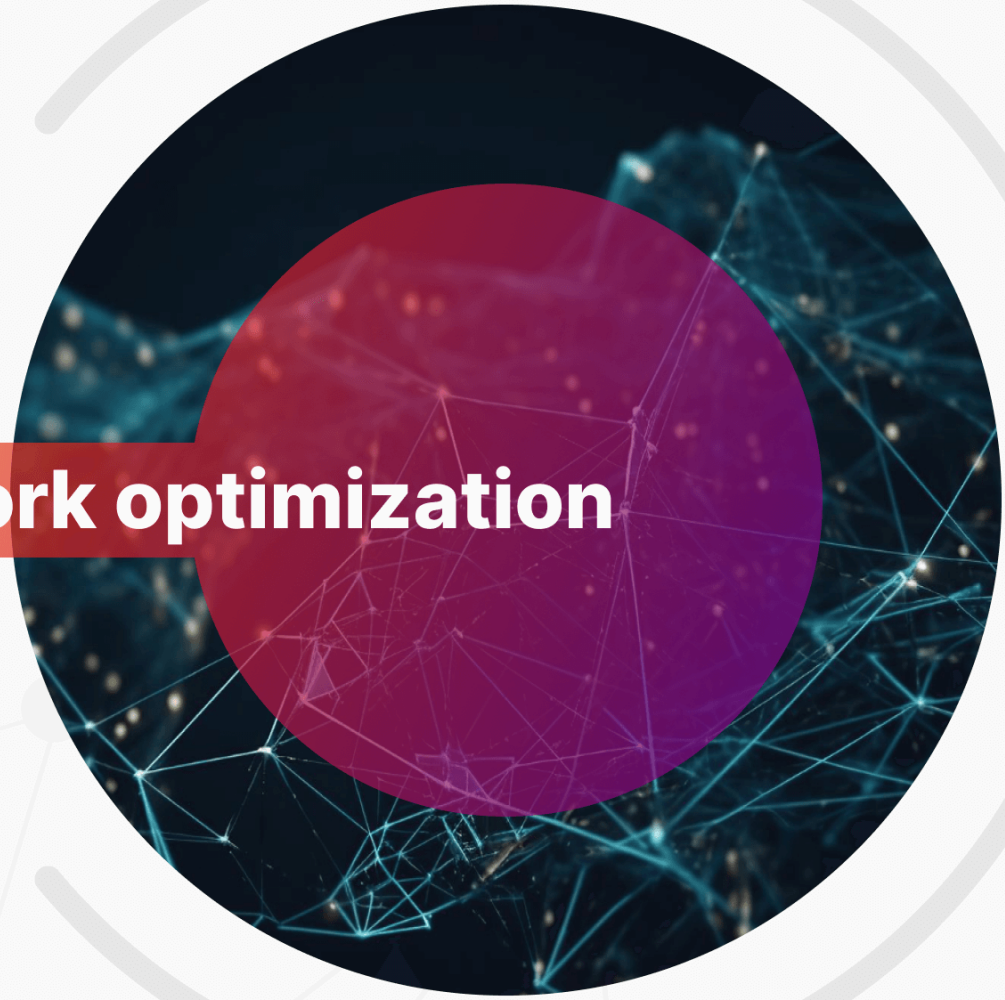
MEM GRAPH



Network optimization

WHITEPAPAPER

Graph Technology in Network Resource Optimization

Table of Contents

Introduction

Networks are a concept that is easily grasped by humans. This is due in part to the visual nature of networks, as they are often depicted through drawings that make information easily accessible. The act of memorizing patterns through these diagrams is much simpler than reading through a traditional textbook filled with highlighted facts.

In design, networks are often depicted through sketches that show processing jobs represented as nodes and connections between them, signifying the order of execution.

Despite the simplicity of networks, some industries that rely heavily on network-like resources struggle with their management.

Graph databases Are the Future for Network Resource Optimization

One study shows that over 84% of companies use over-provisioned cloud resources due to the dynamic demand of processing power during the day.

Chemical plants still done without reliable software tools to perform network and impact analysis on the chemical topology to identify how to improve it.

Supply chain management involves creating a network of processes to reduce delivery time from production to shipping. Despite efforts to optimize process steps, they are often executed by manual staff and are prone to mistakes and subpar solutions. This approach also consumes valuable time that could be used for tasks that require more expertise and skill.

In telecom and power grid systems, power outages occur frequently due to disconnections between neighbouring components. A critical power outage in places such as an airport can result in significant financial losses for a business. To prevent power outages, companies can install redundant components to ensure consistent power supply, although this can be expensive.

The common thread among these industries and the issues they face is that their core entities are networks and their data should be represented and analyzed as such. However, often inadequate management tools and solutions, such as relational databases, are utilized. In this article, we will examine why relational databases are ill-suited for these scenarios, and how graph databases, with their natural network representation of data, can easily handle network analytics and solve these problems.

The issue with relational databases

The advances in technology and processing power created a necessity for storage that would store data and information. Relational databases arrived as the first and optimal solution at the time because of their simplicity. With tabular storage, it wasn't difficult to analyze and aggregate data, and it was well suited for all use cases in all industries. If the database performance degraded, a simple solution was to buy a new, more performant computer.

With the emergence of big data, vertical scaling (upgrading a single machine, adding CPU cores and RAM) became a problem since that kind of scaling is much more expensive than horizontal scaling. What's even worse, in most cases, it doesn't effectively optimize the resources. IT experts began to question whether relational databases are the optimal storage for some use cases, and as a result, NoSQL databases were created, with document (e.g. MongoDB), columnar (e.g. Cassandra) and graph storages (e.g. Memgraph).

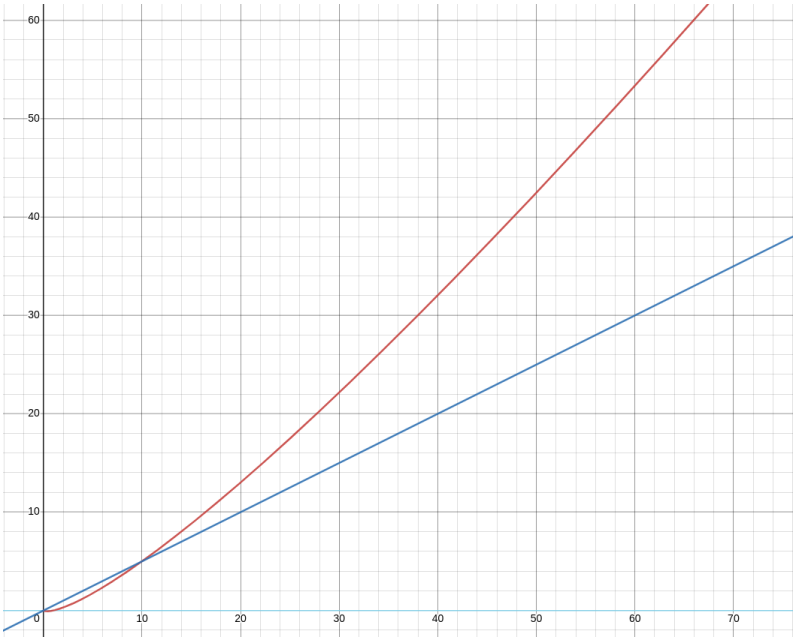
So, why are relational databases not performant enough for the use cases mentioned in the introduction? Let's dissect them one by one, keeping in mind that the topology of the data looks like a network when drawn on a piece of paper.

Visualizing the topology takes around a third of the processing time

In the energy network use case, on average, 25 - 35% of processing time is spent on loading the topology in memory and getting it ready for analysis. The numbers are actually quite similar in other industries as well. If looked under the hood, we can see that in relational databases, components are commonly stored in tables, and to join all the components, the database needs to create as many recursive joins as there are hops between the farthest two components in the network. The complexity of one JOIN is O(n * log(n)), and the number of recursions can grow up to k, with k being the

number of hops from 2 of the most distant components in the topology, which brings us to the big O notation of O(k * n * log(n)).

If we look at the graph below, it shows a much steeper rise in the logarithmic complexity of loading the topology (red line), with respect to the linear complexity (blue). Performance gets worse as the number of components in the network increases.



This graph alone shows storing entities as rows on a disk in tabular storage is not optimal for network data such as connected chemical plants, power grids, mobile stations, cloud resources or supply chain pipelines.

Tabular data cannot answer questions related to network analytics

Relational databases were originally developed to retrieve a single piece of information from one point of an application. They are excellent for providing profile information, order information and similar questions that need to fetch one piece of data or combine them with one other piece of information by using a single JOIN. With big data, big aggregation functions were considered a problem, which was solved by transposing rows into columnar storages (Cassandra, Scylla), capable of processing a large number of numerical information at a time and providing statistics for grouped entities.

However, they still come up short when addressing the following questions:

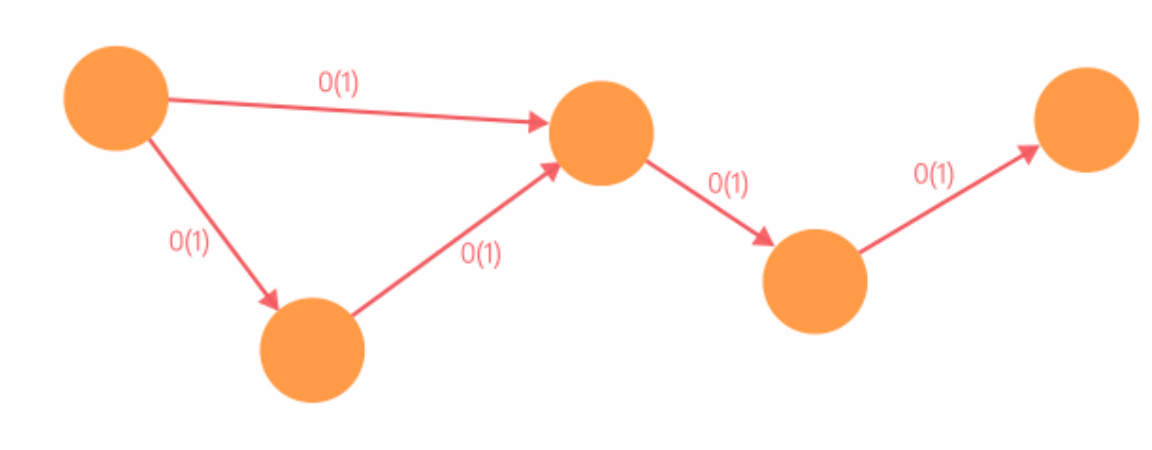
What are the first N jobs in my supply chain that can be executed first to resolve blockers in production?
What is the shortest path between 2 of my chemical plant components?
If I remove this power grid component, what would be the consequences of removal?
Is my graph disconnected, and does that mean there is a network outage?
By how much do I need to adjust my resource allocation during the non-busy part of the day, based on the current numbers?
What is the topology in my supply chain that yields the maximum production of resources?

Any of these questions translated into SQL would generate an extremely complex query. But most importantly, it would not be an optimized query, as it requires network traversals, as well as identifying dependencies and local neighborhoods of the network components. Network analytics mostly consist of structure analysis and less about analyzing information stored as components' properties

Enter graph databases, a natural storage for networks

So why are graph databases optimal for network analysis and resource optimization? The answer lies in their structure. The first class entities of graph databases are nodes (also called vertices) and relationships (also called edges). A correspondent mapping for nodes would be rows in a relational table, whereas relationships would map to foreign keys used for joining the data in tables.

The main conceptual difference is that data is automatically joined as it is stored in the graph database, making sure traversing the graph is done in constant time. Imagine you have a network of connected components in your chemical plant. It should be quite easy to get information on where the outputs of a chemical reactor go next since you can see its pipes and the component next to it. When using relational databases, the process can be described as going to the central registry in your company (many-to-many table) to check information about the connected components, coming back and realizing that the component connected is actually right next to it. Graphs instantly show the connection.



The time complexity of traversing the whole graph and displaying a network is O(V + E), where V + E is the joined number of nodes and relationships in the graph. It equals the linear complexity (the blue line in the chart above) with respect to number of entities in the graph, which offers much better performance compared to logarithmic complexity of relational databases.

Graph representation of data allows new analytical queries

How can graph storage be utilized for companies' benefits? As a language, SQL was not a good fit for graph databases, so a new query language was designed to manage network traversals - openCypher. Let's see a few examples of its syntax. Below is a query that returns the whole database:

```
MATCH (n:Component)-[r]->(m:Component) RETURN n, r, m;
```

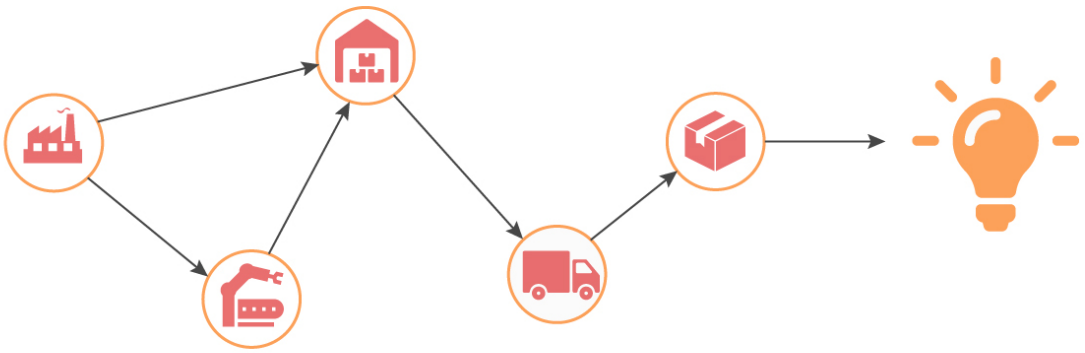
In the query, (n:Component) and (m:Component) represent nodes of Component type and [r] is the relationship between them. The language of openCypher is designed to match patterns in the graph, and from there expand to the whole graph or a subgraph. Seems pretty straightforward and without any need for recursions, the SQL would require.

Let's say that we want to find the shortest path between 2 components in a graph. It can be easily done with:

```
MATCH p=(n1:Component {id:1})-[r:IS_CONNECTED_TO
*wShortest]->(n2:Component {id:2}) RETURN p;
```

The built-in function of graph databases, wShortest, yields the shortest path between 2 components with corresponding IDs. But that's not the only traversing possibility in a graph database. Many algorithms, such as depth-first search, breadth-first search, and more complex graph algorithms, such as PageRank and Community Detection, were intertwined with the core of graph databases to provide industries with a different kind of analytics that will result in discovering new information and insights.

Querying and traversing a network and inferring new knowledge from it should be as easy as thinking about it!



All-in-one tool for your intelligence

Graph database providers have made an effort, not only to make a good representation of storage but also to include various graph algorithms and analytics possibilities, as well as visualization tools. What use is optimal storage if the results are displayed as rows and not rendered as a network in front of you? It's just an optimal storage, without the ability to clearly see and infer patterns yourself about possible bottlenecks in the graph. Moreover, graph analytics and algorithms could be used in a way to highlight and customize your view of the network to detect problems and solve them with ease.

All this makes graph databases not just a storage but a platform for graph analytics. Questions tied to impact analysis, root cause analysis, dependency analysis, simulation of digital twins, and many more, can now be applied in industries consisting

of highly connected network data to infer new intelligence and provide value to the business. It's also cost-efficient to use only one component, and not a complex system of connected tools.

Perform Fast Network Analysis on Real-Time Data With Memgraph

The introduction of automated solutions in various industries brought a new way of managing data and storing it in a centralized database. With the arrival of big data, companies needed to process and distribute millions of data points quickly, which became a challenge for traditional relational databases. They are great for representing data sets, but not efficient for handling topologies and networks.

Graph databases, like Memgraph, are better suited for this task as they have the ability to handle complex network analytics, storage, visualizations, analytics, and streaming capabilities.

Annihilate performance issues with in-memory storage

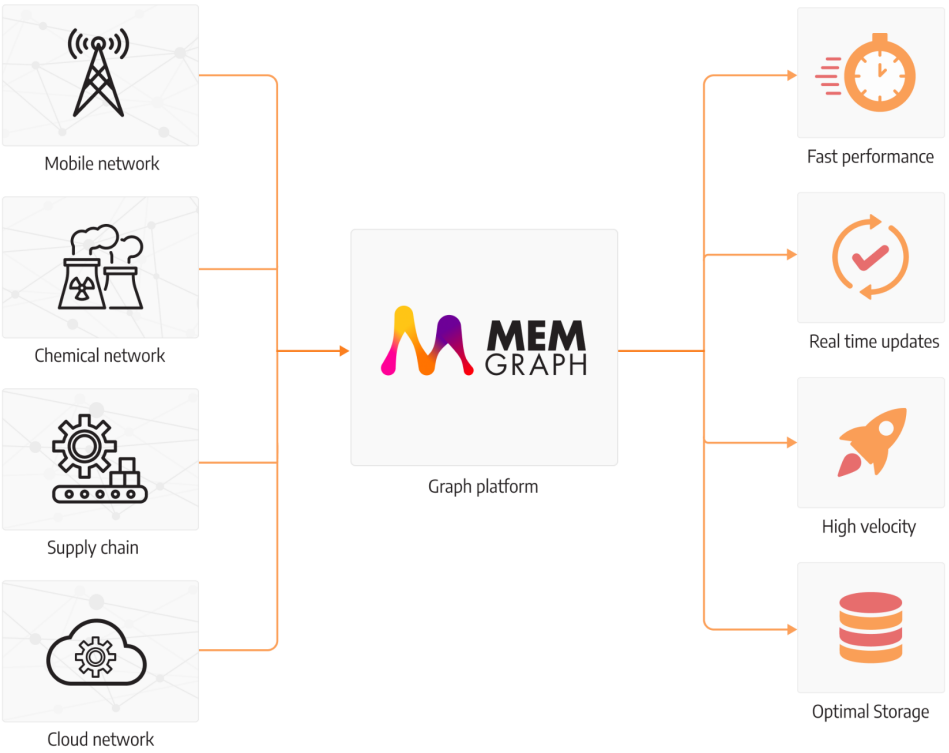
Around 25-35% of query processing time in companies working with networks and network resources is spent on loading topology. The main reason for slow performance is a relational database used as storage. However, changing the type of storage is not enough if the performance degradation is caused by loading a large dataset into memory from a disk.

This is especially a problem in industries such as power management companies and chemical plants, which heavily rely on highly connected components (generators, reactors). Their networks sometimes consist of thousands of connected components. Just imagine how complex is the network of energy pipelines of a whole continent like Europe. Another performance degradation can happen when systems need to instantly execute new online analytics to correctly reflect the changes in supply and demand and provide companies with new information.

Mobile networks are another example of a network of highly connected data, with hundreds of mobile base stations with antennas a phone can connect to for

establishing a call. Since mobile users are moving all the time, the system needs to efficiently re-assign new base stations to provide a stable connection.

The supply chain industry is a network of a vast number of production lines with different pipelines assembling products for shipping. To maximize production efficiently, the production schedule needs to be dynamically adjusted.



Memgraph, as a platform, offers an in-memory graph database. All the data is loaded into RAM during startup ensuring efficient querying, and the memory footprint is further reduced by efficient features written in C++. Graph storage has been designed to represent data networks and allows the speedy execution of complex traversing queries that can prepare the topology for analysis without friction. As networks of data are usually queried from a certain starting node expanding to nearest neighbors or subgraphs, the Cypher query language was designed to allow easy traversals, pathfinding, and loading of the necessary nodes and relationships for inspection.

All in all, graph databases are the solution to performance issues companies have to endure when faced with gigantic networks of highly connected data!

Answer your deepest network analysis questions with graph algorithms

In industries that heavily rely on networks, certain questions need to be answered. Let's see what questions dominate which industries:

Supply chain management

In what order do tasks need to be executed so that my production lines are idle as little as possible during the day?
What are the dependencies of the product I want to make?
Given the current number of ingredients in stock, how many instances of the final product can I make in 5 hours?

Chemical engineering

What is the minimal number of sensors that need to be placed across my plant components to monitor variables in the system?
What is the maximum flow produced out of my system?
What's the quantity of material across all the connections between reactors that yield maximum flow?
If I add a reactor to the network, does it increase production efficiency?

Mobile networks

How to assign codes to mobile base stations to ensure they all have distinct code numbers and that switching between base stations is executed correctly?

Power management systems

What are the 10 most critical points in my power grid that need to be strengthened to reduce the risk of a power outage?

Cloud computing

Given the current network traffic at the moment, how much of my resources should be provisioned?

The SQL query language used with relational databases was designed to aggregate lists of data, not traverse the network. The task of formulating a query to answer any of these questions regarding network analytics is extremely complex.

Although Cypher easily solves tasks involving network traversals, these queries might be too complex to write even in that language. To make every data analyst's life easier, Memgraph has developed an open-source library of graph algorithms and

integrations with various graph tools called Memgraph MAGE [Memgraph Advanced Graph Extensions](https://memgraph.com/mage). The library consists of query modules that can be called and executed inside Cypher queries to perform advanced analytics and gain insights from the network storage. Centrality algorithms like PageRank or betweenness centrality, community detection algorithms, and even Graph Machine Learning are now possible to be executed directly on the database level to reduce latency and give immediate insights.

Let's look at a few examples of calling these algorithms to answer the questions above:

What are all of the dependencies of the made product?

```
MATCH (n:Node {id:1})
CALL graph_util.descendants(n)
YIELD descendants
RETURN descendants;
```

The algorithm above yields all descendants of the source node, from which another node can be reached by following directed paths in the graph. It would give us a good amount of information about which products in the supply chain are followed after a certain process.

What are the top 10 critical places in my power grid that need to be strengthened, reducing the risk of power outage?

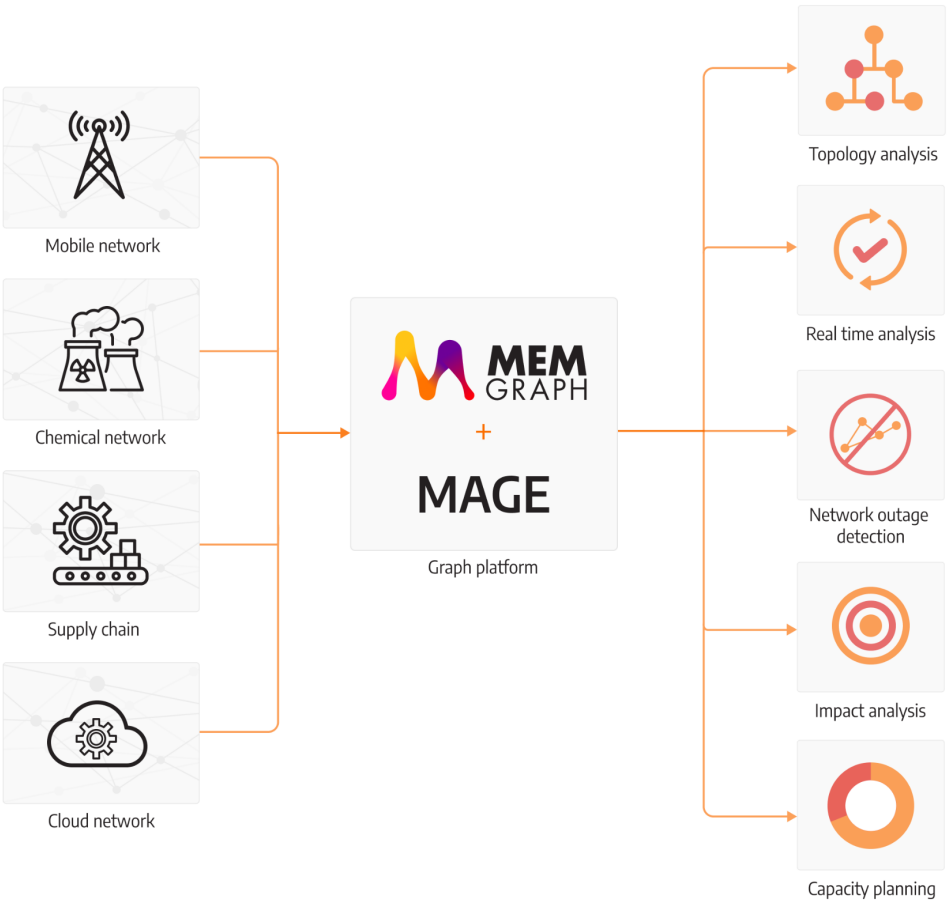
```
CALL betweenness_centrality.get()
YIELD node, betweenness_centrality
SET node.rank = betweenness_centrality
RETURN node, betweenness_centrality
ORDER BY betweenness_centrality
LIMIT 10;
```

Betweenness centrality can yield critical hub points in the graph, through which a majority of paths between pairs of nodes are passing. It would be an excellent indicator of weak points that could help prevent power outages between 2 distinct areas in the country or analyze critical points in the supply chain pipeline.

These are only two answered questions, but all of them mentioned above can be answered with graph algorithms or traversal analytics on the data inside the Memgraph database. The result - analytical queries written with ease to gain instant network analysis insights!

But what if you need some algorithm that isn't available in the library?

There is an answer to it as well! MAGE is open source and highly extendable with additional graph algorithms or custom analytics. APIs for integrating with MAGE accept Python, C++, Rust and C language for seamless integration with your own query modules to solve business problems in your company!



Stay up-to-date with real-time streaming capabilities and dynamic graph algorithms

One of the bigger issues when dealing with a large number of updates in the network state is keeping up with the real-time demands and providing up-to-date information. Even though relational databases can handle a large number of writes in the database, the challenge of recomputing the answers to the questions asked above becomes unbearable and they can no longer guarantee to provide the latest insights.

Unable to keep up with real-time analytics, companies turn to processing data in batches and recomputing information from all the data points at fixed points during the day or night (e.g. nightly reports). However, some situations require updates as soon as new data comes in.

For example, if the flow of materials in the chemical reactor changes, the parameters in the network need to be adjusted without delay. There can be multiple different chemical reactions happening in the same component at different times, so conditions need to be updated in real-time for safety reasons.

On the other hand, if the chemical plant is still being constructed, a simulation of a digital twin needs to be run enough times to decide what is the most fitting topology for the purpose.

Sometimes, we need to react quickly, not only to prevent things from escalating but also to fix unexpected situations that have already happened. In cloud computing, when there is a suspiciously high amount of fees that need to be paid for computing resources, a root-cause analysis needs to be performed in order to detect the cause of spending extra budget on unused resources and cut the spending.

All of these examples have an identical weakness - can they be performant enough if the number of updates rises per unit of time (increasing the data velocity) to provide crucial information the moment they are required? Can they be performant if the number of updates is fairly low, but the amount of time it takes to recompute all the insights still doesn’t meet the real-time expectations of making updates? Even if the computer is high-performant by itself, recomputing insights from a changed graph has to be done in real-time.

To achieve high performance with a large number of updates is to recompute results from the local change in the graph without recomputing results on the whole network.

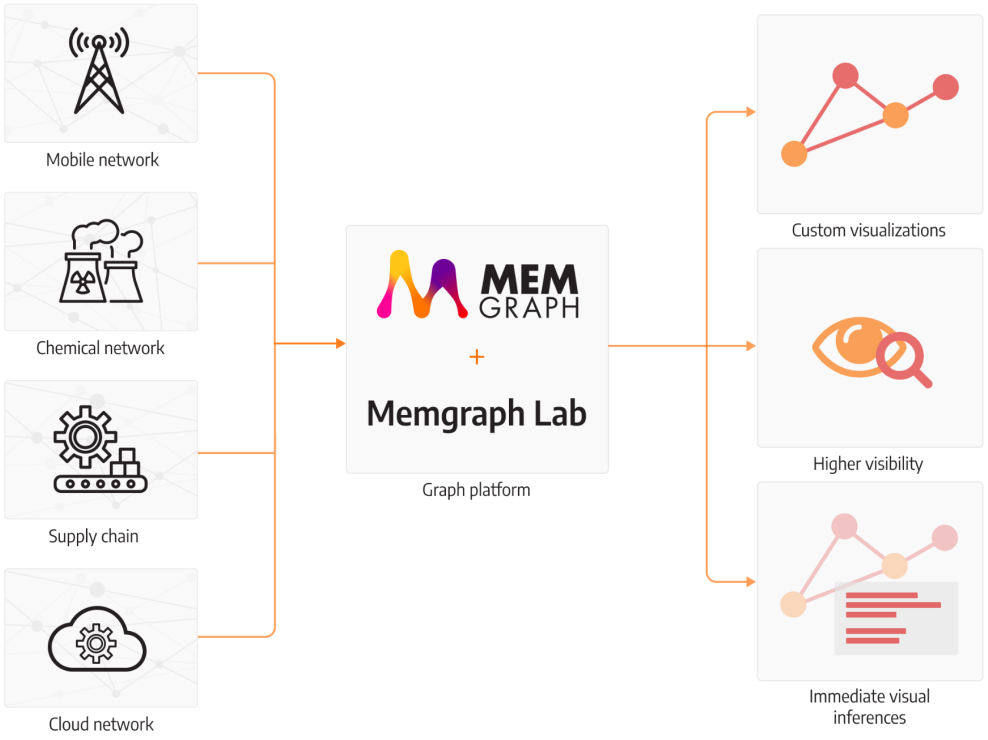
Memgraph extensively researched how to tweak the traditional graph algorithms to handle streaming data and created dynamical graph algorithms. Combined with database triggers, they offer a powerful solution to stay up to date with the newest information from the network, no matter how fast data in the network changes!

To ease up collecting data from streams as much as possible, a “portal” has been opened for message queues, such as Kafka, Redpanda and Pulsar. Memgraph can quickly plug and play to receive streaming data in an instant, therefore connecting the graph from the get-go.

Observe new patterns by inspecting visualized data

If you somehow managed to get intelligence from the painstakingly written SQL queries, it’s displayed in rows and rows of data. This tabular data cannot adequately preview how the network behaves. When presented with a dataset, it’s hard to infer what decision should be made at the company level. Humans are much better at processing images rather than plain data. Some research shows we are 60,000 times faster at processing images than text.

Visuals also extend the human capacity to take in, comprehend, and more efficiently synthesize large amounts of new information, especially to find patterns and relationships. Sometimes, an expert must react without hesitation to resolve an urgent issue in the network. In a case of a network outage, a quick analysis should be sufficient to identify the culprit node that caused the outage. In case of constraints breach in chemical analysis, regardless of whether there is too much material in one component or an imbalance of ingredients in the product, the expert needs a clear alert to react promptly.



Memgraph Lab is a visualization tool that can render the current state of the graph in a matter of milliseconds. The extensive possibilities of customizations allow for making the best view for the user to make the most out of their perception skills. After all, software tools exist to suggest and advise experts on what is going on and provide several options. Experts should be able to perform a quick analysis and decide what's the best course of action to improve and increase the efficiency of production, handle network outages, or deploy a team of repairmen to fix the bottleneck in the power grid.

Optimize and Manage Supply Chain Network With Memgraph

The logistics of managing the complex interdependencies in supply chain management can be challenging. Expert teams are often tasked with creating optimal schedules for product production, always seeking to make the process faster, cheaper, or better. Graph analytics and Memgraph can simplify this process, providing accurate and meaningful information for optimizing the supply chain network.

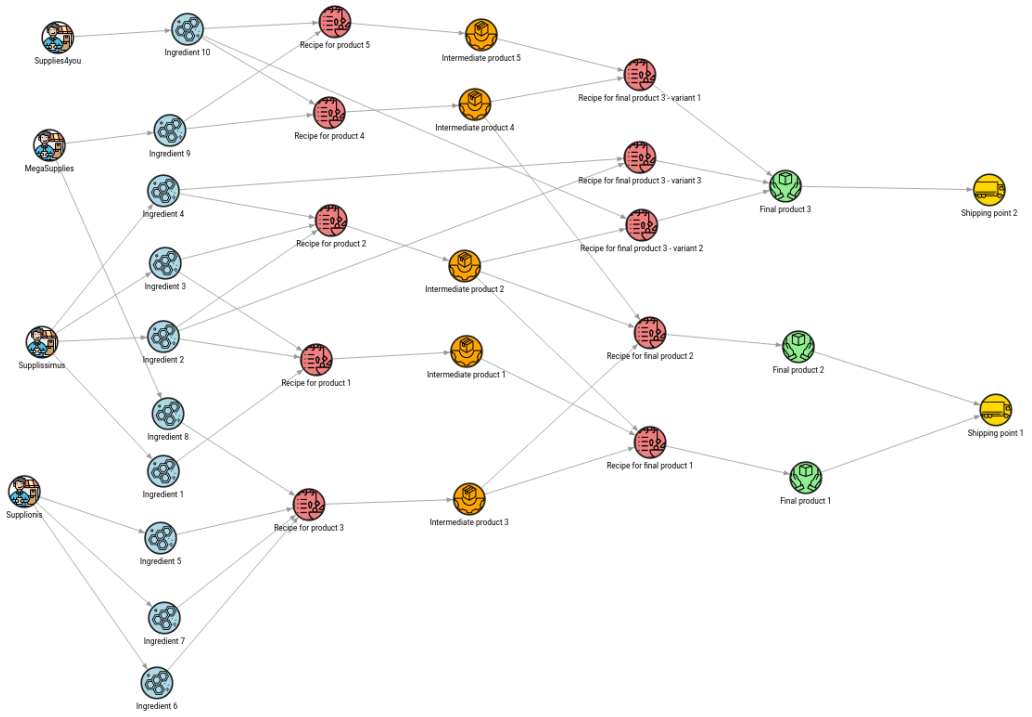
Below is a simple supply chain graph example that uses visualizations made with Memgraph Lab and query algorithms from Memgraph MAGE to explore graphs and perform important analytical tasks.

Motivation

If we look at the supply chain itself, it's not very different from all the other industries that use networks, like chemical plant optimization or energy management networks. The supply chain is indeed a network of processes that needs to be as efficient as possible to reduce costs and bring more revenue. Simple as that.

Or is it actually that simple? When we look at the network itself, it usually consists of different sections with the end goal of shipping the product to the user. Some of the sections are visible in the image below. A simple supply chain has suppliers, ingredients, intermediate products and final products with shipping information inside the same network.

Since data is arriving from different sources, as each supplier and vendor or shipping company has its own information, it makes sense to pool all that data in a graph database, then use it as a source of truth for connecting all the dots.



It's tough to achieve this with relational databases because of all the table joins necessary to discover all the paths from production start to end. But connecting all the nodes is actually a minor problem. The most time-consuming task is to construct a valid supply chain schedule. To design a satisfactory schedule, experts in the domain must have all the information about the current situation in the supply chain and have answers to questions such as:

What are all ingredients needed for making product X?
What is the produce production order that will make the supply pipeline flow in all the areas at all times?
What critical products mustn't run out of stock to avoid potential losses to the company?
What quantity of ingredient X is necessary to make product Y?
What quantity of ingredient X is necessary to make a sufficient amount of product Y, to make an adequate amount of the final product Z, of which Y is an essential part?

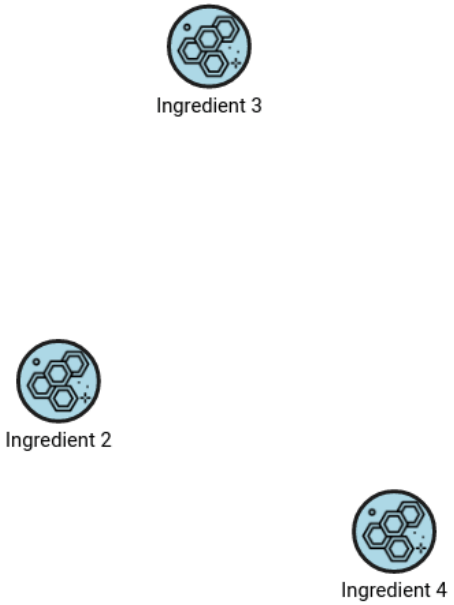
How can graph analytics and Memgraph help answer all of these questions?

Analyze dependant products by using pathfinding algorithms

Every supplier ships its materials to the supply chain company. If the graph is a source of truth, it's easy to check what suppliers provide the company with what materials with the following query:

```
MATCH (s:Supplier {name:"Supplissimus"})-[r:SUPPLIES]->(i:Ingredient)
RETURN i;
```

The query above matches the supplier named "Supplissimus" and returns the ingredients it supplies. With the help of the visualization tool Memgraph Lab and the query execution window, we get the materials provided by the supplier as nodes. The result from Memgraph Lab can be observed in the image below.

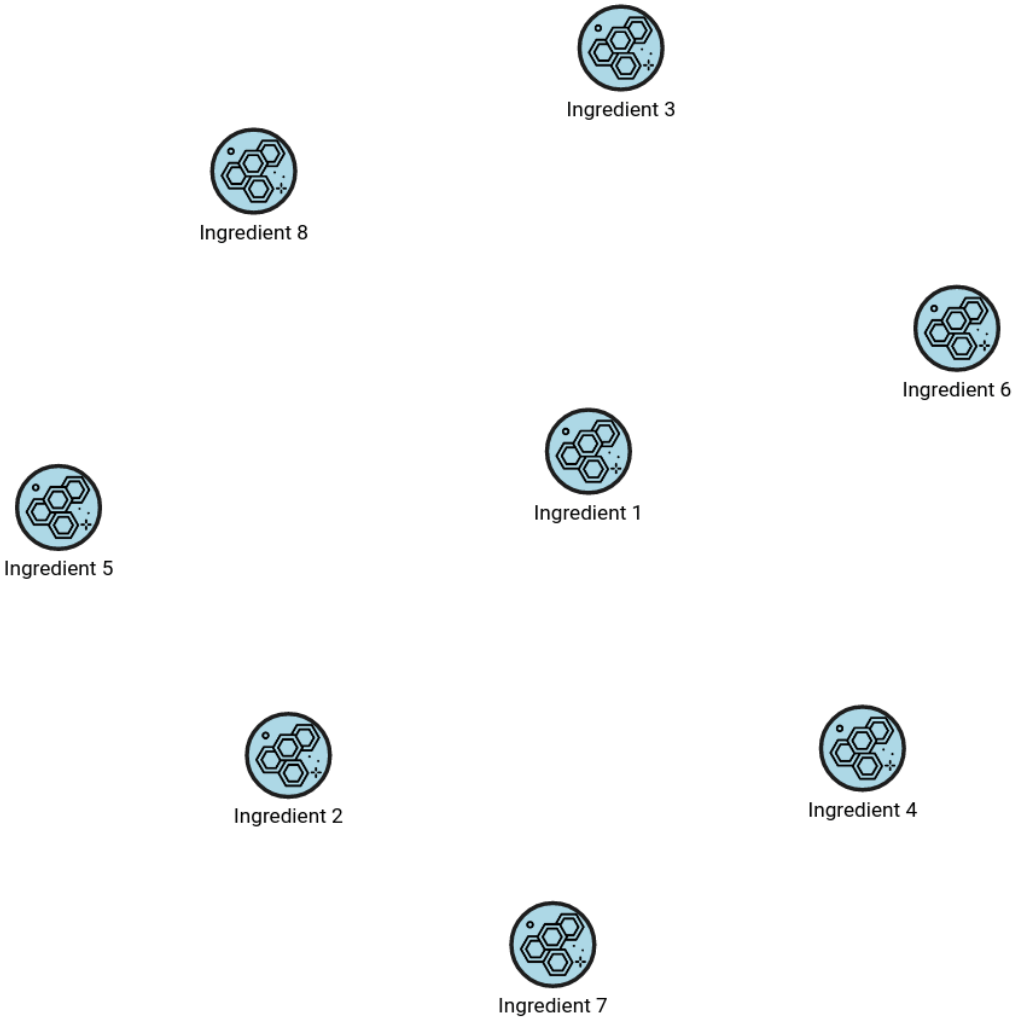


To easily differentiate nodes, each node and relationship type can be styled differently in Memgraph Lab to make visualizations more comprehensible and make different entities in the graph easily distinguishable.

The query above has performed a 1-hop traversal, or querying of nearest neighbors. But since the supply chain is a multi-process network, the goal is to get the information about all primary materials needed to produce the final product. Multi-hop traversals are done with pathfinding algorithms. Several built-in pathfinding algorithms are available in Memgraph, like breadth-first search (BFS), depth-first search (DFS), and weighted shortest path. The BFS algorithm can traverse a graph from the final product all the way down to the primary ingredients needed to produce it:

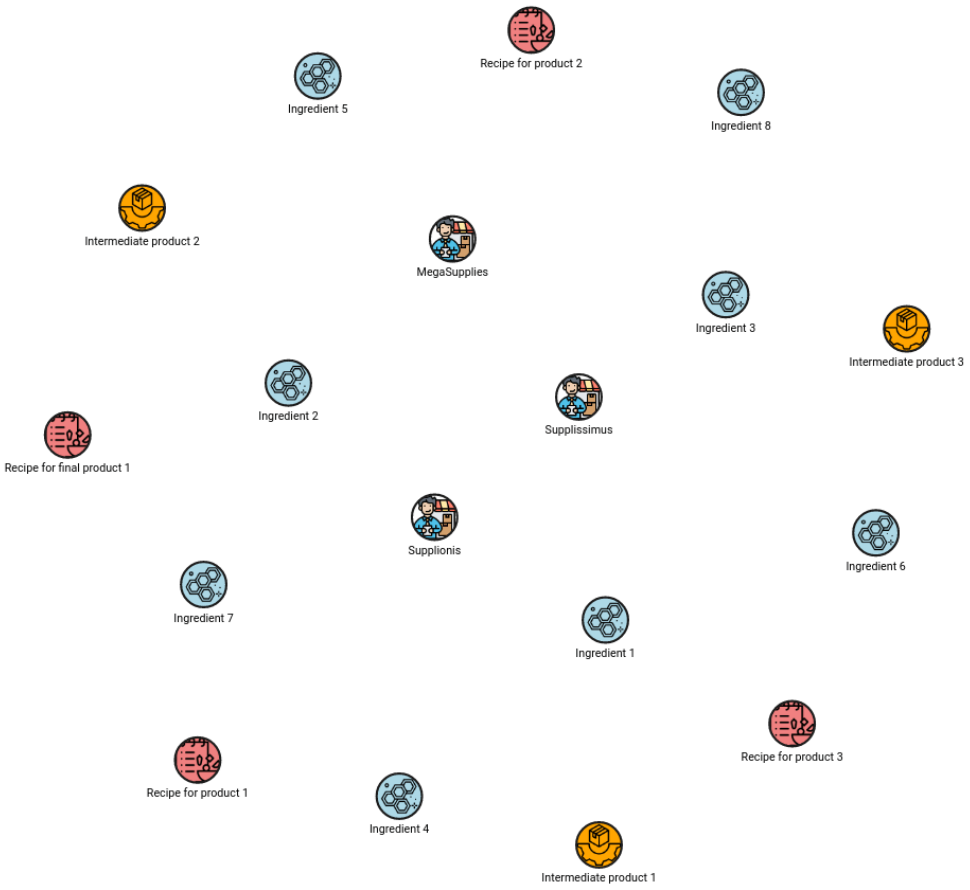
```
MATCH p=(i:Ingredient)-[*bfs]->(f:FinalProduct {id:6});
RETURN DISTINCT i;
```

The above query will return a result like in the image below.



Once the ingredients or materials necessary for product manufacturing have been identified, let's move on to the pipeline analysis. In logistics, it's always helpful to know what processes result in the final product. With the graph_util module, contained in the Memgraph MAGE library, we can find out what nodes and relationships come before the inspected node. The query below returns all the ancestors to the final product:

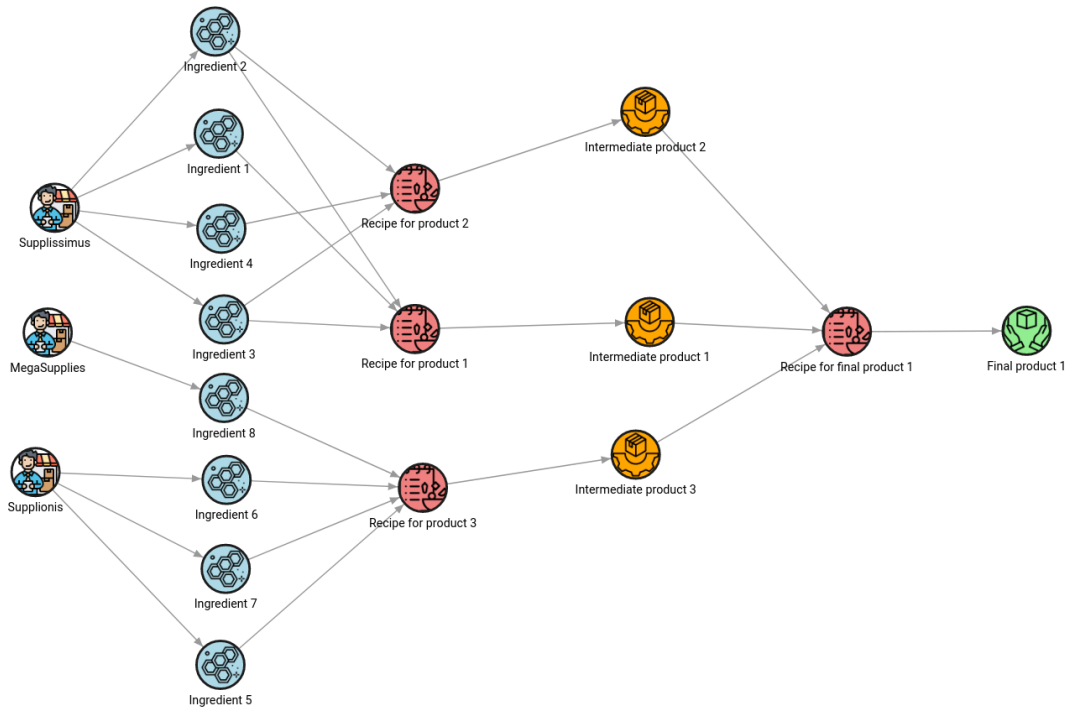
```
MATCH (f:FinalProduct {id:6})
CALL graph_util.ancestors(f) YIELD ancestors
UNWIND ancestors AS ancestor
RETURN ancestor;
```



The result might not look very useful because nodes are scattered on the graph, but this is all part of the learning process! graph_util's procedure connect_nodes connects all the nodes which are input to the final product:

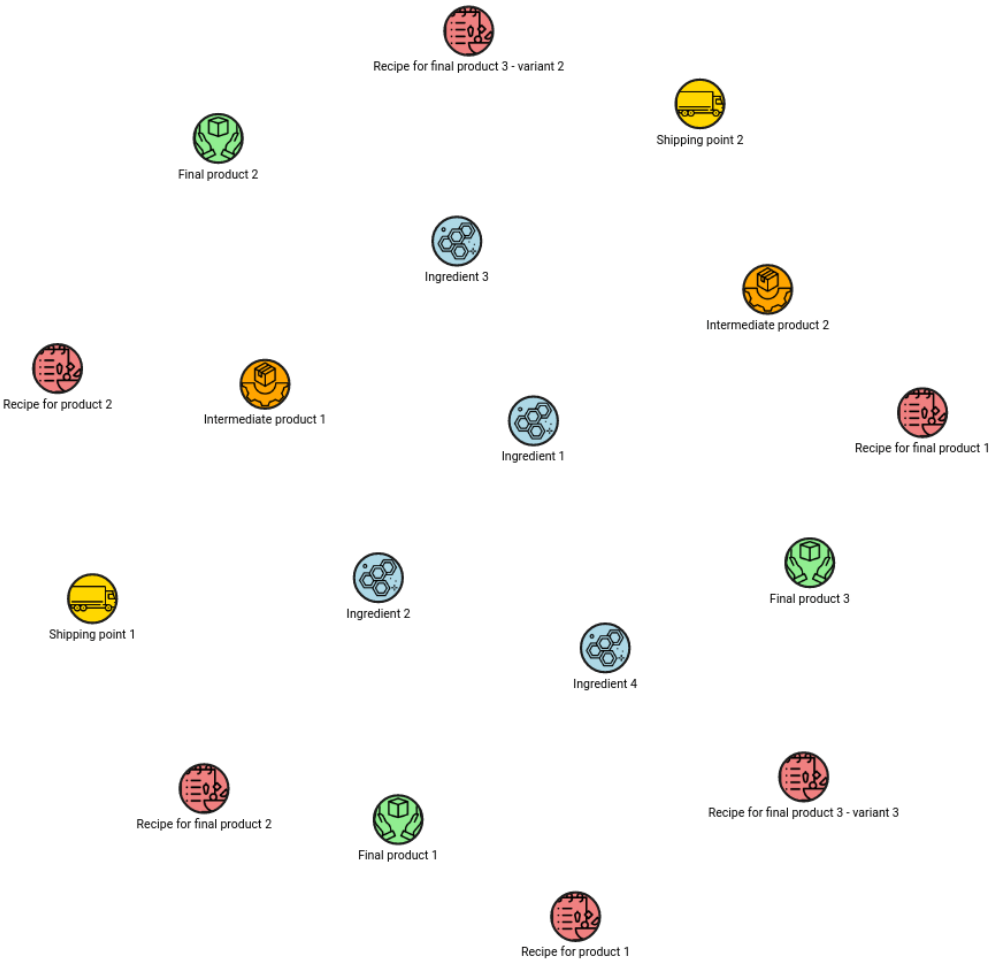
```
MATCH (f:FinalProduct {id:6})
CALL graph_util.ancestors(f) YIELD ancestors
WITH ancestors + [f] AS nodes
CALL graph_util.connect_nodes(nodes) YIELD connections
UNWIND nodes + connections AS graph
RETURN graph;
```

The output of the above query shows all the parameters and steps necessary for constructing the "Final product 1". By looking at the image, you can gather much more helpful information than looking at the tables in a relational database.



The same model can help observe how critical a supplier is to the supply chain. The following query will return all the products depending on the supplies delivered by a specific supplier:

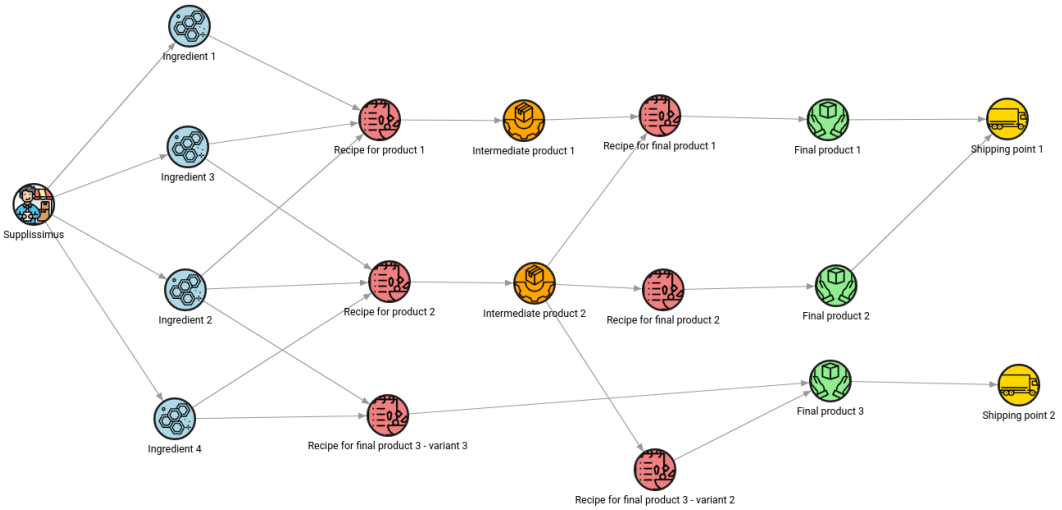
```
MATCH (f:Supplier {name:”Supplissimus”})
CALL graph_util.descendants(f) YIELD descendants
UNWIND descendants AS descendant
RETURN descendant;
```



The graph_util.connect_nodes procedure will show the whole pipeline impacted by the supplier.

```
MATCH (s:Supplier {name:”Supplissimus”})
CALL graph_util.descendants(s) YIELD descendants
WITH descendants + [s] AS nodes
CALL graph_util.connect_nodes(nodes) YIELD connections
UNWIND nodes + connections AS graph
RETURN graph;
```

The following query results show that the Supplissimus supplier impacts the pipeline of all 3 final products. The company depends entirely on this supplier and must have a contingency plan for high demand and production.



Order production by using sorting algorithms

The dependency analysis from the previous chapter made sure all the raw materials were present in the right place at the right time. But before the production can be scheduled, we need to know the order in which each production job is executed to get to the final product. If there is enough raw material in stock, but the production line moves in an illogical order, the schedule will be infeasible or suboptimal. Before the jobs can be ordered, we need to know how much product one job needs to produce so there are enough materials for the next job in the production pipeline.

To solve this conundrum, an algorithm called topological sort needs to be used. Until all the dependencies are produced, there is no way of executing the final job, there are either not enough materials or some other jobs need to be finished for the next job to start. The following query orders jobs from first to the last:

```
MATCH p=(r:Recipe)-[*bfs]->(f:FinalProduct)
WITH project(p) AS graph
CALL graph_util.topological_sort(graph) YIELD sorted_nodes
UNWIND sorted_nodes AS nodes
RETURN nodes.name;
```

Recipes are the connection between materials that form the products. The results in the image below show that the product “easiest” to produce is “Final product 2” since it comes earliest in the pipeline of all the final products in the pipeline.

Data results Graph results unavailable

#	nodes.name
1	Recipe for product 3
2	Intermediate product 3
3	Recipe for product 2
4	Intermediate product 2
5	Recipe for final product 3 - variant 2
6	Recipe for product 4
7	Intermediate product 4
8	Recipe for final product 2
9	Final product 2
10	Recipe for product 1
11	Intermediate product 1
12	Recipe for final product 1
13	Final product 1
14	Recipe for product 5
15	Intermediate product 5
16	Recipe for final product 3 - variant 1

Find critical points in the pipeline with centrality algorithms

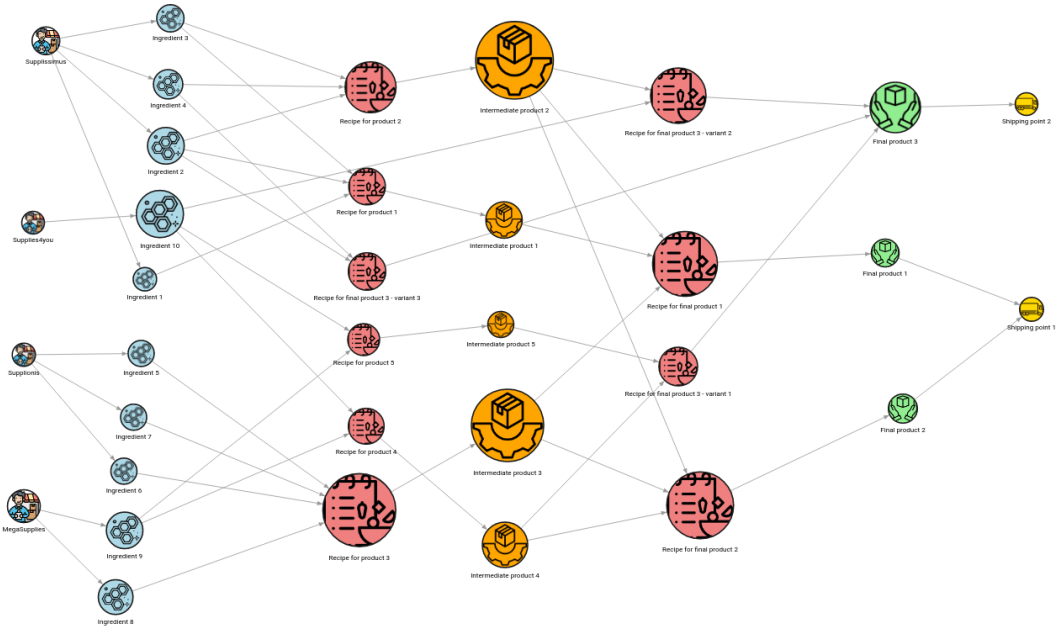
Most of the analysis above was focused on the best-case scenario. If everything goes right, and there is enough raw materials, and the jobs are scheduled correctly, we will get the exact output and maximize our production. However, life is not a fairy tale. There are unexpected events that we cannot control, such as a bad batch of materials or the unavailability of suppliers. The influence of suppliers on the final product has already been analyzed by exploring the connections between nodes. Still, there is one more algorithm that is an excellent tool for impact analysis and what-if scenarios, called betweenness centrality.

To sum up its behavior, the algorithm assigns a greater value to the nodes which lie on as much paths between pairs of all the nodes. If the node lies on a greater number of paths between nodes, it is critical to the pipeline and by having an insufficient quantity of that specific product, we might have a greater number of paths blocked and our pipeline slowed down.

Let's run the algorithm to check for weak points in the pipeline:

```
CALL betweenness_centrality_online.set() YIELD
betweenness_centrality, node
SET node.centrality = betweenness_centrality;
```

```
MATCH (n)-[r]->(m) RETURN n, r, m;
```



The node with the most significant size is “Intermediate product 2”. And if you look at the connections to the final products, we can see that this material affects the production of all three final products. That is why the betweenness centrality algorithm assigned the highest value to that product. This is the product you should never run out of.

The “Intermediate product 3” is also important, as it affects the production of two final products. If any of the three ingredients that form these two products are missing, the company won’t be able to produce them.

By knowing this, you can focus on strengthening the methodology of job scheduling and take necessary precautions to avoid blockages in the production line.

Analyze with custom procedures

As with all solutions in general, there is no silver bullet that works the same for every industry. Graph algorithms may answer many different questions, but it’s impossible to cover all the uncertainties and provide solutions. Each system in the industry is different and has its data models and constraints.

To overcome this issue, Memgraph’s goal was to make graph analytics accessible to anyone. By exposing its API for querying graphs in different programming languages, end users can write their custom analytics on top of graph storage to retrieve

meaningful information. If none of the graph algorithm or analytics capabilities the Memgraph engineering team provides do not suit your use case, develop your own.

Everything inside Memgraph's ecosystem, data persistency, visualization, and analytics are contained in one platform. This reduces the cost of ownership and time spent on menial work trying to connect and maintain a possible alternative of multiple tools and software for a single purpose. The analytics on graph storage also reduces latency and maintenance costs since it's all one platform tied together for the most optimal performance in experience.

Suppose a data scientist is asked how much primary material the company needs to order, to produce a certain quantity of the final product. There is indeed no straightforward generic graph algorithm that can answer that question.

The experts can handle it by developing their own custom query modules which are run inside of Cypher queries. These are simple functions in Python. They define inputs from the query and outputs necessary to continue running the query.

Perform What-if Analysis of Your Network Directly in Storage Without Compromising Data Integrity

Decision-making can be challenging, especially for companies where a wrong decision can impact numerous people, assets, and systems. To ensure that the best option is chosen, companies perform impact analyses or what-if scenarios by analyzing data and weighing the outcomes of different decisions. However, the process is not always easy. Memgraph and Memgraph Lab can be used for effortless and efficient decision-making on networks through impact analysis.

What is the difference between impact analysis and what-if scenarios?

In my experiences with clients, I have often heard the term "impact analysis" from people in leadership positions, whereas data scientists preferred "what-if scenarios". So is there really any difference?

Here's what the literature says about what-if scenarios:

"In scenario planning, a what-if scenario provides a way to consider the effects on a plan, project, or timeline if variability in factors exists. Instead of assuming each part of the process will adhere to some ideal, it involves examining the outcomes if certain things take more or less time, require more or less funding, result in unintended consequences, and so on."

For impact analysis, the definition is as follows:

"An impact analysis (also known as change impact analysis) is a step-by-step process for determining the potential positive and negative consequences of a business decision. A company may also conduct an impact analysis to determine how they would respond to an unplanned disruptive event, such as a loss of data, a breakdown in the supply chain, or a natural disaster".

In my opinion, you won't be far off with any of the terms because one thing is for sure - the goal of both is to predict the outcome of a decision.

Difficulties when dealing with impact analysis in networks

Important events a platform handling impact analysis needs to be able to handle are updates and analytics on the network.

In the chemical industry, data scientists might alter the topology of the chemical plants by adding new reactors to see whether the topology yields better flow. In cloud computing, a cloud engineer might want to alter the number of provisioned resources to see whether the system achieved minimal cost with the same performance. Supply chain companies might add another line of production and see whether it maximizes their production.

Steps different industries take with what-if scenarios are:
Do several updates on the network.
Perform analytics.
If the results are satisfying, continue with the updates or delete updates.
If the results are unsatisfying, continue with the updates or delete updates.

Both updating the graph and deleting the updates are an option because, although the results might be great, the idea can be in its experimental stage, or the staff might want to explore more scenarios with the same network.

The theory of databases tells us that transactions can yield updates with an optional rollback or commit, so pure modifications to the network shouldn't be a problem. The real question is actually, what kind of analytics needs to be performed, and are database systems capable of performing those kinds of analytics?

Database systems nowadays have various integrations with different types of data. Ride-hailing industries might use spatial data and spatial indexes to optimize the positions of taxi drivers in the city. Recording sensor data is a good use case for time series databases. Most relational databases now have functions to work efficiently with text data. However, when it comes to networks, all except graph databases fail to model any kind of efficient traversals or graph analytics in the network.

With insufficient tooling at the database level, there are only two possible workarounds:
Clone the network and do the updates on the clone, then delete it.
Load the network into memory, executing updates and analytics in the code.

Either way, committing changes will need to be done separately and cautiously. The first workaround sounds passable because data remains consistent, but it can double the memory and be a big problem in the case of large networks.

The second solution relies on the performance of data processing libraries, such as NetworkX, igraph, and similar. While some of them are really performant, like igraph, they still need topology to be loaded into the memory and worked with. Not only that, but if the results of the what-if scenario are unsatisfying, there needs to be a way to revert to the previous topology to try again with new modifications, and database systems are not designed to keep the transaction while there is ongoing external processing.

What if you use Memgraph for your what-if scenarios?

If there was only a way to encapsulate storage and data processing. It would ease life dramatically to perform updates in the same transaction.

Fortunately, Memgraph offers a concept called query modules - execution of complex analytics procedures inside the query. Let's look at the example below

```
MATCH (n)
CALL my_awesome_module.my_awesome_procedure(n)
YIELD result
RETURN result;
```

This query also executes a procedure containing programmatic code, which allows for better expressibility directly on the database level. From simple aggregations to complex graph algorithms like community detection, everything that you can't express with Cypher is implementable as a query module.

Furthermore, Memgraph developed Memgraph MAGE, a graph algorithm library with all the typical, streaming and even machine learning algorithms on graphs. Analytics like PageRank or centrality algorithms no longer need to be implemented inside application code, as they're all contained in one platform - Memgraph.

If the algorithm you need is not in the library or it's too custom to implement - you can always write your own. Memgraph supports C, C++, Python and Rust as possible interfaces for writing query modules and developing custom analytics.

How do query modules change the what-if scenario workflow? Let's look at the query below:

```
BEGIN; // start of the transaction

// update 1
// update 2

CALL my_awesome_module.my_awesome_algorithm(...)
YIELD result
RETURN result;

ROLLBACK; // reverting updates
```

The above flow ensures that the graph isn't updated due to the experimental phase of testing the network topology, but it's still yielding useful analytics. If the outcomes are satisfactory, the updates can be applied to the network by using COMMIT instead of ROLLBACK.

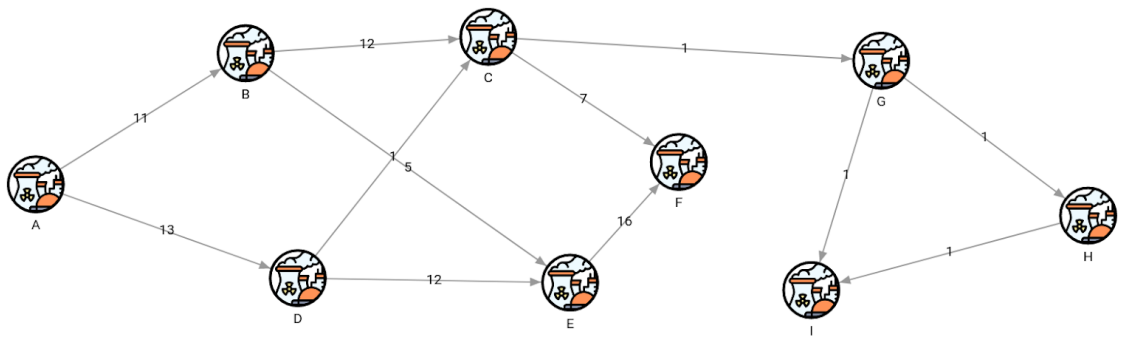
This workflow solves several issues:
The graph is updated without committing the changes.
Custom analytics are executed to show decision outcomes.
Changes can be committed to update the graph.
Changes can be rolled back.
Data remained consistent at all times
Risks are effectively minimized by running trial-and-error experiments without corrupting data

Let's look at a practical example of what-if scenario implementation.

Showcase example with Memgraph Lab

The following example deals with a fictional chemical plant. The goal is to optimize the topology to yield maximum flow through the plant. All queries were run in Memgraph's visualization tool, Memgraph Lab, which offers custom styling for maximum visibility over the network. The following query retrieves the initial topology of the chemical plant.

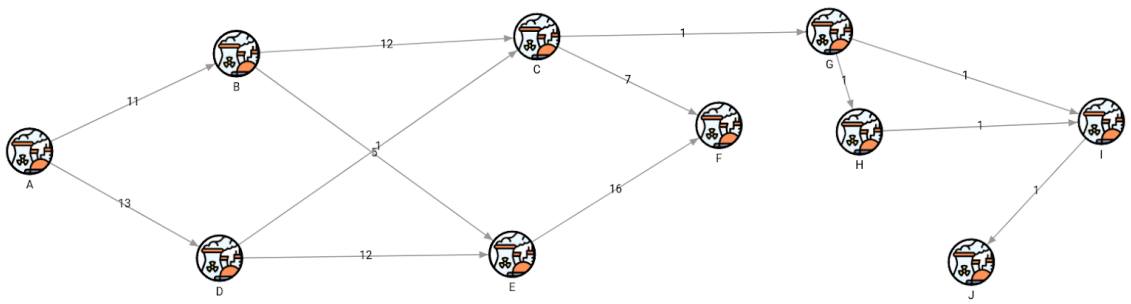
```
MATCH (n)-[r]->(m) RETURN n, r, m;
```



Each chemical plant component is named using a letter of the alphabet, and the relationships connecting the components symbolize pipes allowing a certain amount of flow displayed as numbers on the picture.

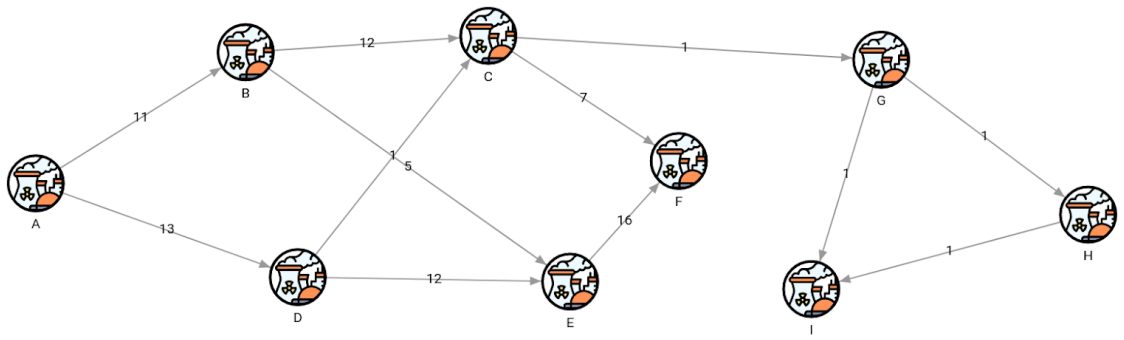
The first transactional query will update the graph by adding an additional plant component and show how the topology changed, then roll back the changes, so the change isn't actually visible in the data.

```
BEGIN;

MATCH (i:I) MERGE (i)-[:CONNECTED_TO {flow: 1}]->(:J);
MATCH (n)-[r]->(m) RETURN n, r, m;

ROLLBACK;
```



Because changes were rolled back, the original topology stays the same as in its initial state.

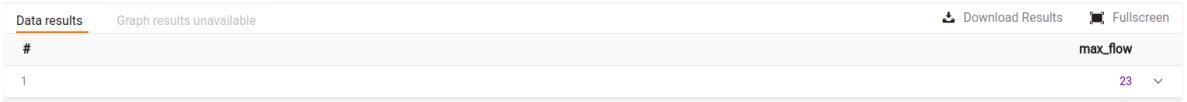
```
MATCH (n)-[r]->(m) RETURN n, r, m;
```



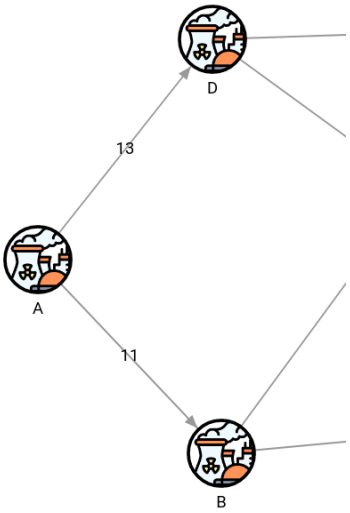
And that's how queries using transactions operate. Let's now do an impact analysis or what-if scenario.

In the following example, we will analyze flows from one point in the network for another. One of the more familiar graph algorithms used to calculate the flows is called maxflow. The maxflow algorithm in the query below will return the maximum flow of resources from component A to component F.

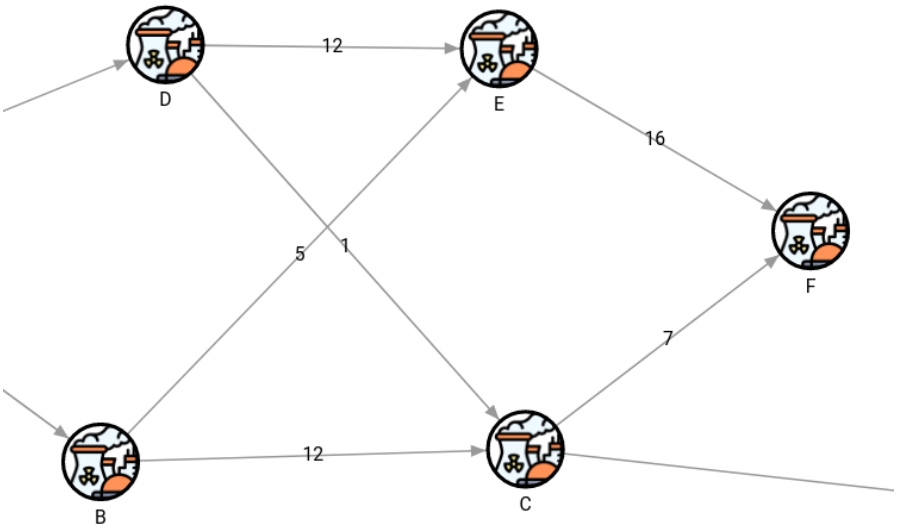
```
MATCH (a:A), (f:F)
CALL max_flow.get_flow(a, f, “flow”) YIELD max_flow
RETURN max_flow;
```



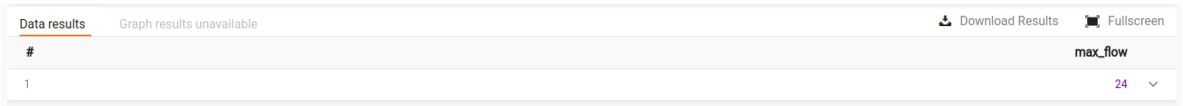
Let's look at the source of the network, which is the component A. It has 2 outgoing flows of 13 and 11, which would ideally total for a maximum flow of 24.



The algorithm calculated that the current maximum flow is 23. Surely, we can do a better job of increasing the flow to 24. However, it is not a straightforward job and might require trial and error until we get it right. After inspecting the network, we have a potential solution to the problem by doing an upgrade on the relationship between components E and F. We will increase the flow between those nodes by 1, so the total input to node F equals 24 (17 from node E + 7 from node C).



```
BEGIN;

MATCH (e:E)-[r:CONNECTED_TO]->(f:F)
SET r.flow = 17;

MATCH (a:A), (f:F)
CALL max_flow.get_flow(a, f, “flow”) YIELD max_flow
RETURN max_flow;

ROLLBACK;
```



Once the flow between components E and F is upgraded, the maximum flow increased to 24, which is the full flow from the source! Because this upgrade solved the problem, the query is run again, but this time the update is committed and the stakeholders are informed about the positive impact on this part of the plant.

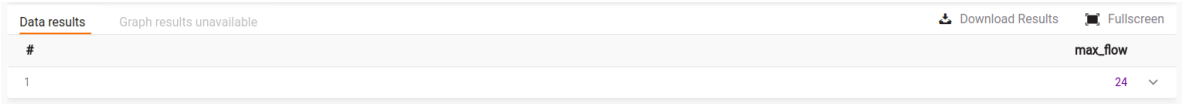
```
BEGIN;

MATCH (e:E)-[r:CONNECTED_TO]->(f:F)
SET r.flow = 17;

MATCH (a:A), (f:F)
CALL max_flow.get_flow(a, f, “flow”) YIELD max_flow
RETURN max_flow;

COMMIT;
```

Let’s upgrade the relationship between components `A` and `D` to maximize the flow yet again.

```
BEGIN;
```

```
MATCH (a:A)-[r:CONNECTED_TO]->(d:D)
SET r.flow = 20;

MATCH (a:A), (f:F)
CALL max_flow.get_flow(a, f, “flow”) YIELD max_flow
RETURN max_flow;

ROLLBACK;
```

However, the flow remains the same. So this upgrade did not actually yield better results, and there is no reason to commit it as different or more updates in the network are necessary to maximize the flow.



After noting the trial and rolling back the topology, the data is ready for more examples and experiments until we get a successful topology, and then actually update the plant on the field.

And that’s basically it! With a few commands, a little bit of knowledge about database transactions, and storage coupled with processing power, we are able to do complex impact analytics and what-if scenarios without any fear of losing or corrupting the data.

Conclusion

In conclusion, graph databases like Memgraph offer a comprehensive solution for network resource analytics. Memgraph's advanced graph analytics library (MAGE) and visualization tool (Memgraph Lab) provide real-time insights and custom visualizations. Memgraph can help with analyzing highly connected data, reducing risks and optimizing the supply chain, telecommunications, and cloud computing industries.

By integrating analytics with graph storage, Memgraph minimizes the risks of quick and efficient network analysis. You can explore the capabilities of Memgraph further by trying out datasets in Memgraph Playground and sharing your use cases on the

Memgraph Discord server. If you have any further questions, feel free to reach out to the Memgraph community for help in optimizing and managing your network.