

SOLUTION GUIDE

How to Combat Money Laundering Using Graph Technology

The Neo4j AML Framework

Darryl Salas, Neo4j

Solution Guide

TABLE OF CONTENTS

Graph Technology Fights Serious Crimes	1
AML Compliance Program Design	3
Enterprise-Class AML Technology	8
Reference Implementation	12
AML Graph Algorithms	20
Graph Data Visualizations	24
Solution Framework Performance Testing	29
The Next Stage of the AML Battle	30

How to Combat Money Laundering Using Graph Technology

The Neo4j AML Framework

Darryl Salas, Neo4j

Money laundering is among the hardest activities to detect in the world of financial crime. Funds move in plain sight through standard financial instruments, transactions, intermediaries, legal entities, and institutions – avoiding detection by banks and law enforcement. The costs in regulatory fines and damaged reputation for financial institutions is all too real. Neo4j provides an advanced, extensible foundation for fighting money laundering, reducing compliance costs and protecting brand value.

Graph Technology Fights Serious Crimes

What Is Money Laundering?

Money laundering conceals the origins of illegally obtained money. Insider trading, drug trafficking, kickbacks and extortion are examples of crimes that require laundering large sums of money by principals through their agents. Agents might include individuals, corporations, financial institutions and law firms.

For example, at the height of its operations, the Medellín Cartel brought in more than US\$70 million per day – more than \$25 billion annually.

How Money Laundering Occurs

The money laundering process begins when illegal sums are placed in a bank, which in turn triggers a complex sequence of banking transfers or commercial transactions (*layering*) that return the money to the launderer in an obscure and indirect way (*integration*).

Money laundering is hard to detect because of the sophisticated layering techniques used to mask parties and layered transactions. Investigators often start with one or more bread crumbs and then follow the money trail through a series of structured transactions involving small deposits, cash and equivalents. The transactions move the funds through layers of agents, companies and financial institutions, making their owners difficult to identify.

The Panama Papers are 11.5 million leaked documents that detail financials and private information for more than 200,000 wealthy individuals, public officials and offshore agencies. The International Consortium of Investigative Journalists (ICIJ) used Neo4j to discover payment chains and shell corporations used for fraud, tax evasion, evading international sanctions, and other illegal purposes. Based on the data and analysis, the result has been hundreds of ongoing investigations, \$1.2 billion in fines and back taxes, and law enforcement has begun prosecuting.

Neo4j unlocks the wealth of insights found by pattern matching on connected people, companies, financial institutions, places and times in a financial network.

The Role of Anti-Money Laundering Regulations

Governments have tightened laws and strengthened regulations around money laundering while investing more in enforcement – producing new revenue from non-compliance fines. Since 2015, annual AML penalty revenue has risen steadily each year. According to Encompass, in 2019 alone, regulators doled out more than \$8 billion in fines. Negative consequences to banks also include damaged client relationships and reduced market competitiveness. One bank suffered a 30% decline in brand value due to negative media associated with money laundering.

Why Use Neo4j to Fight Money Laundering?

Fighting money laundering is an expensive undertaking. Large global banks process millions of transactions daily that involve tens of millions of parties. Different languages, disparate transaction formats, and unstructured data also make fighting money laundering more difficult.

Over the years, technology has automated the process, but data science techniques have not kept pace with the creative minds of money launderers. Natural language processing (NLP) has made it easier to identify potential criminal entities from unstructured media such as news feeds and documents. [Artificial intelligence and machine learning](#) (AI/ML) have helped to accelerate the evaluation of those entities. However, the high number of false positives and negatives makes investigation expensive while real money laundering goes undetected.

These techniques fail to account for relationships in the underlying networks of parties and payment chains across transactions. The fundamental problem with these technologies is that they store data as rows and columns. As a result, their backend platforms are rigid, brittle and slow.

The [Neo4j graph database](#) fills this gap and enables what was once impossible. Neo4j unlocks the wealth of insights found by pattern matching on connected people, companies, financial institutions, places and times in a financial network. Neo4j treats relationships like first-class citizens, making it possible to match [complex and changing patterns of connected money laundering data](#) in real time and at scale.

Neo4j detects previously undiscovered relationships between entities to produce a more accurate score that combines graph and text analytics.

AML Compliance Program Design

Money Laundering Detection Process

Large banks commonly process millions of transactions per day that are subject to AML compliance, each of which is associated with one or more parties.

AML challenges are daunting for a variety of reasons, including:

- Banks must process tens of millions of records in real time each day
- Banks serve individuals, corporations, other banks, and a world of educational, governmental and non-profit institutions
- Globally, transactional data is recorded in many languages, address formats, and character sets, and that data is encrypted and obfuscated to maintain its privacy
- Transactions follow various formats such as SWIFT, ACH and wire transfers
- Data fields often contain unstructured data, misspellings and inconsistencies
- Parties to transactions may be owned by other entities, which makes it even more difficult to identify everyone involved in a transaction

Processing transactions involves two steps:

- **Resolve entities** for each party by determining the degree of certainty the company knows all of the parties
- **Flag suspicious activity** by scoring suspicious transactions and parties for risk

Resolving entities and flagging can be automated using scoring models, which makes subsequent manual investigation more effective.

Regulations require businesses to have an AML program in place. One team typically performs entity resolution (ER) and flagging, and another team investigates.

Resolving Entities

Prior to using [Neo4j](#), companies relied solely on standalone text analytics and AI/ML to resolve entities. While the accuracy and efficiency of these methods has improved, they still cannot account for relationships in the underlying networks of parties and payment chains. Using these relationships makes the process more effective. Neo4j [stores the rich connections in this complex network](#) of people, places, institutions, behaviors, and times in order to improve the accuracy of the entity resolution process.

Moreover, Neo4j performs both text and graph analytics in place without having to move massive amounts of data across siloed applications. This results in faster, more efficient processing and more accurate scores.

To illustrate, two entities with similar text analytic scores may make payments to similar subsets of counterparties and may be affiliated with the same legal entities. These two entities may be associated directly or indirectly with entities that are on a watch list, have unknown ownerships, or are located in high-risk geographies. Neo4j can detect these previously undiscovered relationships between the entities to more accurately identify the same entities using a process that combines graph and text analytics.

One of the challenges of entity resolution (ER) scoring is that information about parties can initially be limited. Initial profiles of customers might be based on information reported by the customers themselves. A counterparty's initial profile might be limited to a party name, address, bank name and address, and account number. Over time, business processes must

How to Combat Money Laundering Using Graph Technology

Neo4j not only enables the native storage of the rich connections in complex network of people, places, institutions, behaviors, attributes and times – it also identifies suspicious patterns in those networks.

enhance party profiles with third-party data, information related to transactions and account activity, and details learned by investigators of flagged transactions. Millions of entities must be resolved in real time for billions of transactions daily.

For each transaction, onboarding and account creation, the entity resolution process:

- Compares parties in the transaction to already-known parties
- Adds previously unknown parties to the database
- Adds new data captured in transactions to the party master

Connecting new information to the information already known about entities gives companies better data for their flagging algorithms, thus reducing false positives and negatives.

The Neo4j scoring process indicates the degree of certainty as to whether a party is already in the database by performing a two-step process:

- Text analytics that determine text similarity
- Localized pattern matching that determine context similarity

The Benefits of Using Neo4j for AML

Neo4j not only enables the [native storage](#) of the rich connections in this complex network of people, places, institutions, behaviors, attributes and times – it goes even further to [identify suspicious patterns in those networks](#).

By leveraging Neo4j to answer these kinds of questions, compliance teams:

- Better comply with AML and other global risk and compliance (GRC) regulations
- Make more accurate predictions
- Save money on regulatory fines
- Increase sales by improving brand reputation
- Reduce costs associated with false positives and false negatives
- Meet the most stringent requirements for performance, availability, security and agility at extreme scale

Text Analytics Scoring

Using text analytics to score transactions introduces significant challenges. Transactions contain unstructured data, misspellings and inconsistencies in language, formats and character types. In other cases, transaction details are quite limited.

After transactional data is parsed, extracted and normalized, to enrich the information Neo4j applies text algorithms to score similarity among entities. For example:

- Jaro-Winkler Distance, which compares first or last names in different sources
- Sørensen Dice, which gauges similarity between two samples
- Levenshtein Distance, which compares strings for misspellings, punctuation differences and other subtleties

Neo4j uses the resulting enhanced, scored party information as the basis for pattern matching.

In the popular television series *Breaking Bad*, Walter White, a brilliant research chemist whose alias was “Heisenberg,” made between \$78 million to \$96 million in profit in a single year by “cooking” methamphetamines. Walter and his wife Skyler laundered the illicit money through their car wash business.

Localized Pattern-Match Scoring

Localized pattern matching looks for attributes having direct or indirect relationships to other anchor attributes. Common pattern-match scores can include shared-attribute anchors such as address, phone number, email address or domain.

Localized pattern-match scoring algorithms also consider the number of paths from the anchor point to resolved entity as well as the number of hops. When scoring paths, techniques such as linear, pareto, logarithmic and weighted scoring are used for aggregation. Other techniques also boost and exclude weights based on relationship types.

Graph Algorithms

Maintaining an AML database presents some unique challenges. Over time, a database grows to contain millions of parties and billions of transactions – an unwieldy size for non-graph technologies that makes analysis inefficient. It is also all too common for an AML database to contain inaccurate, incomplete, old or missing information.

To handle these unique AML challenges, Neo4j uses [graph algorithms](#) to:

- Fill in gaps of missing and inaccurate data
- Identify subgraphs that are relevant for scoring the current transaction
- Combine normalized string-similarity and pattern-match scores to produce an overall similarity score
- Weight relationships among similar entities and apply community-detection techniques

Graph algorithms run in near real time because they are applied to a subset of the overall graph. Later, to enhance the value of information in the AML database, Neo4j can run graph algorithms over the entire graph to assess historical similarities scores between entities as transaction and party histories grow and enhance the richness of the data.

Flagging Suspicious Activity

Once entities have been resolved, the AML process flags suspicious entities and transactions. Transaction analysis follows money trails by tracing transactions across counterparties, who subsequently may make payments to others.

To keep pace with money launderers, investigators must constantly improve their systems with better data, models and judgments. Neo4j's AML Framework gives investigators the power they need to keep pace because it:

- Reduces the volume of false positives and false negatives
- Automates and instills rigor to the ER-scoring process
- Acquires connected data that enriches the master party dataset
- Cultivates domain expertise to model relationships to help identify payment chains

How to Combat Money Laundering Using Graph Technology

In the TV series *Ozark*, Marty Byrde, a forensic accountant, and his wife, Wendy, a politician, laundered money for a Mexican drug cartel. The Byrdes and a local crime lord purchased a casino, and the Byrdes laundered the cartel's illicit money through its operations. "Smurfs" were enlisted to launder some of the money by intentionally losing at the casino. Some of the smurfs were given illicit money up front, and more trusted smurfs were paid back for their losses.

The AML process scores suspicious parties and transactions using localized pattern-match scoring algorithms that look at structural and behavioral patterns that include:

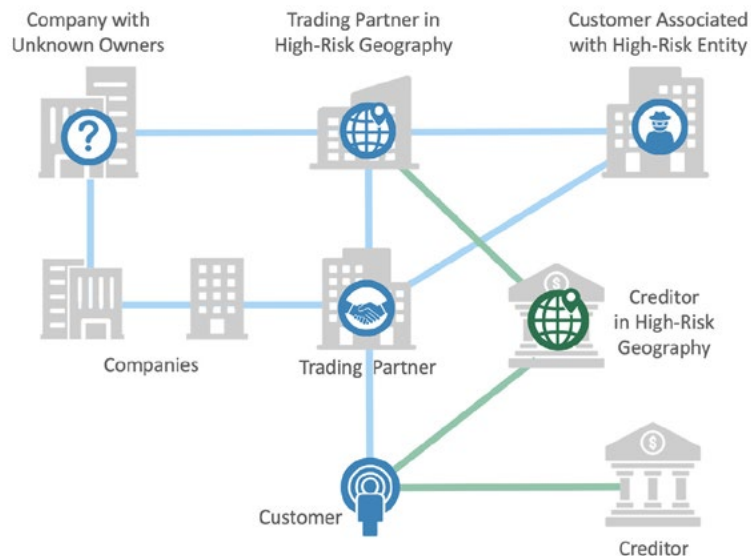
- *Guilty by Association* scores
- Suspicious structure scores such as entities with unknown ownership, cash-incentive businesses and high-risk geographies
- Suspicious behavior scores involving transaction structuring and payment-chain layering such as rapid movements, accumulation and concentration of funds

Guilty by Association Scores

It is common to judge people by those with whom they associate, and the same holds true in anti-money laundering analysis. *Guilty by Association* scores are based on the quantity, quality and distance of a party's relationships with suspicious entities. In graph terms, algorithms create scores that are based on paths and hops from start to end points.

Some Guilty by Association scores include customers associated with the following watch lists:

- Regulatory and law enforcement actions
- Negative news coverage
- Global and narrative sanction lists
- Politically Exposed Persons (PEPs)
- High-risk individuals or legal entities
- Legal entities with unknown ownership
- Counter-parties in high-risk geographies
- Banks in high-risk geographies



Guilty by Association algorithms score parties based on the quantity, quality and distance of related parties' relationships with suspicious entities.

Finding Suspicious Activity Using Graph Algorithms

As with the ER process, the flagging process applies several scoring algorithms before applying graph algorithms.

After Guilty by Association, Layering, Undervalued Invoices and other scenario analyses are executed, run these graph algorithms:

- [Weakly Connected Components](#) algorithm to identify the network of account holders in a payment chain that high-risk accounts use to conduct money laundering
- [Centrality algorithm](#) to determine the role and relevance of the accounts in the network
- Clustering algorithms such as [Louvain](#) or [Strongly Connected Components](#) to identify subnetworks and payment chains within the money laundering network

Investigating Suspicious Transactions

Investigating flagged transactions by centralized teams is expensive. An investigation can lead to their filing a Suspicious Activity Report (SAR), which law enforcement agencies use to initiate enforcement actions. Besides reducing false positives and negatives, the Neo4j AML framework provides more relevant facts and circumstances surrounding the transactions that were flagged as suspicious in the SAR, making the investigation process easier and more thorough.

One of the key goals of flagging is to produce a highly automated playbook of templated, parameterized questions that can be used to flag suspicious entities, transactions and activities.

Parameterized playbook questions written by IT staff help your analysts explore various AML behaviors without having to write code. Just like in sports, playbooks aren't perfect, and being adept at calling a new play that changes the current game plan is crucial.

Investigators need to understand the facts and circumstances surrounding a flagged entity or transaction. They must be able to filter, expand paths and ask ad-hoc questions at interactive speeds to explore the full context of the structure, behavior and time surrounding the flagged items.

The sheer volume of data makes AML information impossible to comprehend without a self-service data visualization tool that supports real-time, natural language and ad hoc analysis. [Neo4j Bloom](#) and the graph algorithms in the [Neo4j Graph Data Science Library](#) enable analysts to visualize suspicious clusters and entities at a high level.

Neo4j's end-to-end graph processing and retrieval is blazingly fast and accurate, even at extreme scale.

Enterprise-Class AML Technology

Neo4j: The Only Enterprise Graph Database

Traditional AML technologies store data in tables of rows and columns, or in documents. Pattern matching requires the data to be retrieved, parsed and JOINed. To attempt all this, they rely heavily on indexes. While fast performance might be possible for simple pattern matching, it is impossible with complex patterns or large volumes of data.

In contrast to these other technologies, Neo4j stores data natively as a graph, an approach known as [index-free adjacency](#). Each node directly references its adjacent nodes, acting as a micro-index for all nearby nodes. As a result, Neo4j traverses paths with blazing speed and does not rely on indexes at all for the traversals, but rather, simply chase pointers in memory.

Because Neo4j natively processes data as a graph, nothing gets lost in translation while interrogating the graph. Neo4j's end-to-end processing and retrieval is blazingly fast and accurate, even at extreme scale.

Neo4j is ideal for addressing the stringent requirements of anti-money laundering because it delivers:

- Blazingly fast query performance at extreme scale for detecting suspicious patterns and behaviors in real time
- In-memory analytics pipelines to provide scored results in real time
- Highly optimized data science library highlights communities and central entities at a high level
- Causal Clustering to ensure an always-available database and disaster recovery
- Data ingestion and management to process billions of records a day
- Trusted integrity and dependability through its [ACID](#) graph transactions
- [Enterprise-grade security](#) that is schema-based for fine-grained access, traversals and reads
- Extreme agility, extensibility and customizability from its whiteboard data models and [Cypher query language](#)

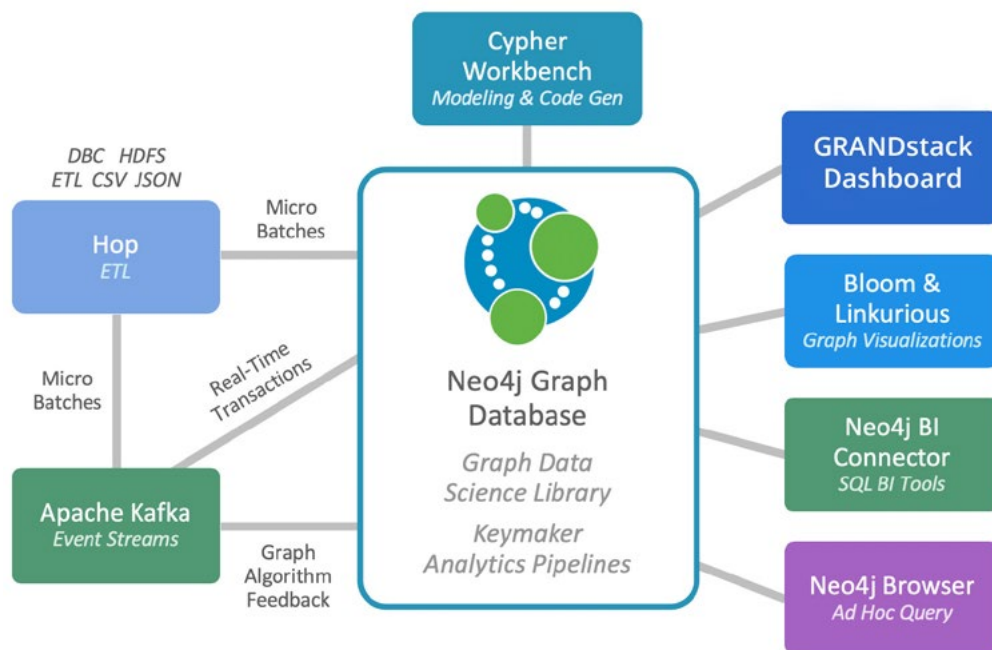
Neo4j uses a native-graph language called Cypher. It is a descriptive yet concise programming language that looks like plain English and requires only a few lines of code where other languages require hundreds of lines. Industry experience shows that an application written with 100,000 lines of code takes 100 times longer to develop and maintain than the same application with 1,000 lines of code written in a more efficient language.

How to Combat Money Laundering Using Graph Technology

The Neo4j AML Framework includes the Neo4j Database, development platform and a pre-built application framework for streamlining AML development projects.

Neo4j: The Only Enterprise Graph Platform

Neo4j provides a flexible and [robust end-to-end graph platform](#) that extends from AML to other global risk and compliance (GRC) areas that use a lot of the same connected data. Neo4j further accelerates time to value by making it easy to integrate with its ecosystem.



The components of the Neo4j Graph Platform makes it an ideal choice for the rigorous demands of AML and other GRC use cases.

Data Modeling Tool

Cypher Workbench

Model the graph and synchronize Neo4j with the model. Also generate Cypher code and scripts.

Input APIs

Hop

Move source data the last mile into Neo4j via this robust, model-driven, open source ETL and orchestration tool. Hop connects directly to data sources via connectors or can pull files from a folder. Performs data validation, look-ups, normalization, scheduling, error handling, parallelization, logging/lineage, auditing and other manipulations.

Native Neo4j drivers

Feed data from applications written in modern languages and scripts such as [Python](#), [Java](#), [JavaScript](#), [.NET](#) and [Go](#). Community-driven drivers exist for other development languages.

Output APIs

Neo4j BI Connector

Deliver direct access to Neo4j graph data from business intelligence (BI) tools such as Tableau, Looker, TIBCO Spotfire Server and Microstrategy using the [Neo4j BI Connector](#).

GRANDstack: GraphQL, React, Apollo, Neo4j Database

Build custom dashboards via the [GRANDstack](#). An example GraphQL API and React dashboard is shown below

Bloom Self-Service Graph DataVisualization

Explore and discover in with no coding required using [Neo4j Bloom](#).

Neo4j Browser

Construct Cypher queries and visualize results in the [Neo4j Browser](#).

Native Neo4j Drivers

Feed downstream applications using [language drivers for Neo4j](#) Database.

Analytics

Neo4j Graph Data Science Library (GDSL)

Deliver optimized graph algorithm performance at scale via [GDSL](#).

Keymaker

Build queries for each step in an AML scoring process and pipeline the queries in-memory.

Packaging

Package Neo4j frameworks for Windows, macOS and most common Linux distributions

Neo4j Advanced Money Queries

The table below explains the basic types of money queries used in the Neo4j AML Framework and how they can be used to build sophisticated AML detection algorithms.

Types of Money Queries

Query Type	Description	Real-Time or Batch
Localized Pattern Match	Complex pattern or values matching. Returns set of matching relationships and entities based on simple scoring.	Real-time response with data updated daily or continuously.
Localized Pattern-Match Scoring	Complex pattern match, then score the paths and hops. Returns a set of scores or a sub-graph.	Real-time response with data updated daily or continuously.
Global Pattern Match/Graph Algorithms	Pattern matching and scoring across entire dataset or subgraph. Returns set of scores or a subgraph.	Sub-second or second response time, up to minutes or hours, depending on the volume of graph data. Data updated less frequently.

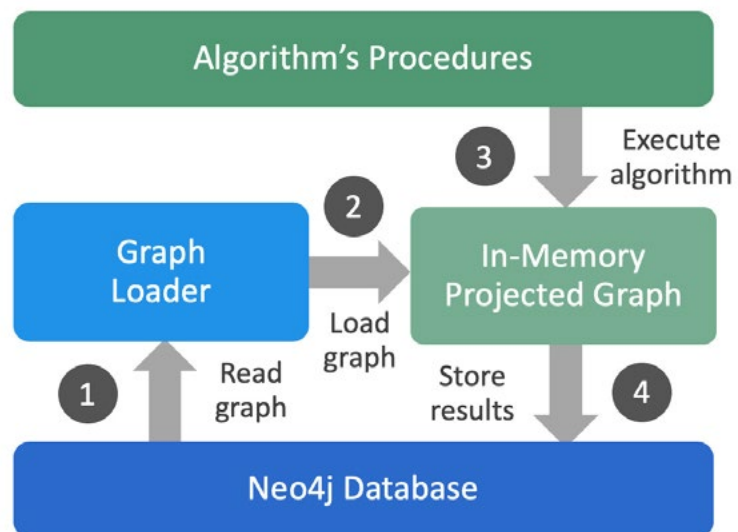
Pattern Matching with Graph Algorithms

Neo4j is the only native graph database that has a robust graph data science library which:

- Frees analysts from moving large volumes of data to integrate scoring algorithms
- Leverages relationships that were previously untapped by AI/ML pipelines

Neo4j has a dedicated team of engineers developing and maintaining a comprehensive, high-performance Graph Data Science Library (GDSL). All of Neo4j's graph algorithms – with the exception of similarity algorithms – directly leverage relationships in the graph. A majority are intended for monopartite graphs (those with a single-node label).

Similarity algorithms, such as [Jaccard](#) and [Cosine](#), use relationships to preprocess data. These similarity and graph algorithms traverse the graph, in some cases multiple times. For this reason they all execute in an in-memory projected graph and their resulting computations are either streamed or stored back into the Neo4j Database. This makes the Neo4j Graph Platform optimal for executing, storing, retrieving and processing these algorithms.



All Neo4j graph algorithms execute in an in-memory projected graph and their results are then streamed or stored back into the Neo4j Database.

Building a simple solution with a handful of carefully chosen Money Queries allows a business to quickly reap the benefits of identifying money laundering.

Reference Implementation

This section describes how to perform sprints when developing Neo4j applications. A sprint starts with identifying a very small number of “money queries.” The sprint continues with creating a graph data model, and then building out a full-stack solution (from ingestion to visualization) for your first set of money queries.

Building a simple solution with a handful of carefully chosen money queries allows a business to quickly reap the benefits of identifying money laundering.

The following queries can be used as building blocks or combined with other queries to arrive at a money query.

Localized Pattern Match: Suspicious Behavior Money Queries

Small Deposits: Concentration

- Cash transactions are deposited to an account, transferred to a central account, then wired to a bank outside of the U.S.
- The goal is to identify parties and largest aggregate amounts.
- Money queries show the accounts involved and the largest aggregate amounts (n hops, aggregating only when pattern matches).
- Suspicious accounts receive a high number of incoming deposits and then send a few large transactions to one or more high-risk parties.

Small Deposits: Accumulation

- This pattern is characterized by consecutive days of deposits with minimal withdrawals in the same period.
- Money queries identify parties and accounts involved and largest aggregate amounts (n hops, aggregating only when pattern matches).

Small Deposits: Velocity

- Customer makes several cash deposits just under \$10,000 over an x -day period.
- Customer receives a large wire followed by withdrawal of most of it as cash via multiple ATM transaction within a short period of time.
- This pattern is signalled by a major behavior change. For example, a business customer whose cash deposit activity has gone from \$50,000 a week to \$250,000 a week over the course of a month.

Layering

- Party A sends to Party B and then B sends to Party C, where the transaction is greater than or equal to the amount between A and B and within $Y\%$ from B to C.
- Money queries in this scenario look for which nodes in the transactions have the highest incoming amount and few or no outgoing transactions.
- The pattern is revealed in a large aggregated set of deposits by a customer followed by a large ACH transaction or a transfer to another account such as a mortgage.

Localized Pattern Match: Suspicious Structure Money Queries

Structural: Entity Resolution

Shared attributes can be used for entity resolution

Structural: Payment Chain

Payment chain between two suspicious parties

Localized Pattern Match Scoring Money Queries

See Keymaker pipelines in the Sample Localized Pattern-Match Scoring section below.

Graph Algorithm Money Queries

Centrality

PageRank, Closeness, Degree, etc.

Closeness scores detect central players, liaisons (betweenness) and the most relevant parties (PageRank) in a path between a customer and a high-risk endpoint.

Community Detection

Louvain Modularity, Label Propagation, Strongly and Weakly Connected Components, etc.

Label Propagation detects common entities and strongly connected components in high-risk rings. These are all based on relationships in the graph.

Link Prediction

Common Neighbors, Preferential Attachment, Adamic Adar, etc.

Link prediction algorithms based on the money trail identify hidden **COLLABORATOR** relationships. These new relationships further inform the analysis of small deposit accounts that involve layering, velocity, concentration, etc.

Similarity

Jaccard, Cosine, Overlap, etc.

Similarity algorithms are used for entity resolution. Also, if there is a path between a customer and a high-risk end point, similarity algorithms indicate how similar each path is to other paths from that specific customer to those high-risk end points. A company fighting money laundering could then create a subgraph of paths (e.g., paths A, B and C) and have weighted relationships representing similarities among the paths.

Pathfinding and Search

Breadth-First Search, All Pairs Shortest Path, etc.

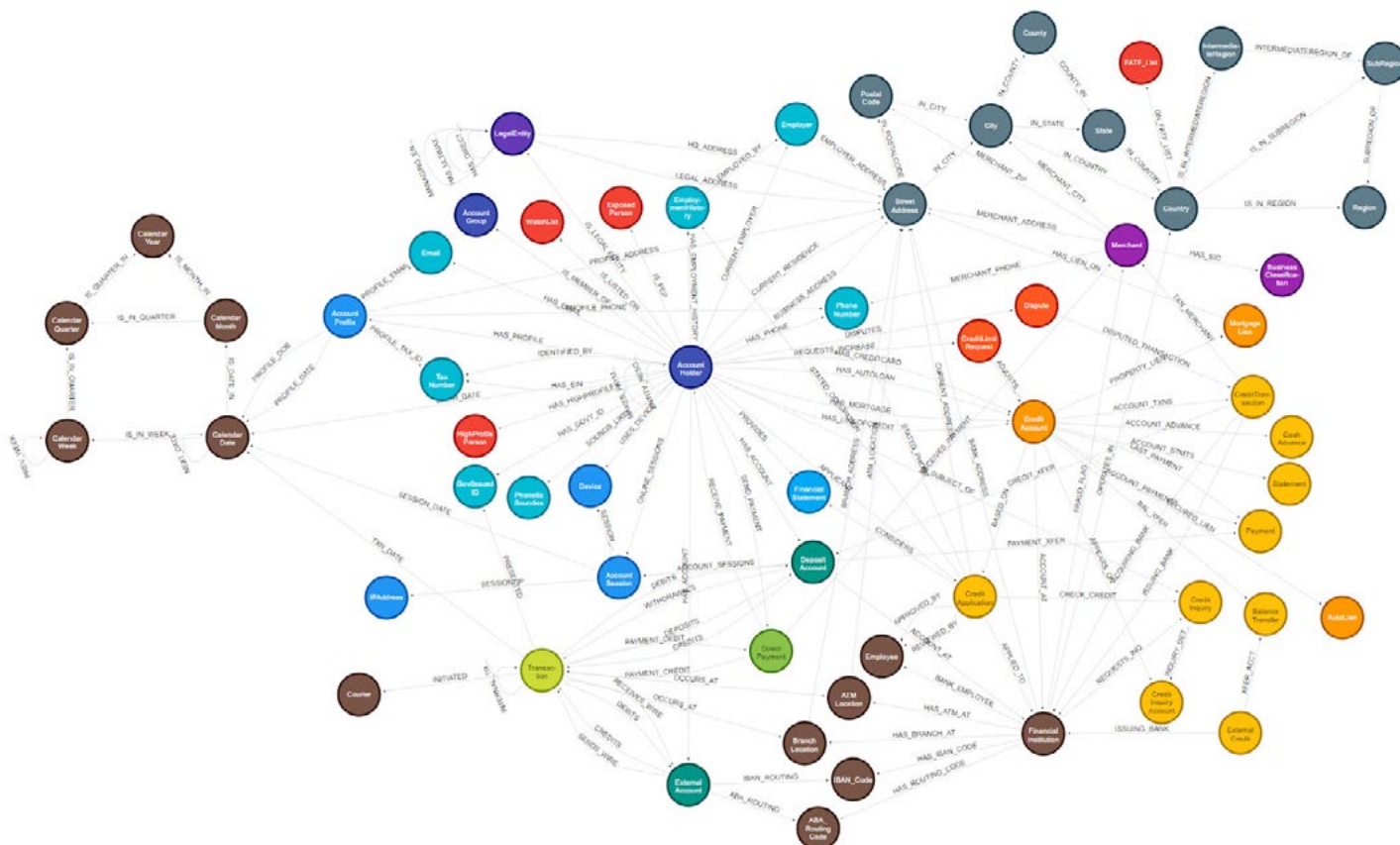
These algorithms identify payment chains and third parties layered between customers or transactions and other end points. They are also used as a fundamental step in Weakly and Strongly Connected Components, Closeness and other [graph algorithms](#).

How to Combat Money Laundering Using Graph Technology

Reference Graph Data Model

Neo4j's AML [graph data model](#) pictured below is a whiteboard-style reference model for the queries described in this document. The graph model used in the framework demonstrates best practices when working with Neo4j to support pattern matching for analysis. Only twenty indexes were required to deliver millisecond response time at scale.

In contrast to Neo4j's graph model, a relational database approach would have an entity-relationship diagram with over 150 tables and 300 to 400 indexes (not including primary keys) for the same questions and dataset. The relational model would also require hundreds of indexes to deliver much slower query-response times measured in minutes or hours.



Neo4j's whiteboard-style graph data model for anti-money laundering.

Structural Elements

- Parties: Individual and Legal Entity
- Accounts
- Transactions
- Addresses
- PaymentMethod
- Invoices
- WatchList
- Address, Phone
- Other PII details

Behavioral Elements

- Events that indicate change in behavior
- Sudden increases in deposits, etc. over a short period of time
- Transactions
- TransactionSize (bucketed)
- Payments

Temporal Elements

- TimeTree: Facilitates point-in-time queries and versioning

Relationship Types

~120 types in the model

- OPERATES_IN
- IN_COUNTRY
- SENDS_WIRE
- RECEIVES_WIRE
- ACCOUNT_AT
- ISSUING_BANK
- DEPOSITS
- WITHDRAWALS
- DEBITS

How to Combat Money Laundering Using Graph Technology

Nodes Versus Properties

The reference model implements certain structural attributes as nodes instead of properties on a customer node. These nodes include shared attributes such as address, phone, email address and SSN, as well as behavioral attributes such as transactions, withdrawals and other events that have time stamps.

Temporality

A time tree helps answer questions about changed behavior or other events over different points in time. This approach allows for fast access and uses date parts such as quarters, weeks and months, without additional properties and indexes. The time tree also supports pattern matching on date values (e.g., accounts opened within 30 days of a suspicious transaction).

Relationship Types

The Neo4j [labeled property graph model](#) unlocks the insights gained from leveraging a rich set of relationship types that provide context in the graph. The model demonstrates several relationship types that connect transactions, such as deposits, withdrawals, debits, etc.

Sample Localized & Globalized Queries

Sample Localized Pattern Match

The following are some sample queries written in [Cypher](#) to query the Neo4j graph database. In contrast, writing equivalent queries in SQL would require up to 10 to 100 times more lines of code and consequently take 10 to 100 times longer to develop and maintain.

Small Deposits: Concentration

```
// *****
// * AML Query for Cash transactions (Deposits) to separate account,
// * Transfer of combination of Cash Transactions to central (Yth) account,
// * Followed by wire to bank outside US
// *****
// Step 1 - Find anyone who made wire transfer to high risk geo (Country's high-risk
score higher than 0.6)
WITH date('2019-06-30') as endDate
WITH endDate, endDate-duration({days:90}) as startDate
MATCH p=(ctr:DepositAccount)-[:WITHDRAWALS]-(wt:Wire:Transaction)-[:RECEIVES_WIRE]-(
ext:ExternalAccount)
    -[:IBAN_ROUTING]->(iban:IBAN_Code)
    <-[:HAS_IBAN_CODE]-(fi:FinancialInstitution)-[:OPERATES_IN]->(cty:Country)
WHERE wt.transactionDate >= startDate
    AND wt.transactionDate <= endDate
    AND cty.high-riskScore > 0.6

// Step 2 - find the accounts that had cash deposits that transferred to the account
that did the wire transfer
WITH ctr, wt, ext, iban, fi, cty
MATCH g=(cu:AccountHolder)-[:HAS_ACCOUNT]->(src:DepositAccount)-[:WITHDRAWALS]-
>(srcTxn:ACH:Transaction)
    -[:INTERNAL_XFER]->(ctrTxn:ACH:Transaction)-[:DEPOSITS]->(ctr)-[:HAS_ACCOUNT]-(
cu2),
    (ctr)-[:WITHDRAWALS]-(wt)-[:RECEIVES_WIRE]-(ext)-[:IBAN_ROUTING]->(iban)
    <-[:HAS_IBAN_CODE]-(fi)-[:OPERATES_IN]->(cty),
    (extName:External:AccountHolder)-[:HAS_ACCOUNT]->(ext)
WHERE srcTxn.transactionDate <= wt.transactionDate
    AND duration.inDays(srcTxn.transactionDate, wt.transactionDate).days<30
RETURN cu.accountName, src.accountNumber, srcTxn.amount, srcTxn.transactionDate,
    ctr.accountNumber, cu2.accountName, wt.amount, wt.transactionDate,
    ext.accountNumber, extName.accountName, fi.bankName, cty.countryName

ORDER BY ctr.accountNumber, src.accountNumber
```

How to Combat Money Laundering Using Graph Technology

Small Deposits: Accumulation

```
//*****
// * AML Query for large deposits/low withdrawals (accumulators)
//*****

// Find anyone who deposited more than 10K in the last 35 days

WITH date('2019-08-01') as curDate
WITH curDate, curDate-duration({days:35}) as curDate35
MATCH (cu:AccountHolder)-[:HAS_ACCOUNT]->(da:DepositAccount)
    <-[:DEPOSITS]->(txn:Cash:Transaction)
WHERE txn.transactionDate >= curDate35
    AND txn.transactionDate <= curDate
    AND ((cu.FinCEN_110_status IS NULL) OR (cu.FinCEN_110_status="ineligible"))
WITH da.accountID as accountID,
    min(txn.transactionDate) as StartDate,
    max(txn.transactionDate) as EndDate,
    duration.inDays(min(txn.transactionDate),
        max(txn.transactionDate)).days as NumDays,
    count(txn.amount) as NumDeposits,
    sum(txn.amount) as TotalDeposits
WHERE TotalDeposits > 10000
WITH collect({accountID: accountID, startDate: StartDate, endDate: EndDate, numDays:
    NumDays, numDeposits: NumDeposits, totalDeposits: TotalDeposits}) as possAggregators

//...now find all the rapid withdrawals
UNWIND possAggregators as curAgg
MATCH (da:DepositAccount)-[:WITHDRAWALS]->(txn)
WHERE da.accountID=curAgg.accountID
    AND txn.transactionDate >= curAgg.startDate
    AND txn.transactionDate <= curAgg.endDate+duration({days:10})
WITH da.accountID as AccountID,
    curAgg.startDate as StartDate,
    max(txn.transactionDate) as EndDate,
    duration.inDays(curAgg.startDate,max(txn.transactionDate)).days as NumDays,
    curAgg.numDeposits as NumDeposits,
    curAgg.totalDeposits as TotalDeposits,
    count(txn.amount) as NumWithdrawals,
    sum(txn.amount) as TotalWithdrawals,
    round((abs(sum(txn.amount))/curAgg.totalDeposits)*100) as pctWithdrawal
WHERE pctWithdrawal < 10

// Now add in the customer info
MATCH (cu:AccountHolder)-[:HAS_ACCOUNT]->(da:DepositAccount)
WHERE da.accountID=AccountID
RETURN cu.accountHolderID, cu.accountName, labels(cu)[0], da.accountNumber, StartDate,
    EndDate, NumDeposits, TotalDeposits, NumWithdrawals, TotalWithdrawals,
    pctWithdrawal

ORDER BY TotalDeposits DESC, pctWithdrawal DESC
```

How to Combat Money Laundering Using Graph Technology

Small Deposits: Velocity

```
//*****
// * AML Query for rapid cash deposits/withdrawals
//*****
// Step 1 - Find anyone who deposited more than 10K in the last 35 days

WITH date('2019-08-01') as curDate
WITH curDate, curDate-duration({days:35}) as curDate35
MATCH (cu:Customer:AccountHolder)-[:HAS_ACCOUNT]->(da:DepositAccount)-[:DEPOSITS]-(txn:Cash:Transaction)
WHERE txn.transactionDate >= curDate35
      AND txn.transactionDate <= curDate
WITH da.accountID as accountID,
      min(txn.transactionDate) as StartDate,
      max(txn.transactionDate) as EndDate,
      duration.inDays(min(txn.transactionDate),max(txn.transactionDate)).days as NumDays,
      count(txn.amount) as NumDeposits,
      sum(txn.amount) as TotalDeposits
WHERE TotalDeposits > 8500
WITH collect({accountID: accountID, startDate: StartDate, endDate: EndDate, numDays: NumDays,
              numDeposits: NumDeposits, totalDeposits: TotalDeposits}) as possAggregators

// Step 2...now find all the rapid withdrawals of high percentage of deposits
UNWIND possAggregators as curAgg
MATCH (da:DepositAccount)-[:WITHDRAWALS]-(txn:Cash:Transaction)
WHERE da.accountID=curAgg.accountID
      AND txn.transactionDate >= curAgg.startDate
      AND txn.transactionDate <= curAgg.endDate+duration({days:10})
WITH da.accountID as AccountID,
      curAgg.startDate as StartDate,
      max(txn.transactionDate) as EndDate,
      duration.inDays(curAgg.startDate,max(txn.transactionDate)).days as NumDays,
      curAgg.numDeposits as NumDeposits,
      curAgg.totalDeposits as TotalDeposits
      count(txn.amount) as NumWithdrawals,
      sum(txn.amount) as TotalWithdrawals,
      abs(sum(txn.amount))/curAgg.totalDeposits as pctWithdrawal
WHERE pctWithdrawal > 0.75

// Step 3 - Now add in the customer info
MATCH (cu:Customer:AccountHolder)-[:HAS_ACCOUNT]->(da:DepositAccount)
WHERE da.accountID=AccountID
RETURN cu.accountHolderID, cu.lastName, cu.firstName, da.accountNumber, StartDate, EndDate,
       NumDeposits, TotalDeposits, NumWithdrawals, TotalWithdrawals, pctWithdrawal
ORDER BY pctWithdrawal DESC, TotalDeposits DESC LIMIT 100
```

Layering: Payment Chain

The following is an example query of a payment chain branching out from a single suspicious entity as well as an example of a payment chain between two suspicious actors in a possible payment chain.

```
//*****
// * AML Query for Fraudulent Accounts - Payment Chain
//*****
MATCH p=(ah1:BusinessCustomer:AccountHolder)-[:MAKES_PAYMENTS_TO|ENTITY_RESOLUTION*1..4]->(ah2)
WHERE ah1.accountName="Lang and Sons"
      AND id(ah1)<>id(ah2)
RETURN p
ORDER BY length(p)
LIMIT 500

MATCH p=(ah1:BusinessCustomer:AccountHolder)-[:MAKES_PAYMENTS_TO|ENTITY_RESOLUTION*1..4]->
      (ah2:BusinessCustomer:AccountHolder)
WHERE ah1.accountName="Lang and Sons"
      AND ah2.accountName="Klein, Johnston and Glover"
RETURN p LIMIT 500
ORDER BY length(p)
```

How to Combat Money Laundering Using Graph Technology

Structuring Cash & Equivalents

The code example below looks for layering through one or more intermediaries.

```
//*****
// * AML Query for passthru payments (A)->(B)->(C)
// * where $ amount >= $X between A and B and within y % in Between B to C.
//*****

WITH date('2019-08-01') as curDate
WITH curDate, curDate-duration({days:35}) as curDate35
MATCH p=(cu1)-[:HAS_ACCOUNT]->(da1:DepositAccount)-[:WITHDRAWALS]->(txn1)
      <-[:DEBITS]->(ext:ExternalAccount)-[:CREDITS]->(txn2)
      <-[:DEPOSITS]->(da2:DepositAccount)-[:HAS_ACCOUNT]->(cu2)
WHERE id(da1)<>id(da2)
      AND txn1.transactionDate <= curDate
      AND txn1.transactionDate >= curDate35
      AND txn2.transactionDate>=txn1.transactionDate
      AND duration.inDays(txn1.transactionDate,txn2.transactionDate).days < 7
      AND txn2.amount > abs(txn1.amount) * 0.85
      AND txn2.amount < abs(txn1.amount)
      AND abs(txn1.amount) > 1000.0
WITH cu1.customerID as sendAccountID, cu1.accountName as sendAccountName,
      sum(abs(txn1.amount)) as amtSent, count(txn1) as numXmit,
      ext.accountID as nextAccountID, ext.accountName as nextAccountName,
      sum(abs(txn2.amount)) as amtRcv, count(txn2) as numRcv,
      cu2.customerID as rcvAccountID, cu2.accountName as rcvAccountName
WITH sum(amtSent) as totalSent, sum(numXmit) as totalTxns,
      sum(amtRcv) as totalRcv, sum(numRcv) as numRcv,
      rcvAccountID, rcvAccountName
WHERE numRcv > 2 AND totalRcv > 6500
RETURN totalSent, totalTxns, totalRcv, numRcv, rcvAccountID, rcvAccountName
ORDER BY totalSent DESC
```

The following query examines deposits or withdrawals that exceed a threshold followed by rapid withdrawal.

```
//*****
// * Potential Structuring in Cash and Equivalents (by grouping actors relationship)
// * Looks across the Lookback Period for episodes where multiple daily
// * transaction amounts aggregate above a desired threshold
//*****

WITH date('2019-08-01') as curDate, 150000 as curThreshold
WITH curThreshold, curDate, curDate-duration({days:15}) as curDate35
MATCH (txn:Transaction)
WHERE txn.transactionDate <= curDate
      AND txn.transactionDate >= curDate35
WITH txn.accountID as curAccountID, txn.transactionDate as transactionDate,
      curThreshold, sum((CASE WHEN txn.amount<0 THEN txn.amount ELSE 0 END)) as
      sumWithdrawals, sum((CASE WHEN txn.amount>0 THEN txn.amount ELSE 0 END)) as
      sumDeposits
WHERE sumWithdrawals < -1*curThreshold OR sumDeposits > curThreshold
      OR (sumDeposits>7500 AND abs(sumWithdrawals)>sumDeposits*0.75)
WITH curAccountID, transactionDate, sumWithdrawals, sumDeposits, curThreshold
MATCH (ah:AccountHolder)-[:Has_ACCOUNT]->(acct:DepositAccount)
WHERE acct.accountID=curAccountID

      // ignore vetted customers
      AND ((ah.FinCEN_110_status IS NULL) OR ah.FinCEN_110_status="ineligible"))
RETURN ah.accountName, acct.accountNumber, transactionDate, sumWithdrawals,
sumDeposits
LIMIT 250
```

How to Combat Money Laundering Using Graph Technology

Keymaker is a development and execution framework for building pipelines that score high-risk entities and payment chains in real time.

Sample Localized Pattern-Match Scoring

Neo4j includes an analytics pipeline development and execution framework called **Keymaker** that uses these queries to score high-risk legal entities, depending on the context of payment chains, and relationships with other people, places and things. Keymaker pipelines execute analytic steps in-memory for nearly instant insights.

Keymaker allows data scientists, developers and analysts to pass code into the steps as parameters. Parameterized plug-and-play steps can be turned on and off, and can be shared among your AML team. Keymaker comes with a development console and an administration console. The results of the pipelines can be visualized as shown below.

Keymaker Pipeline Examples



Keymaker analytics pipelines chain together a series of Cypher queries to perform complex pattern matching in real time

Graph algorithms dramatically increase the accuracy of entity resolution.

AML Graph Algorithms

Graph Algorithms to Resolve Entities

Graph algorithms dramatically increase the accuracy of entity resolution by providing information regarding relationships at the network level rather than at the individual entity level. Graph algorithms are applied globally across graphs or sub-graphs and leverage the information inherent in relationships. Those scores are then used to support tailored pattern matching that in turn feeds into machine learning pipelines for more accurate predictions.

The entity resolution process follows these steps:

- A linear combination of string and shared-attributes similarity scores are used to create pairwise-weighted relationships across a graph of similar entities such as individuals and corporations
- The Weakly Connected Components algorithm segments the graph into clusters
- The Label Propagation graph algorithm determines within each cluster which entity best represents the other entities in the cluster

The first step of creating pairwise-weighted relationships uses Cypher queries to create a weighted **ENTITY_RESOLUTION** relationship. The Cypher queries utilize stored procedures for Jaro-Wrinkler, Levenshtein and Sorensen-Dice algorithms along with graph attributes such as current residence distance.

The following script creates an in-memory graph named **Entity** of the account holders nodes that uses the **ENTITY_RESOLUTION** as relationships among these nodes and the **confidenceFactor** property as weights for these relationships.

```
CALL gds.graph.create('Entity','AccountHolder','ENTITY_RESOLUTION',
  {nodeProperties:['accountHolderID'],
   relationshipProperties:['confidenceFactor']});
```

The Label Propagation algorithm is executed on this in-memory graph – using the **accountHolderID** as seeds – to identify clusters of account holders that potentially represent the same entity. Since the ID of account holders are seeds, the **clusterID** represents the most likely identity represented by the accounts in the cluster.

```
CALL gds.labelPropagation.stream('Entity',
  {relationshipWeightProperty:'confidenceFactor',
   seedProperty:'accountHolderID'})
YIELD nodeId,communityId
WITH gds.util.asNode(nodeId) AS account,communityId AS clusterId
WITH clusterId,collect(account.accountHolderID) AS accounts
WITH clusterId,accounts,size(accounts) AS clusterSize
WHERE clusterSize>1
RETURN clusterId,clusterSize,accounts
ORDER BY clusterSize DESC;
```

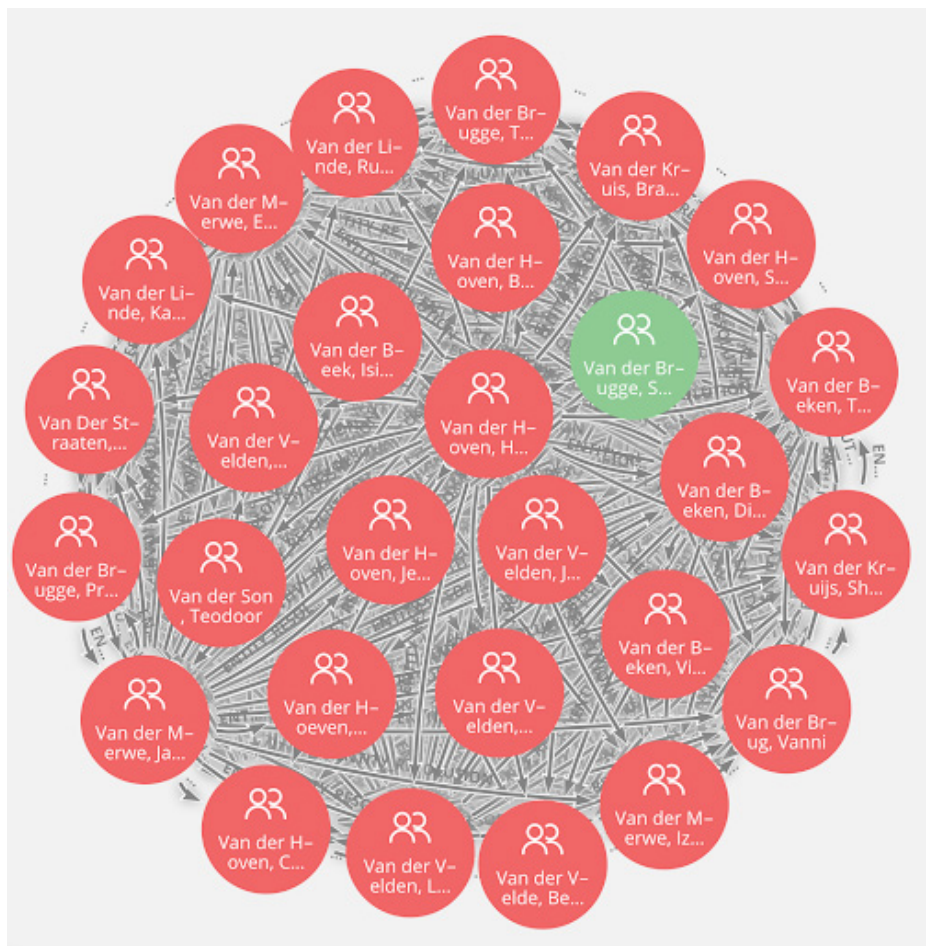
How to Combat Money Laundering Using Graph Technology

```
$ call gds.labelPropagation.stream('Entity', {relationshipWeightProperty:'confidenceFactor',seedProper...
```

	clusterId	clusterSize	accounts
Text	44685	28	[9095, 49262, 39216, 39359, 29383, 39516, 9696, 29921, 20342, 40491, 20700, 32453, 42907, 32964, 22015, 33763, 34072, 34117, 44685, 34619, 44919, 35591, 15784, 15901, 6290, 26350, 36631, 47885]
Code	50620	25	[50620, 51408, 52775, 52813, 52921, 53840, 55008, 55203, 55607, 55760, 55875, 56246, 56571, 56748, 56800, 57237, 57619, 58044, 58589, 58709, 59164, 59281, 59432, 59641, 59743]
	50064	24	[50064, 50721, 50888, 51109, 52529, 52668, 53086, 53422, 54093, 54838, 55093, 55661, 56224, 56316, 56769, 57097, 57194, 57699, 58149, 58730, 58878, 58979, 59195, 59962]
	48947	22	[38899, 48947, 40785, 18819, 28843, 36900, 48948, 28844, 40786, 18820, 38901, 48949, 28846, 40539, 39881, 19925, 11386, 43509, 14594, 46068, 48671, 37216]
	50122	22	[50122, 51241, 52092, 53267, 53421, 53502, 53617, 54235, 54297, 55168, 55650, 55862, 56153, 56270, 57248, 57358, 57790, 58232, 58831, 58846, 58993, 59881]
	50455	22	[50247, 50455, 50681, 51197, 51262, 51651, 52907, 53015, 53749, 54463, 54725, 55257, 55465, 55604, 55863, 55952, 56735, 58850, 58950, 59006, ...]

Started streaming 11430 records after 486 ms and completed after 495 ms, displaying first 1000 rows.

In the screenshot above the first cluster with ID 44685 is composed of 28 accounts that all might be representing the account holder with ID 44685.



A cluster of accounts in red are likely to represent the same entity shown in green

How to Combat Money Laundering Using Graph Technology

Graph Algorithms to Identify Payment Chains

To identify money laundering networks potentially being used by high-risk accounts, create a property named **highRisk**. This property identifies account holders in a high-risk watch list. These accounts potentially use the networks of money transfers identified by Weakly Connected Components to conduct money laundering. Centrality algorithms detect liaisons, central and more relevant players in these money transfer networks. Clustering algorithms, Louvain Modularity and Strongly Connected Components can be used to identify subnetworks and high-risk rings, respectively.

If there is a path between a customer and a high-risk end point, then Jaccard, Cosine or Overlap Similarity algorithms indicate how similar each path is to other paths from that specific customer to those high-risk endpoints. Then create a subgraph of paths and have weighted relationships representing similarities among the paths. Pathfinding and search algorithms identify payment chains and third parties layered between customers or transactions and other endpoints.

To illustrate, the following script loads an in-memory graph of money transfers between account holders.

```
CALL gds.graph.create('Payments', 'AccountHolder', 'MAKE_PAYMENTS_TO',
  {nodeProperties:['highRisk'],
   relationshipProperties:['maxAmount', 'avgAmount', 'sumAmount', 'numPayments']});
```

On this in-memory graph, the following script executes Weakly Connected Components (WCC) to identify a cluster of account holders in a payment chain that high-risk accounts use to conduct money laundering.

```
CALL gds.wcc.stream('Payments',{concurrency:16})
YIELD componentId,nodeId
WITH componentId AS cluster,gds.util.asNode(nodeId) AS customer
WITH cluster,collect(customer) AS customers,sum(customer.highRisk) AS fraudNum
WITH cluster,size(customers AS clusterSize,fraudNum WHERE fraudNum>0
AND clusterSize>1
RETURN cluster,clusterSize,fraudNum
ORDER by fraudNum DESC;
```



cluster	clusterSize	fraudNum
0	93217	324

Started streaming 1 records after 383 ms and completed after 383 ms.

One cluster has 93,217 account holders, of which 324 are high-risk accounts

How to Combat Money Laundering Using Graph Technology

Each customer has a property called `amlNetwork`. The WCC algorithm updates this property for every customer in the cluster.

```
CALL gds.wcc.stream('Payments',{concurrency:16})
YIELD componentId,nodeId
WITH componentId AS cluster,gds.util.asNode(nodeId) AS customer
WITH cluster,collect(customer) AS customers,sum(customer.highRisk) AS fraudNum
WITH cluster,customers,size(customers) AS clusterSize,fraudNum
WHERE fraudNum>0 AND clusterSize>1
UNWIND customers AS customer
MATCH (a:AccountHolder)
WHERE a.accountHolderID=customer.accountHolderID
SET a.amlNetwork=cluster;
```

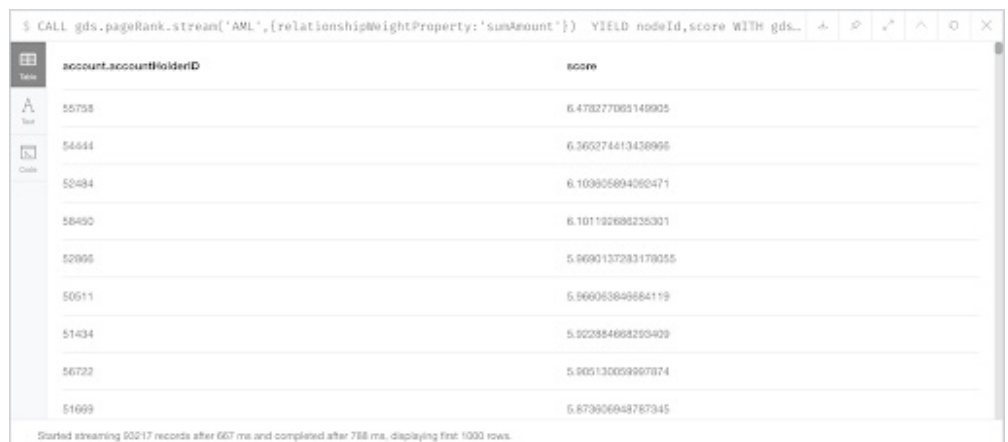
The following scripts create an in-memory graph named **AML** and execute 3 more algorithms:

- *PageRank* detects the most relevant players in the network
- *Closeness* detects the accounts that are closer to any other account (by number of hubs)
- *Louvain* identifies subnetworks. Each of these subnetworks can be used to prioritize the anti-money laundering investigation based on the size of the clusters and the relevance (*PageRank* score) of the accounts contained in each cluster

```
CALL gds.graph.create.cypher('AML',
'MATCH(a:AccountHolder) WHERE exists(a.amlNetwork)
RETURN id(a) AS id,a.highRisk AS highRisk',
'MATCH(a1:AccountHolder)-[r:MAKES_PAYMENTS_TO]->(a2:AccountHolder)
WHERE exists(a1.amlNetwork) AND exists(a2.amlNetwork)
RETURN id(a1) AS source,id(a2) AS target,r.maxAmount AS maxAmount,r.sumAmount AS
sumAmount,
r.avgAmount AS avgAmount,r.numPayments AS numPayments',
{nodeProperties:['highRisk'],
relationshipProperties:['maxAmount','avgAmount','sumAmount','numPayments']});
```

```
CALL gds.pageRank.stream('AML',{relationshipWeightProperty:'sumAmount'})
YIELD nodeId,score
WITH gds.util.asNode(nodeId) AS account,score
RETURN account.accountHolderID,score
ORDER BY score DESC;
```

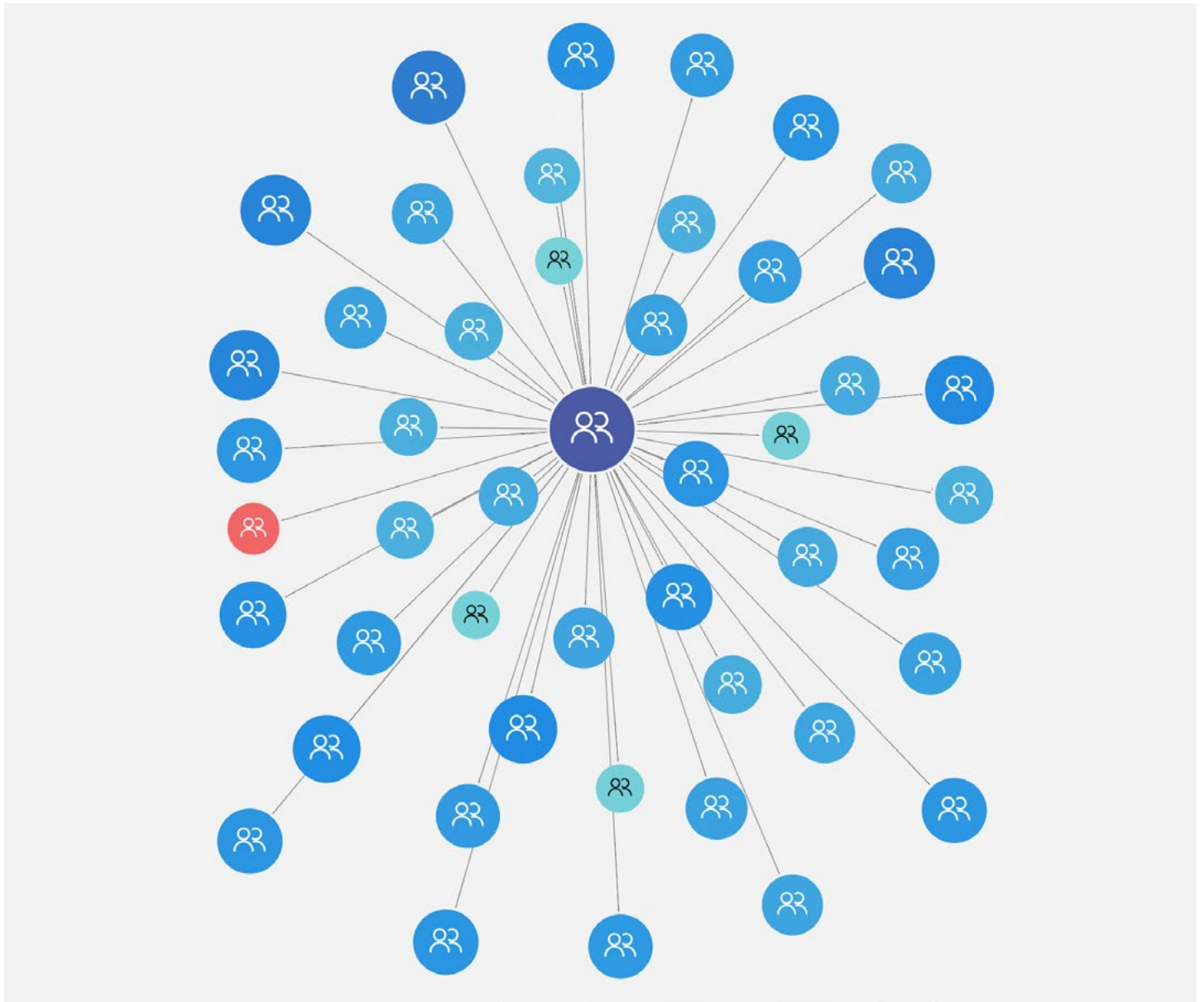
```
CALL gds.alpha.closeness.stream('AML',{concurrency:16})
YIELD nodeId,centrality
RETURN gds.util.asNode(nodeId).accountHolderID AS customer,centrality
ORDER BY centrality DESC;
```



The screenshot shows a Neo4j Cypher query result window. The query is: `CALL gds.pageRank.stream('AML',{relationshipWeightProperty:'sumAmount'}) YIELD nodeId,score WITH gds.util.asNode(nodeId) AS account,score RETURN account.accountHolderID,score ORDER BY score DESC;` The result is a table with two columns: `account.accountHolderID` and `score`. The table contains 10 rows of data. At the bottom, a status bar indicates: "Started streaming 50217 records after 667 ms and completed after 788 ms, displaying first 1000 rows."

account.accountHolderID	score
55758	6.470277005149905
54644	6.365274413436996
52484	6.100905894260471
58490	6.101192698235301
52996	5.9690137283179255
50511	5.969053849584119
51434	5.922884668295429
56722	5.905130059967874
51699	5.873906648787345

How to Combat Money Laundering Using Graph Technology



This graph uses color and size to depict the riskiest entities based on their relationship to the red account under investigation

Graph Data Visualizations

Neo4j allows for money queries to be viewed in various ways. Output can feed downstream applications or be displayed directly to users:

- **Executives** often want interactive dashboards with graphic visualizations and green, yellow and red scoring of key performance indicators
- **Analysts** prefer to explore context behind dashboard results without writing code and using IT-developed queries in playbooks
- **Data Scientists & Power Users** may write their own ad hoc queries or graph algorithms or to use the results of graph algorithms to segment data and identify central entities

How to Combat Money Laundering Using Graph Technology

Dashboards and APIs

Tableau Dashboard

Neo4j has a [Business Intelligence \(BI\) Connector](#) that connects to Neo4j Graph Database from business intelligence tools such as Tableau.

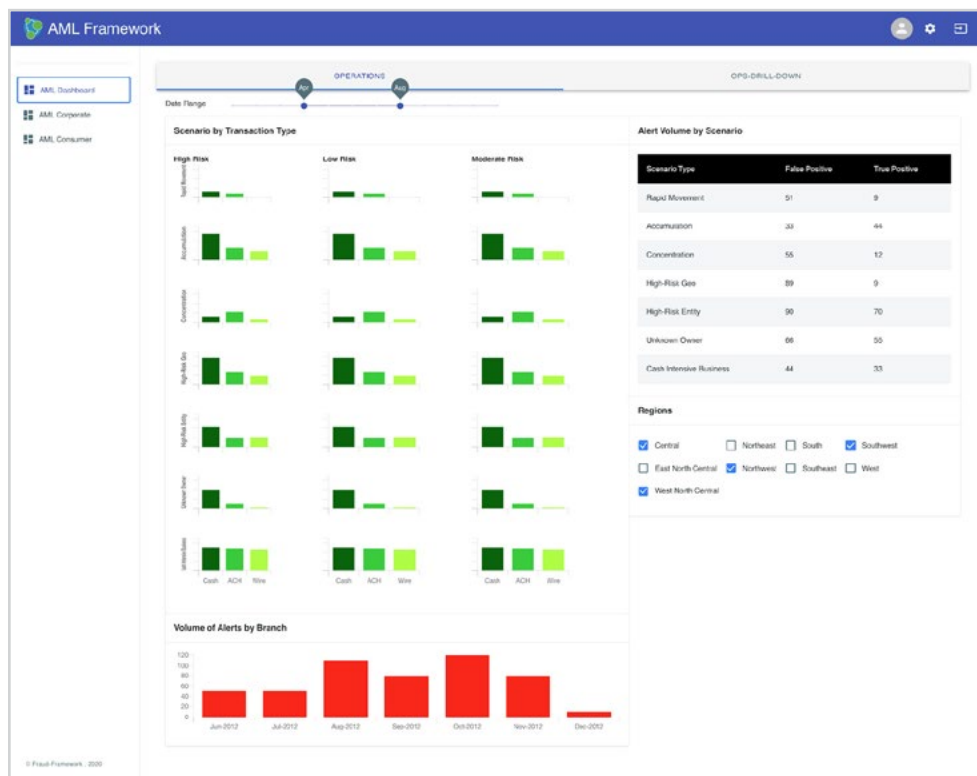


The Neo4j BI Connector can be used to create AML dashboards such as this one from Tableau.

Neo4j's BI Connector makes it easy to build dashboards and visualizations using popular business intelligence tools.

React Dashboard

Developers build dashboards using the GRANDstack (GraphQL, React, Apollo, Neo4j Database). The React dashboard pictured below is built atop the GraphQL API.



This React dashboard provides a quick summary of risk profiles by financial transaction type

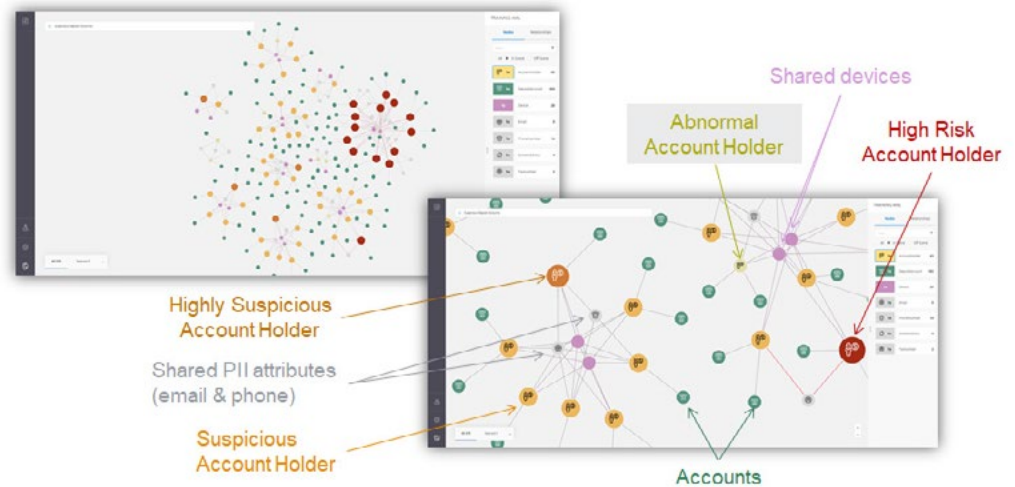
How to Combat Money Laundering Using Graph Technology

The sheer volume of AML data is impossible to comprehend without a self-service data visualization tool like Neo4j Bloom that supports real-time, natural language and ad hoc analysis.

Graph Visualizations

Shared Attributes

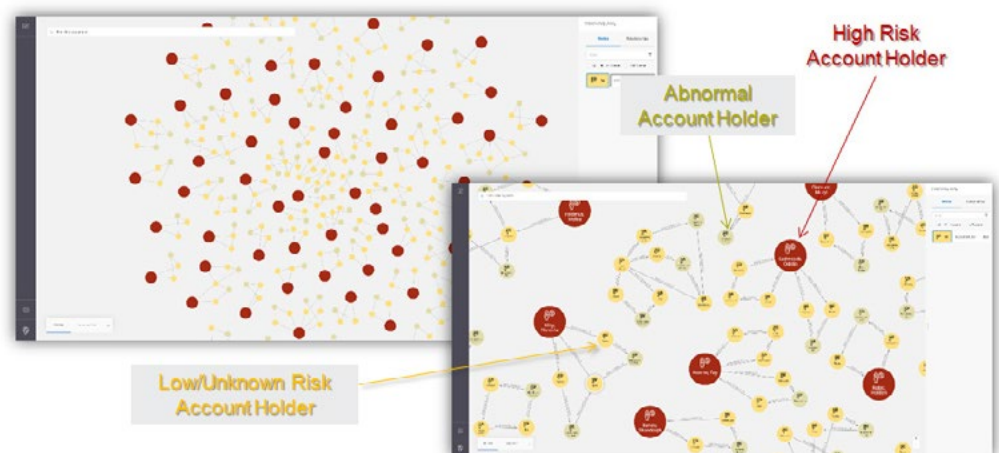
Shared attributes can indicate that entities are the same. These kinds of patterns are useful in both ER and flagging processes. The [Neo4j Bloom](#) visualizations below show account holders who share attributes – such as a tax number, address and phone number – with other account holders. The intensity of the shade of red increases as suspiciousness increases.



Neo4j Bloom helps analysts visualize which accounts have shared attributes, making it more likely that they have the same high-risk owners

Circular Payments: Placement, Layering, Integration

One common money laundering technique is to use circular payments to exchange dirty money for laundered assets. The Bloom graph visualizations below depict circular payments. This is a mono-partite graph of scored account holders. Red nodes are high-risk, tan nodes are medium-risk and yellow nodes are low-risk.



Bloom visualizations help detect circular payments that exchange dirty money for laundered assets.

How to Combat Money Laundering Using Graph Technology

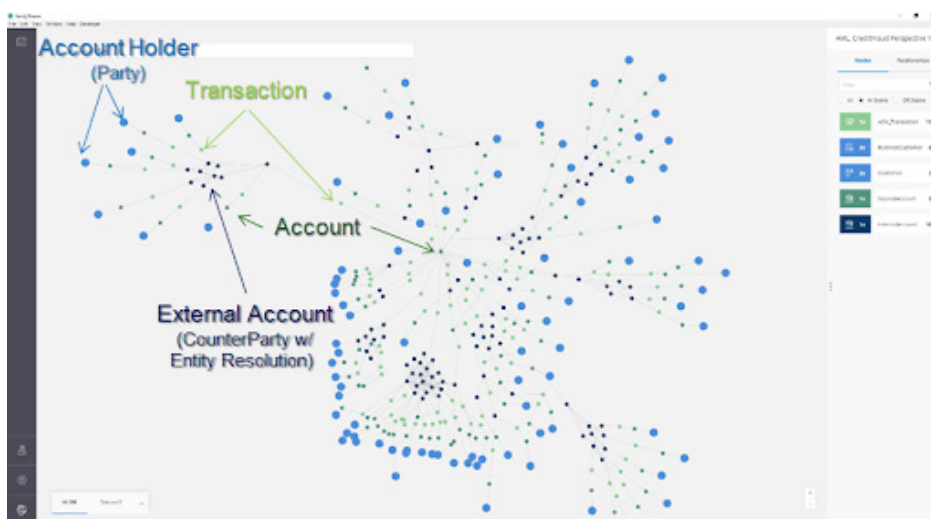
Neo4j Bloom enables analysts to visualize suspicious payment chains and money laundering rings from end to end.

Payment Chains

Once an entity reaches a threshold of risk that strongly indicates suspicious behavior, a subsequent set of templated queries look for high-risk entities associated with a party.

A common money laundering behavior is to transfer funds to cohort entities for pass-through transactions. These transactions are often layered several levels deep. A templated query result worth exploring could lead to discovery of a payment chain.

The Neo4j Bloom graph visualization below depicts a payment chain starting from a suspicious entity on the upper left. The ER scores for individuals, corporations or financial institutions who might be the same (based on ER process scores) are depicted in dark blue clusters. The process flagged multiple accounts in dark green for investigation of connections to the original, suspicious cluster that appears at the upper left.



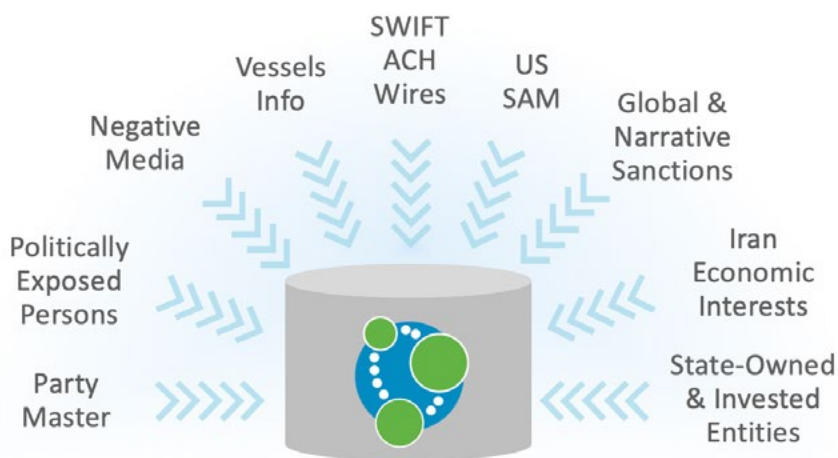
Neo4j Bloom helps bankers and regulators detect and visualize payment chains and identify suspicious entities and events.

How to Combat Money Laundering Using Graph Technology

Data Sources & Input API

A Neo4j anti-money laundering solution ingests data from multiple sources.

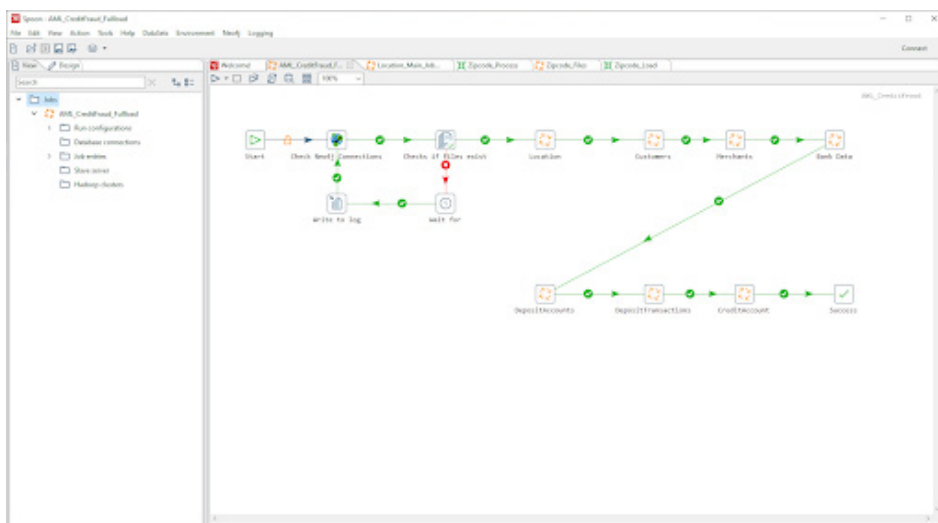
The ingestion pipeline is model-driven, automated and flexible enough to stay ahead of the fast-changing techniques used by money launderers. The Neo4j AML Framework processes billions of transactions per day in near real time and can be extended to address all GRC functions beyond AML.



Hop ingests many sources into one Neo4j database

Hop provides a flexible API to handle constantly-changing data sources and money queries. It has plugins to read SWIFT, ACH, wire transfers and other transaction formats, and it can also utilize Java libraries for that purpose.

Hop's automated, model-driven approach makes it ideal for automating ingestion pipelines, processing more than a million rows per second. It consolidates from multiple data sources, parses and extracts entities, validates, enriches and normalizes data. Hop also connects to native sources directly and maps data formats to the Neo4j graph data model API.



A Hop pipeline is an automated, model-driven process that ingests data from multiple sources.

The Neo4j AML Framework can unify data from internal and external sources into a single graph database, easing advanced analysis and visualization.

Solution Framework Performance Testing

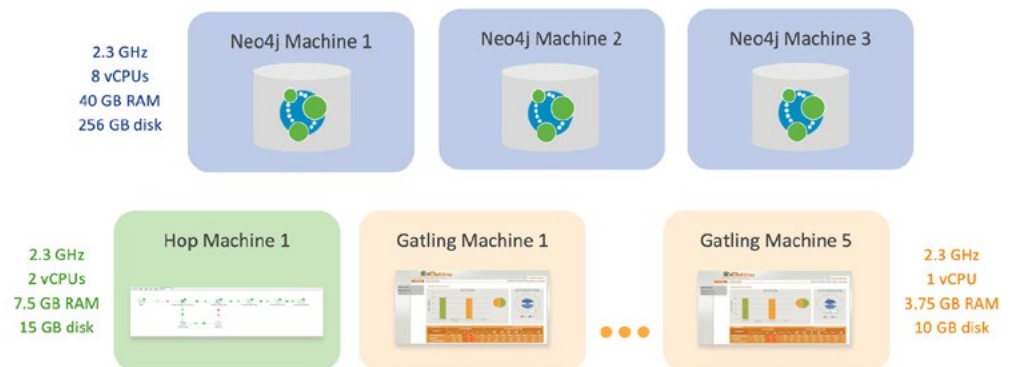
Neo4j is performant, trusted, available, secure, agile and extensible, and it scales linearly to hundreds of billions of ingested records while still returning complex pattern-match results in milliseconds. The Neo4j AML Framework easily meets service level agreements for tens of millions of inserts per day.

The Neo4j AML Framework is an extensible framework with customizable plug-and-play components. AML staff can customize and extend the base framework with:

- Money queries
- Custom graph data models
- Internal and third-party data sources and streams
- Ingestion APIs
- Output APIs

The following configuration shows the best practice for stress testing and performance tuning to ensure that the Neo4j AML Framework meets the most stringent service level agreements.

The largest AML workloads process tens of millions of transactions and party inserts per day. Meanwhile dozens – or hundreds – of analysts and other GRC staff analyze entities and transactions. They typically require sub-200 millisecond query response time on 99% of all queries by dozens or hundreds of active concurrent users while simultaneously processing thousands of write transactions per second.



Neo4j tests its frameworks using the Gatling test harness.

How to Combat Money Laundering Using Graph Technology

The Next Stage of the AML Battle

Winning the battle against money laundering requires a technology that better harvests information from transactions – and other sources – and better detects suspicious activity in real time and at scale. This has been challenging because companies process billions of transactions per day involving tens of millions of parties.

The first step in improving detection is to harvest the information from transactions by connecting it to already-known information. The next step is applying algorithms that leverage relationships to pattern match and score relationships and behaviors that connect a network of people, places, corporations, financial institutions, merchants, transactions and events.

By leveraging Neo4j to connect data, compliance teams can:

- Better comply with AML requirements and make more accurate predictions, thereby saving money on fines and detecting real money laundering more accurately
- Reduce costs associated with fines and with investigating false positives and false negatives
- Increase sales by improving brand-value reputation
- Better comply with other global risk and compliance (GRC) requirements
- Meet the most stringent AML requirements for performance, availability, security and agility at extreme scale

Learn More about the Neo4j AML Framework

To learn more about using Neo4j as the foundation of your anti-money laundering initiatives, [contact us today](#).

Neo4j is the leader in graph database technology. As the world's most widely deployed graph database, we help global brands – including [Comcast](#), [NASA](#), [UBS](#), and [Volvo Cars](#) – to reveal and predict how people, processes and systems are interrelated.

Using this relationships-first approach, applications built with Neo4j tackle connected data challenges such as [analytics and artificial intelligence](#), [fraud detection](#), [real-time recommendations](#), and [knowledge graphs](#). Find out more at [neo4j.com](#).

Questions about Neo4j?

Contact us around the globe:
info@neo4j.com
neo4j.com/contact-us