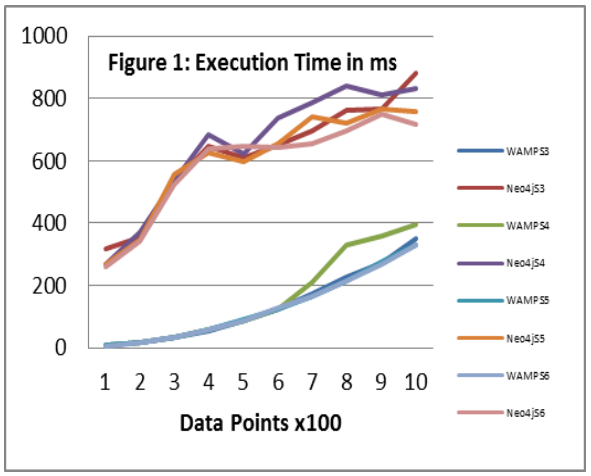
Trial 5: 1000-10,000 queried at 1000 data point intervals starting at first record with index on the purch_date column.

Data was filtered by 45 neighboring zip codes in a southern US state. The data was collected between March 22, 2013 and May 23, 2013. The first 20,000 records were utilized in various tests (March: 347 records; April: 11,987 records; May: 7665 records). The first 10,000 records contain 347 records from March, 1987 records from April, and 7665 records from May. Records 10,001 through 20,000 contain records from April 2, 7:00 am to April 28, 11:15 pm. The order in which data was received (from dispensers) was preserved to mirror a real-world application. The data includes multiple data sets for 96 dispensers (stores). All data records of returns were removed. Additionally, data records of "blocks" were removed. Blocks are instances where a person has exceeded the legally allowed limit and the monitoring system blocked further purchases. The most basic data cleaning queries were utilized, such as rejecting records with NULL in a non-NULL column; removing escape characters or actionable characters affecting .csv and .txt file parsing (such as: , ^ * \ / etc.); and removing records containing NULL values in key fields (such as purchaser_id). All other malformed data or other erroneous entries were preserved to mirror real-world operations. Some effort was made to ascertain how the two systems handled these data anomalies.

Note, Trial 2 and Trial 3 were conducted with the same amount of data at the same interval, but used different data (Trial 2: data points 1-10,000; Trial 3: data points 10,001-20,000). Data points 1 to 10,000 were recorded March 31, 2013 to May 23, 2013 and included records from 94 dispensers. Records from the individual stores ranged from 531 records to 1 record per store. Data points 10,001 to 20,000 were recorded April 4, 2013 April 28, 2013 and included records from 95 dispensers. Records from the individual stores ranged from 528 records to 1 record per store.

Trials to test the Triangular Relationship query were also run using the Cypher query:

```
START n=node(*)
MATCH cir=n-[r1:timeBtwn]-n2-[r2:timeBtwn]-n3-[r3:timeBtwn]-n
WHERE n.type = 'date'
AND n.purch_date < n2.purch_date
AND n2.purch_date < n3.purch_date
AND r1.TIME_BTW < 5
AND r2.TIME_BTW < 5
AND r3.TIME_BTW < 5
RETURN n.purch_date, r1.TIME_BTW, n2.purch_date,
r2.TIME_BTW, n3.purch_date, r3.TIME_BTW
```



A Java application was created to employ this query. Execution time and result sets were assessed for usability and compared against CoBuy queries' execution time and result set. The execution time was assessed to ensure the end product could execute this type of analysis in a realistic (usable) timeframe. The number of triangular relationships, execution time, and execution time per triangular relationship were recorded.

V. Empirical Results

A. Performance Testing

Neo4j demonstrated superior performance in execution time in identifying and retrieving CoBuys. Neo4j took an average of 2.9 seconds for 10,000 data points while WAMP required an average of 22.9 seconds to conduct the same operation.

Even through Trial 3 and Trial 4 used different datasets both databases found a remarkably similar number of CoBuys (Trial 3 RDBMS: 5481, GDBMS: 6068; Trial 4 RDBMS: 5579, GDBMS 6236). Such results are not unexpected since the overwhelming majority of PSE purchases are legitimate purchases and CoBuys (single instance CoBuys) are inevitable. Also, the large majority of these CoBuys are, most likely, the result of happenstance, not meth conspiracies. However, the first 10,000 data points included reporting with a wider range of dates. For instance this dataset included 1987 data points for the month of April ranging from April 1, 2013, 1:09 am to April 30, 2013, 10:41 pm. The data set consisting of data points 10,001 to 20,000 included records from April 2, 2013, 7:00 am to April 28, 2013, 11:15 pm. Therefore, in this example, a large number of CoBuys can be expected between data points in the first and second datasets. If this data pattern holds true the growth of CoBuys will not be additive as more data is added, yet instead could be multiplicative or even exponential.

With smaller datasets (datasets < ~2100 data points), the WAMP outperformed Neo4j. However, even observing performance over smaller datasets, the tendency for Neo4j's performance to follow a more logarithmic path and WAMP's performance tended to follow a more exponential path is evident; (see Fig. 1).

Figure 1. Execution time in ms.

Additionally, the execution time per CoBuy drops sharply for the Neo4j database while the execution time per CoBuy slightly increases for WAMP; (see Fig. 2).

Several trials were conducted to determine where the performance of the two databases crossed in terms of execution time. Neo4j demonstrated superior performance on datasets larger than 2100 data point; (see Fig. 3).

The difference in performance becomes evident when the dataset increases to 10,000 data points; (see Fig. 4).

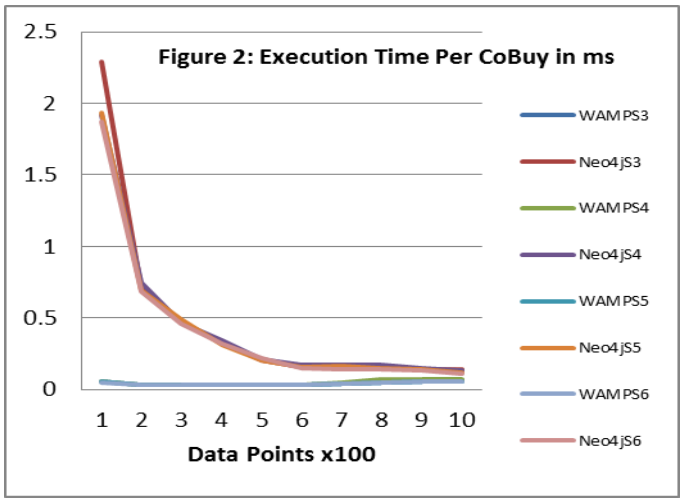


Figure 2. Execution time per CoBuy in ms.

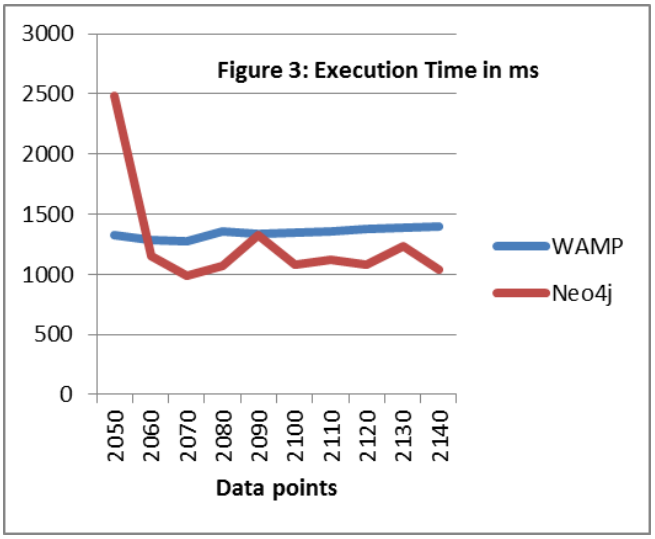


Figure 3. Comparison of execution time for datasets>2100 points, in ms.

On average, Neo4j executed the CoBuy query on 10,000 data points 20 seconds faster than WAMP. Additionally, the disparity between the two systems quickly grows. Given that Neo4j takes an average of 2.8 seconds to conduct the query the difference is significant. Moreover, the WAMP database performance followed a smooth curve, seeming to indicate its performance is set by the size of the data and not the content. The WAMP performance was easily fitted to a power law ($0.0003X^{1.98}$) using least squares methodology. The equation made fairly accurate prediction with a standard deviation of 47.99 milliseconds. Neo4j performance on the other hand was more erratic it seemed to respond more to the number of CoBuys present in the data rather than to the volume of the data. Further mathematic proofs were not pursued as it was not the focus of the research at this time. Projecting out the database

performance using equations derived from least squares methodology, the following results were developed. (See Table 1.)

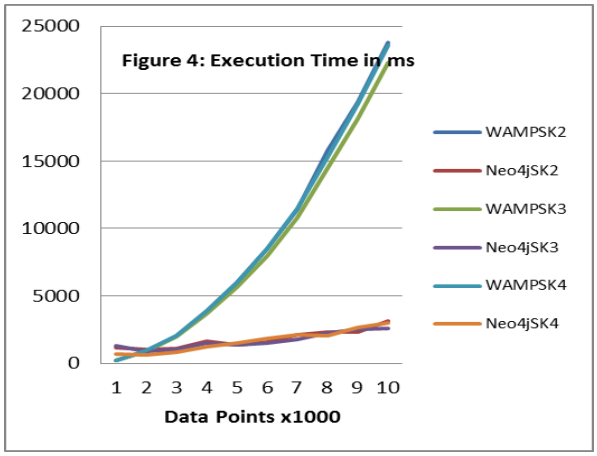


Figure 4. Comparison of execution time for large datasets, in ms.

TABLE 1: PROJECTED DATABASE PERFORMANCE

DBMS	Equation	100,000 Data Points	1,000,000 Data Points
WAMP	$0.0003X^{1.98}$	0.606 hours	57.536 hours
Neo4j	$31.55X^{0.47}$	0.0002 hours	0.006 hours

As noted above, since data is not always received in chronological order, the addition of new data is likely to contain CoBuys internal to the new dataset and CoBuys between data points in the new dataset and existing data points in the database. This could mean that, in the real world, CoBuys may grow much faster than projected from smaller datasets. Whether these predictions hold true with more robust data tests, Neo4j has demonstrated its ability to improve the performance of the Meth Hunter when identifying and analyzing relationships between data points.

When comparing the data objects in the WAMP (rows) to the data objects in Neo4j (nodes), Neo4j has an average of 2.6:1 ratio. Since Neo4j creates four nodes for each row, if they do not already exist, then Neo4j is cutting down on data redundancy. Additionally, observing this ratio at consecutive increments of 1000 data points it demonstrates a line with a continuous negative slope, starting at an average of 2.8 for 1000 data points decreasing to 2.6 at 10,000 data points. This is consistent with the findings of Vicknair et al. [28] who noted the disparity of required disk space from Neo4j and MySQL databases diminished as the size of the data set grew.

.

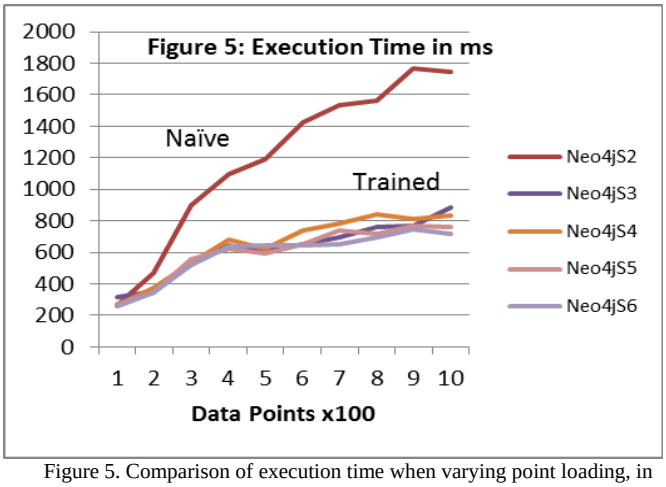


Figure 5. Comparison of execution time when varying point loading, in ms

Interestingly, Neo4j demonstrates learning in some sense. On the smaller dataset (1-1000 points, incremented by 100 data points) two different trial designs were run. Hence, two different states of the Neo4j database are conceived: Naïve and Trained. The Trained Neo4j is a database with 100-data-point increment sets are added at a time to the existing dataset of data that has already been uploaded and queried for CoBuys. The Naïve Neo4j is a database, in which the amount of data uploaded is incremented, but the database is cleared before the next trial, i.e., 100 data points are uploaded, the database is truncated, and then 200 data points are uploaded, and so on. The Trained Neo4j database showed significant improved performance over the Naïve Neo4j database.

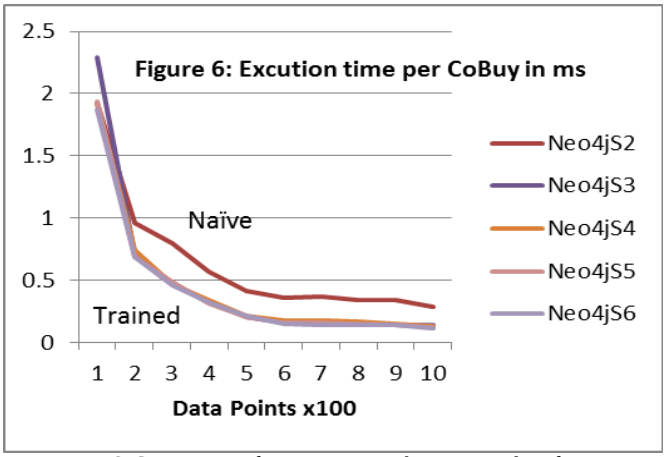


Figure 6. Comparison of execution times between modes of Neo4j, in ms.

The mechanism responsible for this improved performance is unclear. Each CoBuy query is run as if the database is naïve or in other words, has no knowledge of existing CoBuys. The query is designed to retrieve nodes connected to each "time between" relationship having a property of TIME_BTWN less than 10 minutes. This fact may indicate that the performance recorded above should not be expected for novel queries. However, the shape of the performance data still resembles a logarithmic curve versus an exponential curve. Furthermore, a graph of the Naïve Neo4j execution time per CoBuy also demonstrates a sharp decline resembling a negative power law, consistent but as dramatic as the Trained Neo4j; (see Fig. 6).

Very likely, a Naïve Neo4j database will still out perform WAMP (which demonstrates power law curve in execution time) on larger datasets, although its performance should be expected to be less than a Trained Neo4j database. WAMP did not demonstrate any training effects.

Indexing showed no effect on WAMP's performance. The most critical field for CoBuy queries is the date field. An index was placed on the purch_date field in the WAMP database and the CoBuy query was run against 10,000 records to ascertain whether indexing had any effect on the WAMPs performance. The results are detailed in the table below.

TABLE 2: TRIAL RESULTS INDEXED AND NOT INDEXED

	Execution time in ms			Execution time per CoBuy in ms	
Series	Indexed	Not Indexed	# of CoBuys	Indexed	Not Indexed
S1	23399	23142	60526	0.38766	0.38235
S2	23465	23786	60526	0.38769	0.39299
S3	23477	22277	60526	0.38788	0.36806
S4	23475	23996	60526	0.38785	0.39646
AVG	23454	23300.3	60526	0.38750	0.38496

Also, both Neo4j and WAMP seemed to optimize the processors well. This effect became evident quite accidentally. The data tests could take a significant amount of time. During one test, the first author began streaming a movie on the testing computer to pass the time. Quickly realizing the potential error of this, the time and increment of the trial running was recorded. Upon reviewing the trial results, the effect of the streaming was evident; (see Fig. 7). All subsequent tests were conducted without running other applications.

VI. SUMMARY

Neo4j found more CoBuys than WAMP. On average Neo4j found 1.09 CoBuys to each CoBuy WAMP found. CoBuys for each database were validated. First a Java program was created to loop through each CoBuy and check it against all other CoBuys found by the same database to ensure they were not duplicates. No duplicates were found. Second, another Java application was made to loop through each CoBuy, call the original records from the raw_data table, and recalculate the time between the original dates to ensure accuracy. All CoBuys were found to be accurate. Finally, a Java program was written to retrieve each CoBuy comparing compared the first and second record IDs, dates, and the time between values to the records in a table containing the CoBuys found by the other database. Out of 1000 data points WAMP found 5481 CoBuys, 3 of which Neo4j failed to identify. Neo4j found 6068 CoBuys, 590 of which WAMP failed to identify. The exact mechanism causing the disparity is not well understood; however, since all CoBuys could be validated, explaining the mechanism remains as future work.

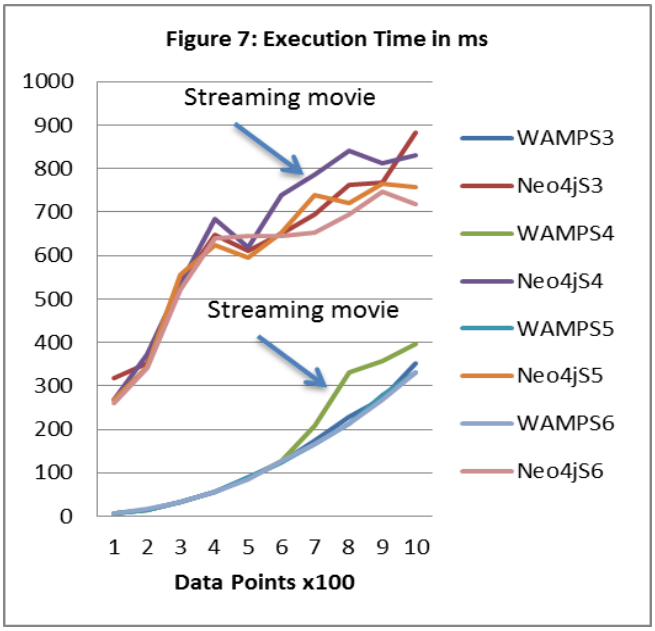


Figure 7. Influenced exceution time comparisions in ms.

REFERENCES

[1] R. J. Heuer Jr, Richards, "Psychology of Intelligence Analysis", Center for the Study of Intelligence, Central Intelligence Agency, 1999.

[2] P. Gogoi, "Knight Capital blames software for computer trading glitch", *USA Today*, 02 Aug 2012.

[3] N. N. Taleb, "The Black Swan: The Impact of the Highly Improbable", New York, Random House, 2007R.

[4] J. Heuer Jr., R. H. Pherson, "Structured Analytic Techniques for Intelligence Analysis", Washington DC, SQ Press, 2011.

[5] R. J. Heuer Jr., R. H. Pherson, "Structured Analytic Techniques for Intelligence Analysis", Washington DC, SQ Press, 2011.

[6] "US Army and Marine Corps Counterinsurgency Field Manual 3-24", 1st ed., Chicago, IL, UC Press, December 2006.

[7] D. Chakrabarti, C. Faloutsos, "Graph Mining: Laws, Tools, and Case Studies", Morgan & Claypool Publishers, Kindle Edition, Oct 2012.

[8] S. H. "Strogatz, Exploring Complex Networks", *Nature*, vol. 410, Mar. 2001, pp. 268-276.

[9] D. Chakrabarti, C. Faloutsos, "Graph mining: Laws, generators, and algorithms", *ACM Comput. Surv.* vol 38, issue 1, Article 2, June 2006.

[10] T. Washio, and H. Motoda, 2003. "State of the art of graph-based data mining", *SIGKDD Explor. News, vol.*5, issue 1, pp.59-68, July 2003.

[11] D. J. Cook, and L. B. Holder, " Substructure discovery using minimum description length and background knowledge", *J. Artificial Intel. Research*, vcol. 1, 1994, pp. 231-255.

[12] K. Yoshida, H. Motoda, & N. Indurkhya, "Graphbased induction as a unified learning framework", *J. of Applied Intel* ., vol. 4, 1994, pp. 297-328.

[13] Y. Jia, J. Zhang, and J. Huan, "An efficient graph-mining method for complicated and noisy data with real-world applications", *Knowledge and Information Systems, vol. 28*, issue 2, 2011, pp. 423-447.

[14] Z. Zhaonian, L. Jianzhong, G. Hong Gao, and Z. Shuo, "Mining Frequent Subgraph Patterns from Uncertain Graph Data", *Knowledge and Data Engineering, IEEE Transactions on* , vol.22, issue 9, 2010, pp.1203-1218.

[15] L. Jianzhong, Z. Zhaonian, and G. Hong, "Mining frequent subgraphs over uncertain graph databases under probabilistic semantics", *The VLDB Journal* vol. 21, issue 6, 2012, pp. 753-777.

[16] D. J. Cook and L. B. Holder, "Graph-based data mining", *Intelligent Systems and their Applications, IEEE* , vol.15, no.2, Mar/Apr 2000, pp.32-41.

[17] L. B. Holder and D. J. Cook. 2003. Graph-based relational learning: current and future directions," *SIGKDD Explor. Newsl.* vol. 5, issue 1, pp. 90-93, 2003.

[18] H. Yamasaki, Y. Sasaki, T. Shoudai, T. Uchida, and Y. Suzuki, "Learning block-preserving graph patterns and its application to data mining", *Machine Learning*, vol. 76, issue 1, 2009, pp. 137-173.

[19] K. Barmpis and D. S. Kolovos, "Comparative analysis of data persistence technologies for large-scale models," In *Proceedings of the 2012 Extreme Modeling Workshop* (XM '12), D. Di Ruscio, A. Pierantonio, and J. Lara (Eds.). ACM, New York, NY, 2012, pp. 33-38.

[20] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective", In *Proceedings of the 48th Annual Southeast Regional Conference* (ACM SE '10). ACM, New York, NY, Article 42 , 6 pages, 2010.

[21] R. De Virgilio, A. Maccioni, and R. Torlone, "Converting relational to graph databases", In *First International Workshop on Graph Data Management Experiences and Systems* (GRADES '13). ACM, New York, NY, Article 1, 6 page,2013.

[22] L. B. Holder and D. J. Cook. 2003. Graph-based relational learning: current and future directions," *SIGKDD Explor. Newsl.* vol. 5, issue 1, pp. 90-93, 2003.

[23] H. Yamasaki, Y. Sasaki, T. Shoudai, T. Uchida, & Y. Suzuki, "Learning block-preserving graph patterns and its application to data mining," *Machine Learning*, vol. 76, issue 1, pp. 137-173, 2009.

[24] K. Barmpis and D. S. Kolovos, "Comparative analysis of data persistence technologies for large-scale models," In *Proceedings of the 2012 Extreme Modeling Workshop* (XM '12), D. Di Ruscio, A. Pierantonio, and J. Lara (Eds.). ACM, New York, NY, pp. 33-38, 2012.

[25] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective," In *Proceedings of the 48th Annual Southeast Regional Conference* (ACM SE '10). ACM, New York, NY, Article 42 , 6 pages, 2010.

[26] R. De Virgilio, A. Maccioni, and R. Torlone, "Converting relational to graph databases," In *First International Workshop on Graph Data Management Experiences and Systems* (GRADES '13). ACM, New York, NY, Article 1, 6 pages.