MEM GRAPH

Energy management

WHITEPAPAPER

Graph Technology in Energy Management Systems

Table of Contents

Introduction

Graph databases are a smart choice for dealing with energy management systems. Some key features that explain their superiority when talking about energy management systems are performance, scalability, visualizations and analytics. However, none of these things matter if you can't answer positively to the single most important question when choosing a technology - is it solving my use case? And they do!

Memgraph offers a full-fledged solution for dealing with problems in energy management systems. Read on to find out how fast computations, visualizations, extensions to analytics and additional features that enable real-time processing can be used to manage networks no matter how complex or dynamic the system is.

How Can Companies Meet Energy Management Demands in the New Era - A Graph Approach

As blockchain and machine learning were the buzzwords 10 years ago, it seems that energy will be the buzzword of this decade. As the most powerful asset in the upcoming years, it is a surprise that handling and managing power remains rather poor.

Let's dive into the reasons why the current analytical tools are no longer performant enough and introduce a graph data approach that solves problems regarding performance, analytics, and visualizations of energy systems and topologies.

Motivation

With the rise in size and complexity of energy system topologies, it's fair to ask: "Can the underlying, supposedly state-of-the-art tools and systems handle the amount of processing power and data increasing every year?" We need to look at the architecture beneath those tools to answer this question.

The base is a network of interconnected components representing various plants, energy power grids, gas pipelines, etc. Each component inside the network can have

one or many incoming, outgoing, or bidirectional links to other components, which indicate the transport of resources, such as oil, gas, electrical current, etc.

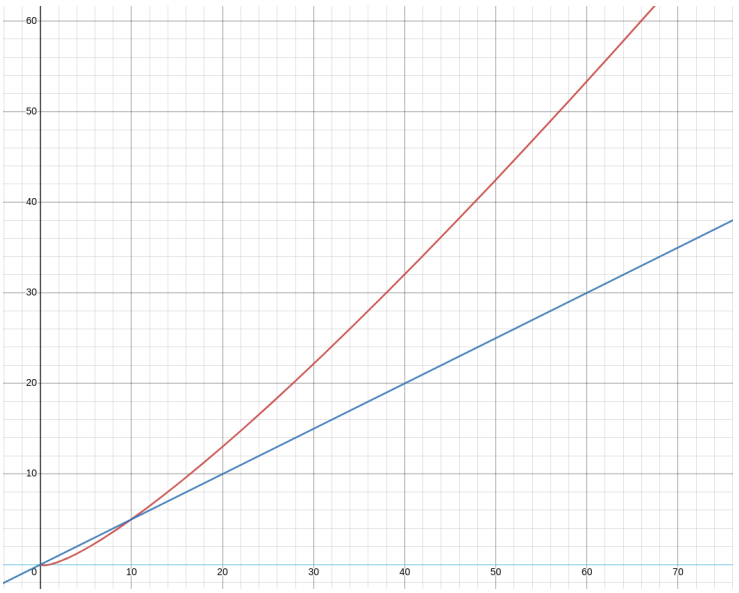
The end goal of the tools is to provide additional insights besides the raw topology of the network. Additional insights can be gained from analyzing power flow calculations, balancing supply and demand at different locations, estimating the state of single components, and many more. If the goal is to have a static topology, the current tools work as expected, with the occasional exception of doing some time-exhausting mathematical calculations.

Problems arise when topology needs to be dynamic or when the topology size increases from the topologies of cities to the topologies of countries or even continents. If we look at Europe's topology, the data's size and complexity quickly rise because every country has different demands, expectations, or constraints. At this point, the tools which were once passable become slow and unable to perform analytics at the required pace.

This decrease in performance stems from the fact that data is traditionally stored in relational databases. Relational databases have been around since the 80s, and before the age of big data, they were the most rational choice when developing applications and analytical tools. The data size was, to a great extent, smaller than today, and computer hardware was improving vertically to support computing operations and improve processing time, which was the biggest bottleneck at the time.

Over time, data complexity has increased so much that relational databases are becoming increasingly impractical for many use cases, including energy management use cases. This is largely due to the usage of JOINs and recursions, which are essential in modeling a topology in a row-based manner.

To display a topology, all the connected components, usually listed as pairs in a many-to-many foreign key table, must be joined by a complex Cypher recursion. 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)). The results in the image below imply an extensive complexity considering the only goal is to list the whole topology without doing any analytics. The red line, representing the time complexity of JOINs, is clearly rising steeper than the linear complexity (blue line).



This alone proves that tabular data is not optimal for analytics and queries that look for patterns in the topology. Analysis of tabular data is well-suited for aggregations and mathematical operations. However, when it comes to impact analysis, root cause analysis, or simulating new scenarios, row-based calculations perform poorly at best. They are unable to perform traversals and detect weak points which could lead to power outages (impact analysis) or analyze what has actually led to a power outage (root cause analysis). The most recent context where we could use a tool that performs well is to search for alternative topology compositions in political turbulences affecting energy pipelines.

What's worrisome is that even though these flaws are common knowledge and can lead to devastating consequences, traditional software tends to outlive the test of time. They provide information about flows, supply, and demand but are becoming outdated due to the deprecated implementations for the use case. All the work on dynamic systems with a high amount of updates per second is left to the staff, and the tools they are provided with complicate their lives instead of making them easier.

The good news is that there is a technology we can turn to in order to solve these issues, and that is graph-based data representation.

Enter graphs

The problems regarding the tabular representation of topologies have been addressed with the arrival of new types of NoSQL databases, namely graph databases, which are able to store networks of data.

Networks consist of nodes (also called vertices) and relationships (also called edges) and are the first-class citizen of the graph database. This approach was constructed specifically to address the problems of JOINs in relational databases. Still, as we will see in the next few sections, graph databases didn't only solve the issue of complex JOINs, but they also pave the way for graph analytics and network visualizations as well.

In a graph database, all the data is already joined the moment it arrives in the database, and relationships represent pairs of connected nodes. The result of having all the data connected immediately enables access to joined elements in constant time (O(1)). Not only that, but it also abolishes the need for foreign keys, as checking around multiple tables to search for pairs is no longer required. Everything is already paired and connected.

Here is another explanation of how this approach would work in real life. Consider that a component in your power plant is a node, and you're looking to see what is the closest component the power plant can be connected to for energy transfer. You would probably immediately think of the neighboring power component since you know your power plant's outline. There shouldn't be any friction or complexity on the path to finding that the neighboring power component is the answer to your question. That would correspond to a graph-like approach to matching data.

In the same scenario with relational databases, you need to look at the company's administrative center (foreign-key pair table) to find the location of the power component and realize it's the neighboring component. A conclusion you couldn't make without having the information about what components are joined first. To visualize a whole topology of neighboring components with relational data, the JOINs become a problem, not a solution.

Let's dive deep into how graph data representation can benefit the energy management use case.

Representation of the topology feels most natural in a graph

The advantage of being able to search neighboring nodes in constant time has already been explained in the chapter above. Let's expand this concept to the whole energy system topology scale. Under the hood, a graph is modeled with adjacency lists optimized for expanding the graph to inbound or outbound nodes. Adjacency lists

correspond to skip lists in the implementations using a relational database. To load the whole topology would mean using a graph traversal algorithm such as breadth-first search (BFS) or depth-first search (DFS), which are quite common traversing algorithms, to discover and visualize all the nodes and their relationships.

The time complexity of these algorithms is O(V + E) if adjacency lists are used, which increases search performance drastically as we reduce complexity to a linear scale. Furthermore, if more components are added to the topology, the time spent traversing the network is less affected than in relational databases.

Another reason for improving the storage implementation is the need for dynamic systems analysis. Frequent updates of component states in the network require an efficient way of preparing the data for processing (mostly loading data into memory). In relational database systems, an average of 25 - 35% of the query processing time is spent on joining tables required for estimating flow and component states in energy networks. We can safely assume that cutting the calculation process by ~30% would greatly improve the efficiency of people using energy management tools.

A new approach to analytics

With the new way of representing highly connected data, searching for patterns and insights within the dataset also changes. SQL databases were designed to do transactional processing mostly, and back in the day, powerful analytical tools weren't really needed. With the need for big data analytics, new database products moved away from row-based table storage to column-based ones, called columnar storage. Columnar storages are better at analytical queries but still cannot be efficient enough to analyze connected systems because they were not actually designed for that purpose.

But both the row and column-based databases used SQL, which cannot easily express relationships between objects in graph databases. Hence, with the introduction of graph databases, new languages were developed, such as Cypher (Memgraph, Neo4j), Gremlin (Azure Cosmos DB), GSQL (TigerGraph), and others. They were designed with the main idea that patterns in the database are easily searchable by a simple pattern corresponding to a connection between the 2 nodes in the graph.

Searching through a whole electrical grid system should be as easy as thinking about it.

Examine the Cypher query below:

```
MATCH (n1:Generator)-[r:IS_CONNECTED_TO]->(n2:Generator) RETURN n1,
n2;
```

Without knowing any Cypher commands or syntax before reading this query, you can probably conclude it will return all the pairs of directly connected generators in the graph.

We can expand this query by traversing the graph via different paths. The query below will find all the paths from one node to another by traversing a subgraph and returning its corresponding nodes and relationships.

```
MATCH p=(n1:Generator)-[r:IS_CONNECTED_TO *bfs]->(n2:Generator)
RETURN nodes(p), relationships(p);
```

These queries are the simplest information we can get from a graph representing a topology. But most graph algorithms are based on this simple principle of traversing the graph and analyzing incoming and outgoing relationships, such as BFS, DFS, PageRank, Community Detection, and Centrality algorithms implemented in most graph databases. Graph traversals are at the heart of an analytical world unimaginable in relational databases.

But in the end, it's not the algorithms that bring the most value but the solutions they provide for the problems in the domain. Impact and root cause analysis in case of a power outage, simulations of what-if scenarios, and flow analysis of gas pipelines can all be researched with traversals and graph algorithms since they are the perfect tools for problems in highly connected systems.

To expand on the previous thought, searching through a whole electrical grid system and analyzing patterns and gaining insights should be as easy as thinking about it!

All-in-one tool for your intelligence

As graph databases were sparsely used as a storage and analytical layer, graph database providers focused on improving processing power and exposing different insights when data is represented as a graph.

That is why graph databases are not only storage but platforms that frequently include an arsenal of graph algorithms, traversing capabilities, and visualization tools.

The whole idea of a centralized system that handles storage, computations, and visualizations is extremely important for energy management systems since alerts, state updates, and additional information on the topology can be displayed ad-hoc in one place without using a number of clunky underperforming tools that need to be somehow wired together. Additionally, it speeds up the analytics pipeline since the storage and computation power is located in one place and doesn't need to be transported to another platform.

Solving the Most Common Problems in Energy Management Systems With Graph Analytics

Graph databases are a good choice when dealing with highly connected data in energy management systems. Graph databases can handle energy management systems' needs for performance, scalability, visualizations, and analytics.

But they also solve problems that the energy management systems most often encounter. Here is a list of key questions and problems in the energy network and answers to how to graph analytics and algorithms can help detect, prevent and solve them.

Motivation

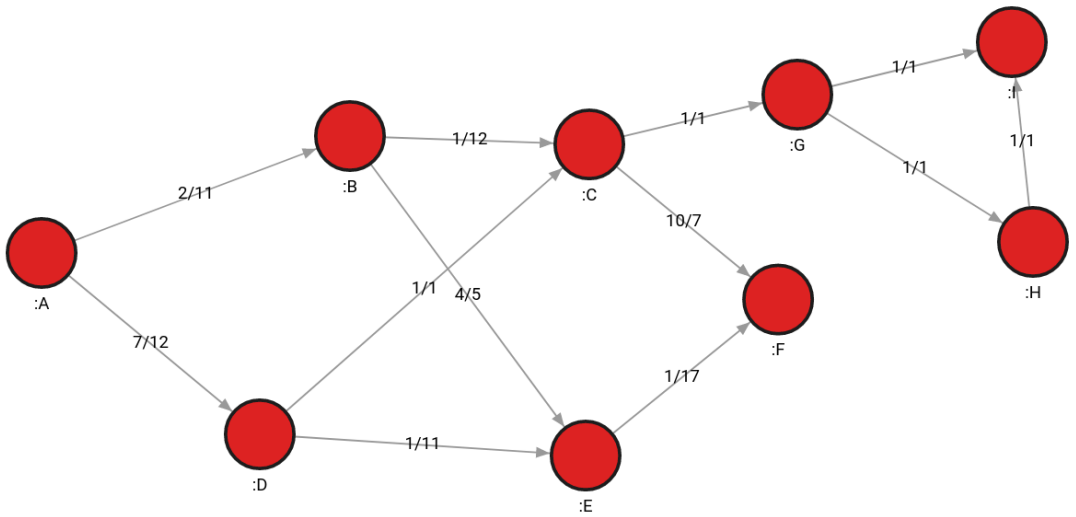
When dealing with an energy topology, many questions arise that typically don't exactly correlate with the storage implementation and the query language of relational databases. People in the energy management industry want to find bottlenecks, and hubs, analyze the flow in the network and find paths between

different nodes. At the same time, a relational database is only good enough for statistics and aggregate functions. And if your primary use case is aggregating data points and finding insights based on numerical analysis, a relational database might be a good decision. However, a graph database is structure-oriented and can answer those truly important questions by traversing the graph to see if certain neighboring nodes fit into different groups, check the connectivity strength among the nodes, etc.

Due to that difference in asking about the structure of entities, rather than only information they hold, graph databases have made Cypher query language to traverse and find patterns and paths along the graph. In the next few sections, we will incorporate some of the basic questions and problems from the energy management systems and show how to graph traversal queries written in Cypher query language, Memgraph's graph algorithms, and data visualizations in Memgraph Lab might benefit the company and add value in solving these problems.

Example graph

For the purpose of demonstrating traversals and graph algorithms, we will use a simple energy system graph in the image below. The graph below consists of 9 nodes and 12 relationships between them. Additionally, they have the weight and flow properties on each relationship, separated by a dash. Weight and flow properties are used for calculating of shortest paths and flows.



For example, the weight of the relationship between nodes A and B is 2, and the flow is 11.

Finding the path between 2 nodes

Traversing the graph is one of the key benefits of graph databases with a linear time complexity that grows with the number of nodes in the graph. For that purpose, many graph databases leverage graph algorithms already existing in the mathematical world.

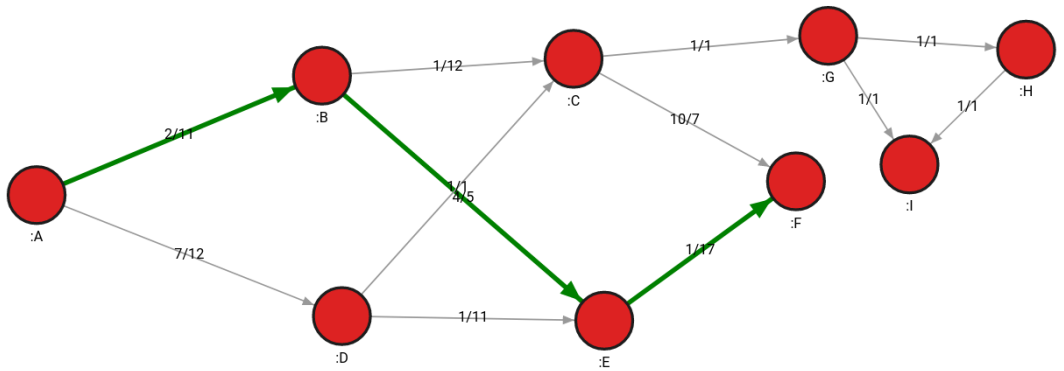
The most common problem concerning pathfinding is how to get the power, or oil, from point A to point B. Most of the time, the goal is to find the shortest distance, but it can also be about using the minimum number of components or avoiding certain country borders. These paths can be found on static data or dynamical data that often changes. The system needs to find a path quickly and be ready to calculate alternative paths without lag.

In relational databases, designing algorithms for finding shortest paths, such as Dijkstra's algorithm for weighted shortest paths, is extremely difficult. However, most graph databases, including Memgraph, include predeveloped pathfinding algorithms that are accessible as a function that is called within the query.

Here is an example of Dijkstra's algorithm called within a Cypher query:

```
MATCH p=(n:A)-[r *wShortest (r, n | r.weight)]->(m:F) RETURN p;
```

The query takes the source node (A) and the target node (F), and returns both the shortest path (the minimum number of hops between nodes) and the least weighted path.



Although both the (A, B, E, F) and (A, D, E, F) paths are the shortest paths, (A, B, E, F) path also weighs less (for example, requires fewer resources). It has a weight of 7, while the (A, D, E, F) path has a weight of 9.

Say that the pipeline between E and F nodes is blocked, or had a power outage, consequently a raise in the relationship weight by a lot. By rerunning the algorithm, a new path would be found consisting of nodes A, B, C, and F with a total weight of 13, thus always using the minimum amount of resources.

In another graph network, it would be beneficial to have all the possible alternative paths between 2 nodes to ensure there are enough viable options for transporting energy. In that case, the ALLSHORTEST algorithm is the most useful one:

```
MATCH p=(n:A)-[r *ALLSHORTEST (r, n | r.weight)]->(m:F) RETURN p;
```

Finding weak links and high-risk nodes

When there is a power outage on the field, well-organized companies immediately start fixing the damage, but some households or companies can still be left without power for a considerable amount of time. Sometimes just reacting to damage is not a long-term solution. Companies should investigate and predict the behavior of their network and support it with additional or alternative power lines to make the flow of energy consistent and unaffected. Analyzing the network, business, decisions, and investments and deciding upon contingency plans or improvements is called risk analytics, and its main goal is to prevent events that lead to unexpected losses.

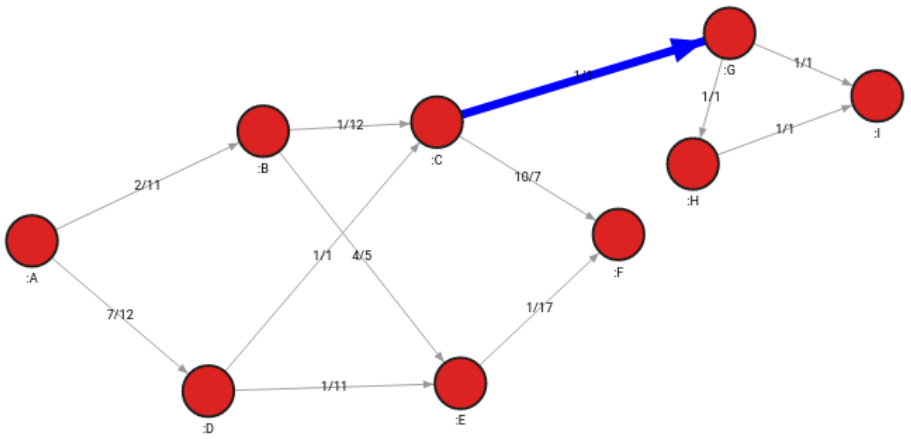
Graph theory refers to that relationship as a bridge when one group of nodes is connected to an energy source by only one connection. It implies that the graph would be divided into 2 separate networks if the path becomes unusable, causing damage by interrupting the flow of energy going into the second component.

To identify bridges in Memgraph, MAGE library includes a special procedure bridges.get():

```
CALL bridges.get()
YIELD node_from, node_to
```

```
MATCH (node_from)-[r]->(node_to)
SET r.bridge = True;
```

The query identifies the bridge and connects the nodes with an additional relationship to signify that the relationship between the 2 nodes is indeed a bridge.



After inspecting the graph again, we can see that the top part is poorly connected to the rest of the graph, and by disconnecting the relationship, we will get 2 separate components. This might follow a decision to strengthen the poorly connected part and decrease the risk of a power outage.

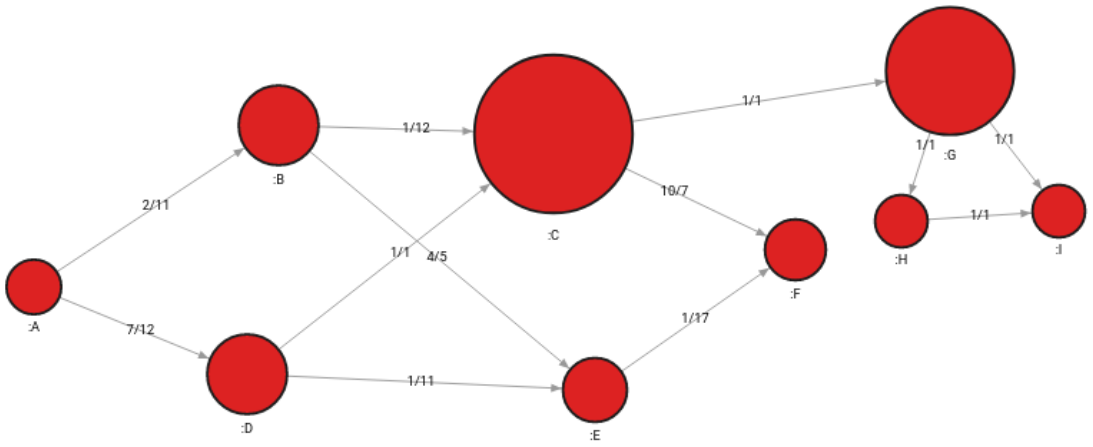
Another algorithm that can be used to analyze risks and help identify weak points in the topology is betweenness centrality. It measures the extent to which a certain node lies on paths with other nodes. Basically, if a node is connecting more nodes, its betweenness centrality rating is rising. Cutting out nodes with high betweenness centrality might make your network connectivity sparse or possibly disconnect parts of your graph.

Based on the example graph, we can expect that the nodes with the bridge relationship have the highest centrality, since the nodes connected with that bridge have a high risk of being disconnected. Indeed, that is what the query below will show:

```
CALL betweenness_centrality_online.set()
YIELD node, betweenness_centrality
```

```
SET node.betweenness_centrality = betweenness_centrality
RETURN node, betweenness_centrality;
```

Node G got a centrality rating of 0.428 and node C got a centrality rating of 0.6, while for example, node F got a centrality of 0.047, meaning it's not of high risk to the topology.



How risk analysis parameters change upon network update

The previous examples showed algorithms that can point out weak points in the topology and urge network support. If the graph is fairly large, it's not always feasible to recalculate all the algorithms from scratch when data is updated. Some updates are minutely and concern only several nodes, so the whole graph doesn't need to be recalculated every minute.

Fortunately, a new branch of graph theory involves dynamic graph algorithms, which can calculate the correct outcomes of the traditional graph algorithms based on the local change in the network without recomputing the whole graph.

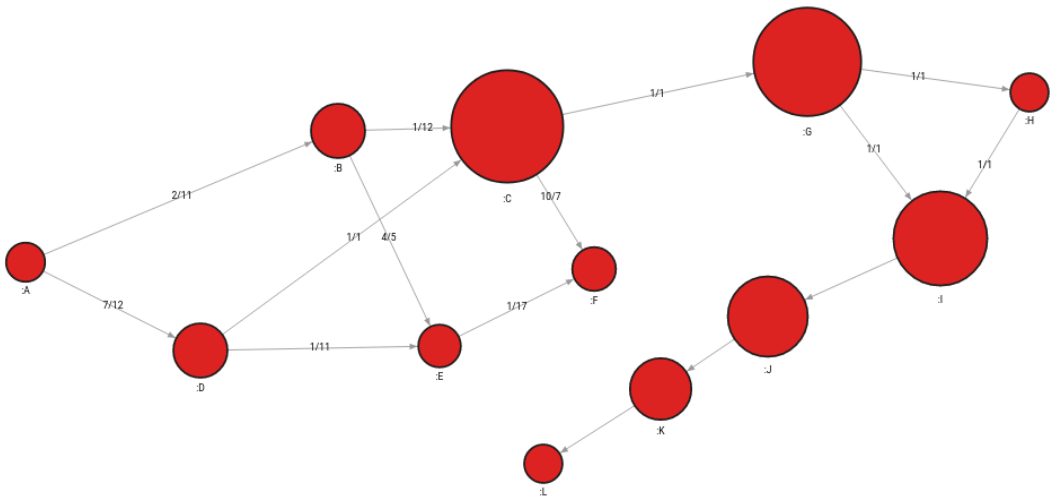
One of the implementations in Memgraph involves dynamic betweenness centrality, so let's see how it works.

First, a trigger needs to be created, which will dynamically update betweenness centrality after a node or relationship is created or deleted:

```
CREATE TRIGGER update_bc_trigger
BEFORE COMMIT EXECUTE
  CALL betweenness_centrality_online.update(createdVertices,
createdEdges, deletedVertices, deletedEdges)
  YIELD betweenness_centrality, node
  SET node.betweenness_centrality = betweenness_centrality;
```

Now, let's modify a graph a bit by adding a few nodes to the secondary component, thus making the risk on nodes G and C even higher:

```
MATCH (n:I)
MERGE (n)-[:PATH]->(:J)-[:PATH]->(:K)-[:PATH]->(:L)
```



The updated betweenness centrality of node C is 0.579, and that of node G is 0.545, which means they are now of even greater importance and risk to the topology. These nodes need further support with additional paths between components. Although node C's values have decreased a bit, that of node G has risen, and those 2 nodes remain to be holding most of the topology together.

Finding points of disconnection with impact analysis

Prevention seems great as a concept, but it's human to err, so monitoring the current situation and looking for problems is vital. Power outages will occur, and although the topology has been improved after a risk analysis, there is always a possibility that one day all hell breaks loose and everything goes dark.

An algorithm such as weakly connected components might come in handy to help prevent such scenarios. The algorithm analyzes a number of disjoint components in the whole network. If there is an increase in the number of disjoint components compared to the result of the previous query, it might be a good time to sound the alarm because one part of the graph just got detached.

By running the following query on the graph in the image below, the result should be 1 since the graph is fully connected:

```
CALL weakly_connected_components.get()
YIELD component_id, node
SET node.component_id = component_id
return component_id, count(*);
```

If nodes C and G become disconnected and the algorithm is rerun, the result will increase to 2 to indicate there are now two disjoint components since the bridge was removed from the graph.

This is a good example of how graph databases utilize the linear time complexity concerning the size of the graph with an algorithm, while a relational database might spin in neverending loops to answer a simple question like - is my graph connected?

Algorithms like weakly connected components can also be run in impact analysis using the so-called "what-if scenarios." Parts of the network are connected and disconnected to find out what will happen with the network and whether it will stay stable.

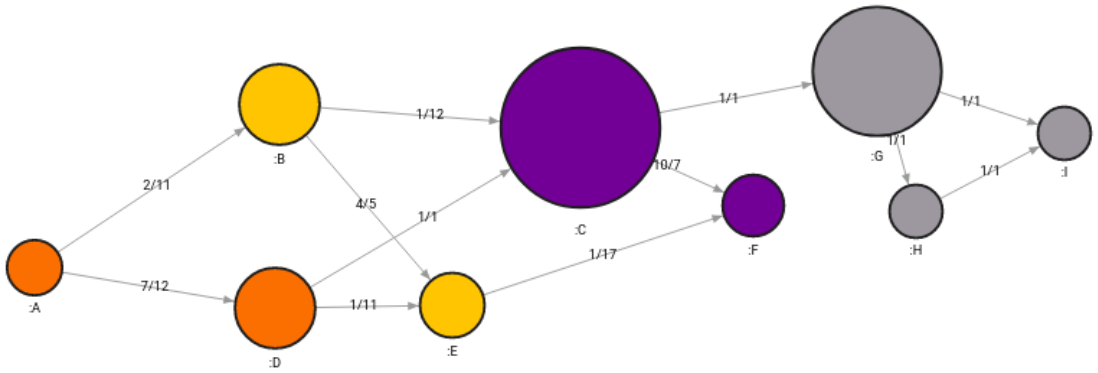
Finding well-connected components

The motivation behind finding out what components in the topology are strongly connected together is to make assumptions and analyze the borders of such components. By directly investigating the cross-component relationships, it can be decided if all components are connected well.

An ideal algorithm for that analysis would be community detection, which groups nodes together and assigns them with a community id. Although the algorithm is mostly used to infer hidden links between nodes of the same community, it can also be used to discover links across communities.

The query below will detect all communities in the graph:

```
CALL community_detection.get()
YIELD node, community_id
SET node.community_id = community_id;
```



The algorithm detected 4 communities which can then be analyzed independently or collapsed together and observed for connections between communities. On a bigger scale, when dealing with the management of oil pipelines across Europe, for example, we can assume that each community represents the network of a specific country and that borders would mark links between communities.

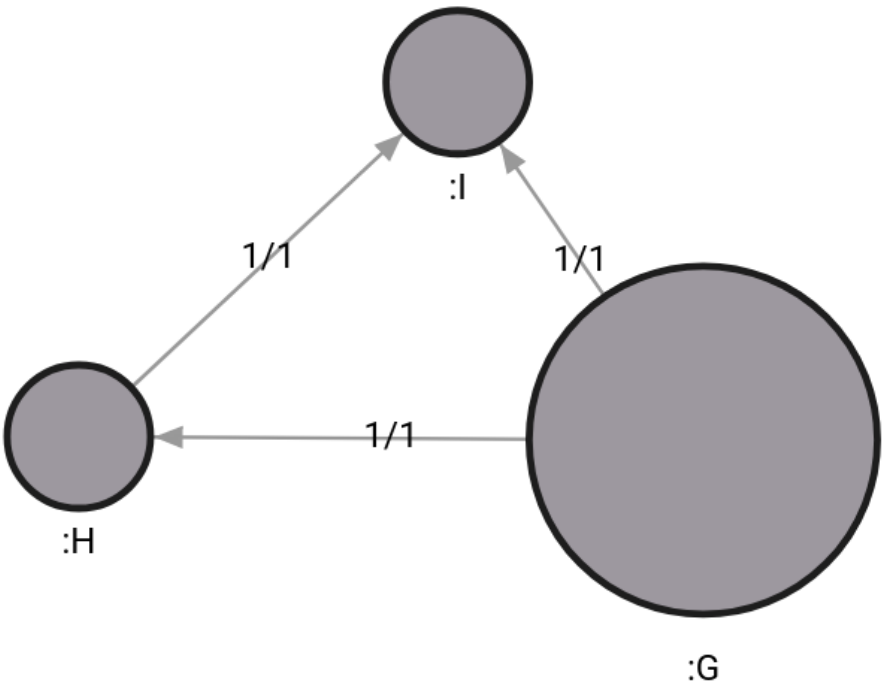
Analyzing separate components of the graph

The whole energy system might be represented with a single dataset, including the entire network topology creating a large graph. However, every data scientist or analyst might be accountable for a different part of the graph, making sure that each subcomponent of the graph (also called subgraph) is well designed and has passed all the criteria. Until now, we have only analyzed graphs as a whole, and algorithms calculated results considering all the nodes and relationships contained in the database.

However, graph projections in Memgraph enable subgraph analysis. The following query will extract a subgraph with the community ID value of 3:

```
MATCH p=(n)-[r]->(m)
WHERE n.community_id = 3 AND m.community_id = 3
WITH project(p) AS graph
UNWIND graph.nodes + graph.edges AS result
RETURN result;
```

Every single graph algorithm in Memgraph can be adjusted to work on a subgraph by adding a variable representing a certain path as the first parameter of any algorithm procedure.



Flow analysis

Last but not least, flow analysis is another crucial analysis when dealing with oil and gas pipelines since they can be built differently, have different widths, speeds of flow, and the flow at point A might affect the flow at point B. If we look at the network completely, we can calculate the maximum flow that we can pipe through the network, ensuring maximum capacity to the sink nodes receiving energy.

To calculate the maximum flow algorithm in Memgraph, use the query below:

```
MATCH (source:A), (sink:F)
CALL max_flow.get_flow(source, sink, "flow") YIELD max_flow
RETURN max_flow;
```

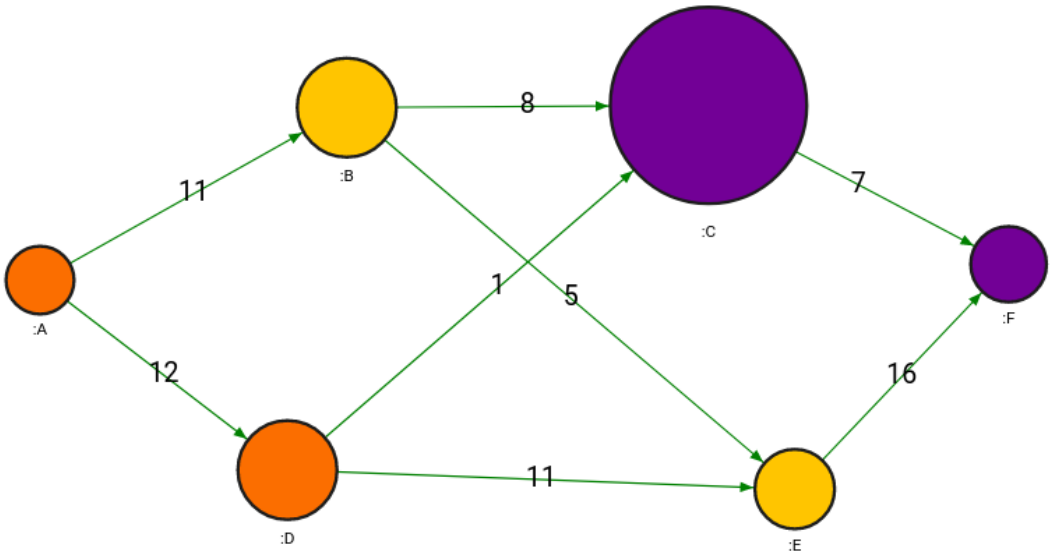
If we look at the initial graph, the maximum flow should be 23, with not all the capacity flowing through all the paths, but some paths get partially used since there is an alternative option that, in the end, yields a better flow.

To discover what's the maximum flow of each relationship, we can use the following query:

```
MATCH (n)-[r]->(m) SET r.max_flow = 0;
```

```
MATCH (source:A), (sink:F)
CALL max_flow.get_paths(source, sink, “flow”) YIELD flow, path
UNWIND relationships(path) AS rel
SET rel.max_flow = flow
RETURN startNode(rel), endNode(rel), rel, flow;
```

After running the query, we observe the following maximum flow across the network.



Why Should You Use Memgraph When Dealing With the Power Grid and Energy Topologies

Every graph database vendor has different features and capabilities that fit different use cases. Find out below what Memgraph offers as a full-fledged solution for dealing with problems in energy management systems.

Motivation

In the age of digital innovations, many industries followed the same path of introducing technology and computing ideas from scratch to adopting fully scalable solutions. A decade ago, most technologies didn't even have referential domain applications able to deal with day to day operations of the people working in the industry. The same can be said for the energy industry as well. When someone finally saw value in automating that industry and the grandiose application was built to solve all the problems, it worked marvelously until it no longer does.

That point is the era of big data. It is no longer good enough to have a solution that just automates human actions. Modern systems need to be scalable. They need to be able to handle a large number of updates of component states and not crash in highly

dynamic environments. They need to be able to do huge amounts of computations per second to balance supply and demand, as well as ingest and visualize data.

The graph representation of the power grid topology enables companies to gain insights more easily. Modern energy management systems also need to be able to react with minimum latency when power outages happen, so companies can take meaningful action and decrease the impact of the power loss. Systems need to easily simulate different scenarios to support analysts while doing risk analysis and managers making business and investment decisions.

When big data became crucial for analysis, the original systems began to hiccup and initial architectures started showing flaws. At the same time, new technologies, databases, event streams, and many other products emerged as new scalable solutions. One of those solutions is Memgraph, a high-performance graph database designed to work in dynamic environments. Find out how the Memgraph ecosystem can solve some of the biggest challenges businesses in energy topologies and pipelines are facing at the moment.

Topology analysis

Loading the topology of an energy system takes up 25-35% of the processing time, which is a staggering amount. Once that gigantic network is loaded up and rendered with visualization tools, there are several basic questions regarding the topology that need answers. What is the maximal flow through the entire network? What is the critical part of the gas pipeline, and does it require the support of additional resources? From what angles can the network be looked at, and can a change in perspective lead to new realizations? It's either really difficult to write these queries with SQL queries or impossible due to an enormous number of constraints these systems can have.

Memgraph, being a graph database, supports traversal queries, which are designed to analyze networks and highly connected data. Memgraph is able to ingest networks, retrieve already stored networks, or analyze paths and bottlenecks by inspecting simple neighboring connections or the whole network. Its graph analytics library, MAGE, which stands for Memgraph Advanced Graph Extensions, offers a number of graph algorithms that are reusable in any domain, including the energy management industry. For example, questions about the maximum capacity flow or hubs analysis can be immediately answered with betweenness centrality algorithms which will provide meaningful information without any data preprocessing whatsoever.

Another feature incorporated in Memgraph are graph projections which offer the possibility of traversing, querying, and executing graph algorithms on subgraphs rather than doing it on the whole network. The key takeaway of graph projections is an environment where people can work against the right abstractions in the graph. One data analyst might be interested in connections between power generators in one country, while the other might work against the gas pipelines in the whole EU.

Fast computations for real-time state estimation and flow analysis

To be able to react quickly to unexpected events, such as power outages in the grid, the system needs to react fast and with as low latency as possible. Upon an event, the system either needs to alert specific people, transfer energy through a different path, or do whatever it's designed to do, but the action needs to be immediate. With dynamic systems, this challenge is sometimes impossible to overcome. Systems mostly batch updates and calculate states discretely to explain what happened at some point in the past. Systems would behave more accurately if they could precisely calculate the flow or state of the system within a minimum time interval.

With Memgraph being an in-memory database, all data is loaded to memory from the initialization while still retaining the database's ACID features since snapshots are backed up to disk. Optimized in C++ to reduce memory footprint, it allows the user to stay in real-time with the high-velocity system if there is a spike in usage or data ingestion.

We talked about how graph algorithms can solve some of the typical problems related to topology. Memgraph's graph algorithms are robust to changes in the database and calculate the graph analytics ad-hoc on changed data based on deeply researched dynamical algorithms combined with triggers. Since state updates in the network sometimes affect only a local neighborhood of the nodes, insights obtained from large datasets don't need to be recomputed again. Still, they are optimized to keep on ingesting high throughput data and provide updated insights in real-time.

Extendable analysis

Although versatile tools that can handle a wide range of tasks and be set to different modes of work depending on the domain and the system do exist in the industry, there is no tool that can account for all the different day-to-day operations and situations businesses need to cover.

Versatile tools are built for what are believed to be the most common tasks for grid topology analysis, such as hubs analysis, that will help analysts further decide should the company invest in a robust power grid. But here and there, a task will emerge that is not supported by any of the tool's features.

Because Memgraph is an open-source product, its algorithms are highly extendable. Users can create new algorithms and modify existing ones to their needs. Memgraph offers APIs in Python language for prototyping solutions and analytics or the performant Rust or C++ languages with wrapped automatic memory management to decrease time to value.

Additional tools include GQLAlchemy, a wrapper around the Memgraph client that interacts with the database objects in a Pythonic way and allows Python data analysts to have an additional tool for analysis and data inspection under their belt. The result of all these tools is being able to seamlessly write any kind of traversing logic to gain insights about the current state of the energy network.

Topology custom visualization

If all the people working in the industry were identical, they would all be comfortable working with only one visual representation of the data. And although the issue might not be considered crucial, the design and visualization of the network often help people understand what's happening with the data so they can apply their own knowledge to make decisions. Because people, not computers, usually make decisions of great importance to business, such as decisions about investment, reputation or safety measures, it's in the company's best interest to enable visual customization to fit peoples' needs.

Among its many other features, Memgraph's visual interface Memgraph Lab offers customizable visualizations of graphs displayed on the plain or certain geographic grid. The latter is specifically useful for displaying topologies between physical components in various towns, countries and continents.

Nodes and relationships can be edited and saved as graph styles for any occasion, such as showing specific topologies or running algorithms that highlight different types of connections. Its visualization brain is called Orb, an open-source graph visualization library developed by Memgraph as well. It's customizable and can be embedded into any application.

Conclusion

Graph databases and algorithms are powerful tools that can answer questions regarding energy management topologies and, more importantly, solve any issues with ease and bring immediate insights and value to companies. Except for simple traversals and impact and risk analysis, graph algorithms can be used to do flow analysis, detection of communities, and dynamic analysis. Although changing legacy software is often considered a hassle, it definitely pays back if your system needs to be scalable, fault-tolerant, and performant.

Memgraph ecosystem is the best solution to analyze your highly connected power grids or gas pipelines to make meaningful decisions and improve the impact on your business, the people, and the environment around you. Besides supporting traversal queries, Memgraph can ingest networks or retrieve already stored ones, has a graph projection feature, and, most importantly, by being an in-memory graph database, it gives you the possibility to stay in real-time.