MEMGRAPH

Paulina Gacek, Jakub Hulek

Technology Stack

All performance test were conducted on a machine with following specifications:
Cpu: Intel Core i5-14600KF
Memory: 32.0 GB 6800 MT/s DDR5
Hard drive: Lexar SSD NM790 2TB
System: WSL2 UBUNTU 22.04, 20GB RAM, 50 GB swap

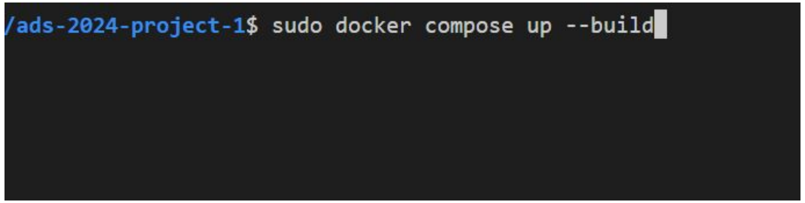
Database, graph operations, simple spatial calculations

Data preparation, CLI, most spatial calculations

Project building and data sharing

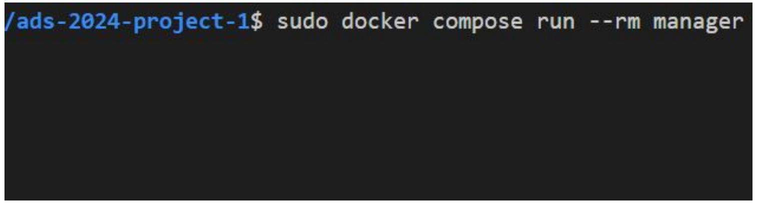
How to run

Build terminal



```
/ads-2024-project-1$ sudo docker compose up --build
```

CLI terminal



```
/ads-2024-project-1$ sudo docker compose run --rm manager
```

1. Build the project
2. Run the CLI
3. Add your data to directory “/ads-2024-project-1/data”
4. Its ready! Enter any query you like into the CLI

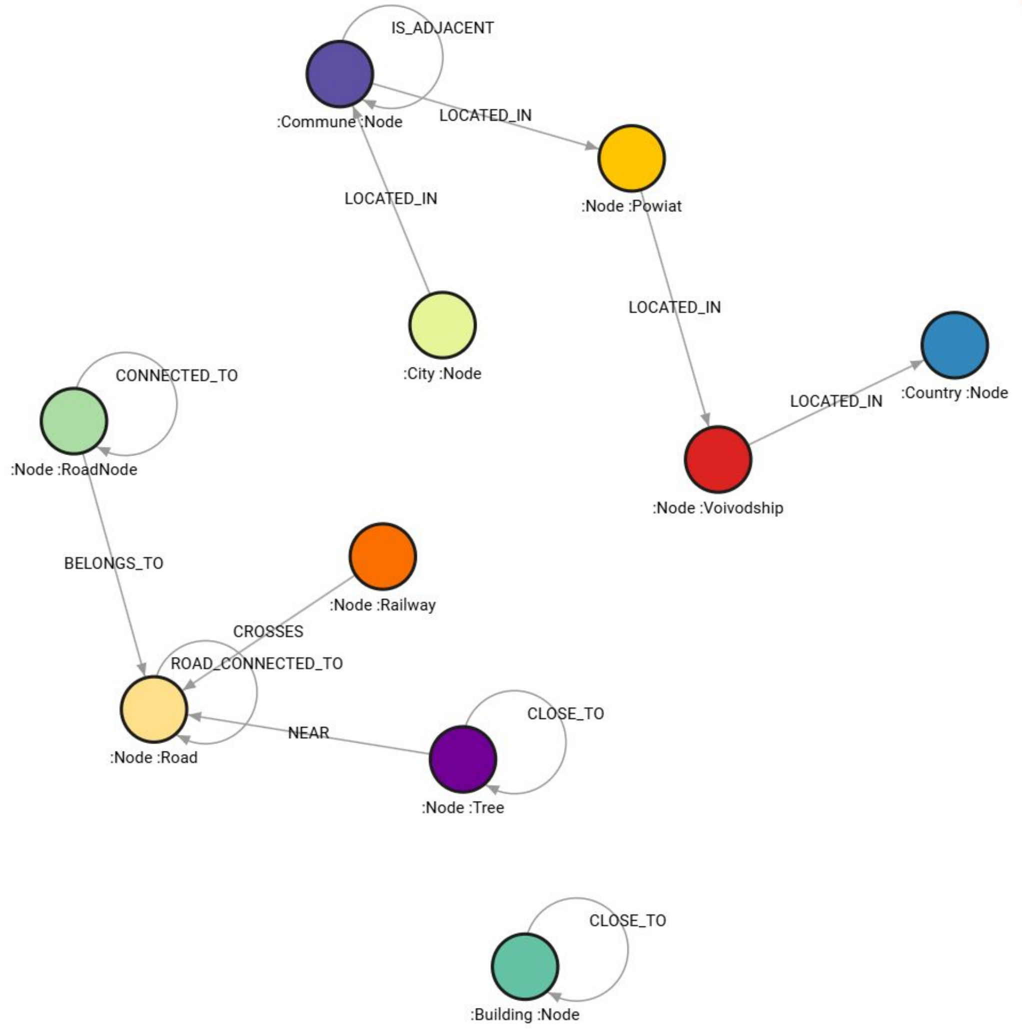
Graph Schema

All nodes contain float variables “lat” and “lng” for visualisation.

In case of points, spatial data is stored as type “point” in EPSG:2180.

In case of more complicated shapes, spatial data is stored as wkt string in EPSG:2180.

They also include a bounding box as two “point” variables for query optimization.



Data Import

Data preparation and data import

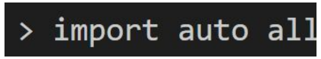
Roads data imported in 539.32 seconds.
Buildings data imported in 693.21 seconds.
Cities data imported in 18.64 seconds.
Communes data imported in 7.90 seconds.
Countries data imported in 0.77 seconds.
Powiats data imported in 3.76 seconds.
Railways data imported in 7.90 seconds.
Trees data imported in 14.39 seconds.
Voivodships data imported in 1.34 seconds.

Total time taken: 1287.24 seconds.
Around 21 minutes 37 seconds.

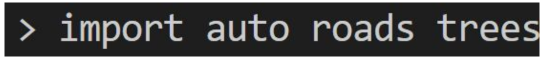
Only data import

Roads data imported in 68.20 seconds.
Buildings data imported in 42.62 seconds.
Cities data imported in 4.19 seconds.
Communes data imported in 1.90 seconds.
Countries data imported in 0.40 seconds.
Powiats data imported in 1.05 seconds.
Railways data imported in 2.05 seconds.
Trees data imported in 3.89 seconds.
Voivodships data imported in 0.52 seconds.

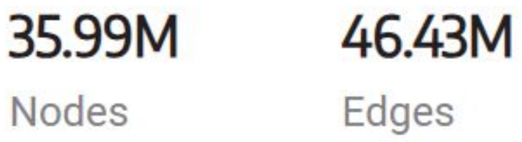
Total time taken: 124.82 seconds.
Around 2 minutes 5 seconds



```
> import auto all
```



```
> import auto roads trees
```



35.99M
Nodes

46.43M
Edges

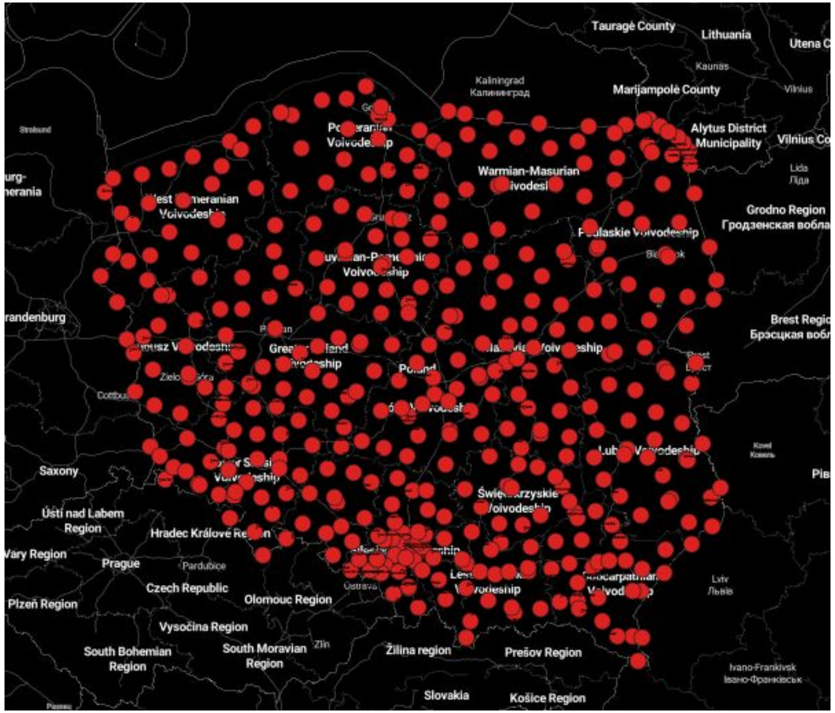
Memory usage after Data Import

23GB

(25GB peak usage)

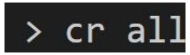
Data Import Examples

Example road and its nodes

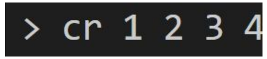


All powiats in Poland

Relationship Creation



```
> cr all
```



```
> cr 1 2 3 4
```

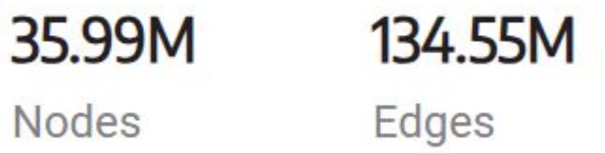
Data retrieval, relationship calculation and relationship import

Relationship creation report:
Relationship 1 created in 76.08 seconds.
Relationship 2 created in 9.92 seconds.
Relationship 3 created in 8.82 seconds.
Relationship 4 created in 3.97 seconds.
Relationship 5 created in 46.87 seconds.
Relationship 6 created in 1513.88 seconds.
Relationship 7 created in 27.98 seconds.
Relationship 8 created in 54.51 seconds.
Relationship 9 created in 148.55 seconds.
Relationship 10 created in 167.63 seconds.
Total time taken: 2058.22 seconds.
Around 34 minutes 18 seconds.

Due to problems with memory usage, we had to switch memory settings to IN_MEMORY_ANALYTICAL.
This disables ACID compliance, but achieves at least 56B savings on each node and edge and improves performance.

Only relationship import (with index creation)

Relationship creation Report:
Relationship 1 created in 41.63 seconds.
Relationship 2 created in 2.00 seconds.
Relationship 3 created in 1.40 seconds.
Relationship 4 created in 1.37 seconds.
Relationship 5 created in 14.53 seconds.
Relationship 6 created in 175.36 seconds.
Relationship 7 created in 37.89 seconds.
Relationship 8 created in 24.87 seconds.
Relationship 9 created in 154.68 seconds.
Relationship 10 created in 36.32 seconds.
Total time taken: 490.06 seconds.
Around 8 minutes 10 seconds.



35.99M
Nodes

134.55M
Edges

Memory usage after Relationship Creation

36GB

(55GB peak usage)

Relationship Creation

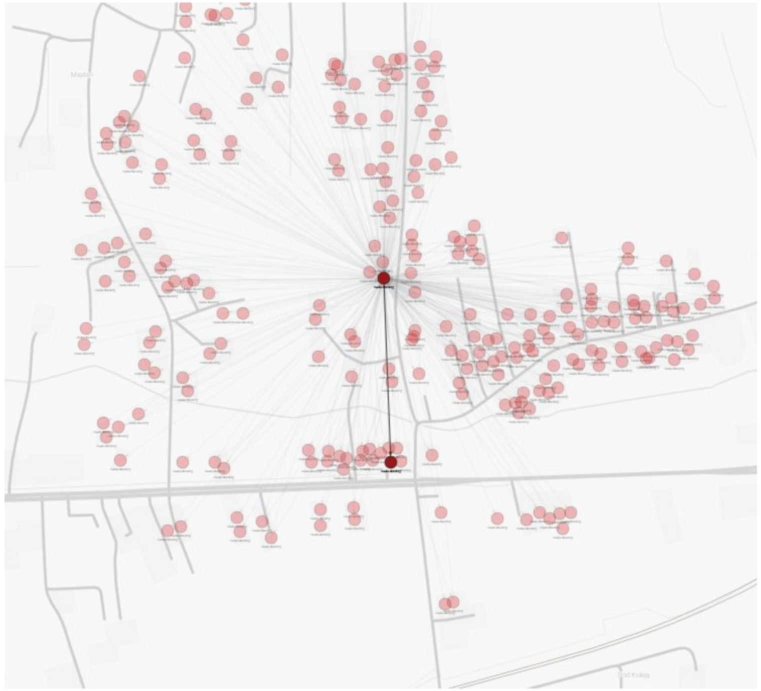
Relationships 1-6, 8, 10:
Cities in communes
Communes in powiats
Powiats in voivodships
Voivodships in countries
Adjacent communes
Neighbouring buildings
Trees close to roads
Railways crossing roads

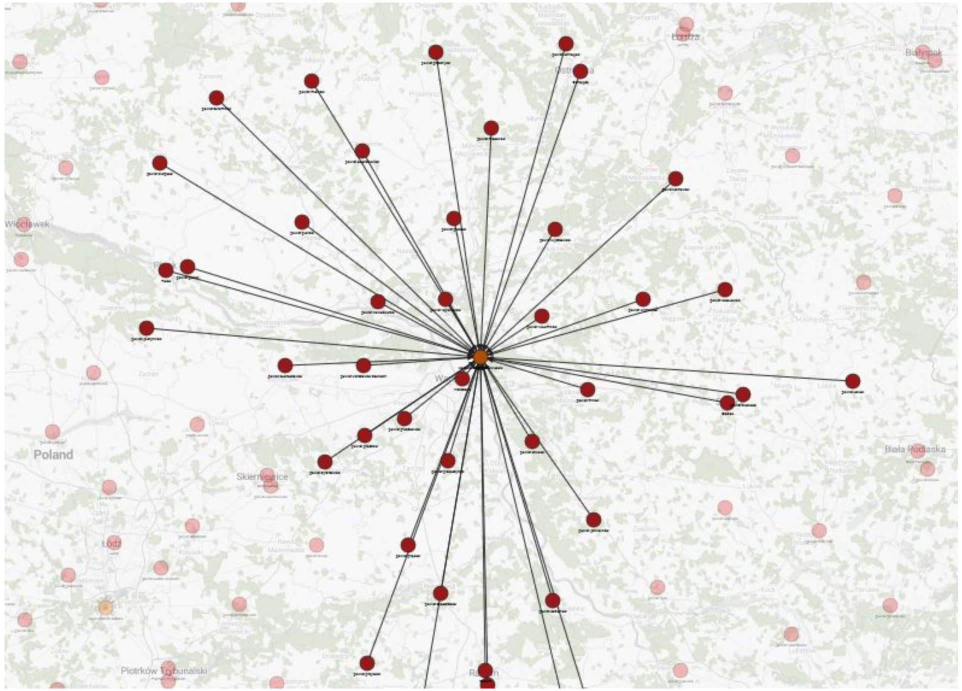
retrieval optimization with area of interest, calculated with shapely in manager created with “LOAD CSV”

Relationship 7: Neighbouring trees	calculated and created in memgraph, optimized with a POINT INDEX
Relationship 9: Connected roads	calculated and created in memgraph

Due to problems with computational complexity neighbouring buildings detection had to be clipped to a small area of Poland. Bounding box of Powiat Wielicki was chosen (140 000 buildings, less than 1% of all buildings). This resulted in *only* 23 million new edges.

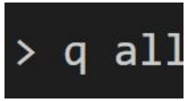
Relationship Creation Examples


Example building neighbours

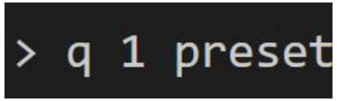


Example Powiats in Mazowieckie voivodship

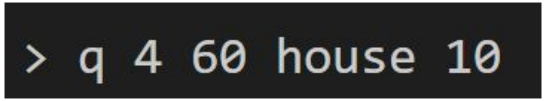
Querying



```
> q all
```



```
> q 1 preset
```



```
> q 4 60 house 10
```

Creating indexes, running queries

Query running report:
Query 1 run in 2.06 seconds.
Query 2 run in 1.38 seconds.
Query 3 run in 1.03 seconds.
Query 4 run in 102.21 seconds.
Query 5 run in 1.47 seconds.
Query 6 run in 161.85 seconds.
Query 7 run in 90.03 seconds.
Query 8 run in 5.16 seconds.
Query 9 run in 163.84 seconds.
Query 10 run in 5.62 seconds.
Total time taken: 534.65 seconds.
Around 8 minutes 54 seconds.

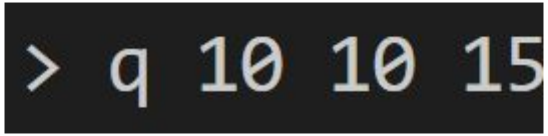
Only running queries

Query running report:
Query 1 run in 0.88 seconds.
Query 2 run in 0.78 seconds.
Query 3 run in 0.72 seconds.
Query 4 run in 60.12 seconds.
Query 5 run in 0.55 seconds.
Query 6 run in 150.70 seconds.
Query 7 run in 33.43 seconds.
Query 8 run in 0.99 seconds.
Query 9 run in 153.68 seconds.
Query 10 run in 3.29 seconds.
Total time taken: 405.17 seconds.
Around 6 minutes 45 seconds

All queries (apart from query 6 - parallel roads) are executed with a CYPHER query, and results is transformed to an appropriate json form in the manager.

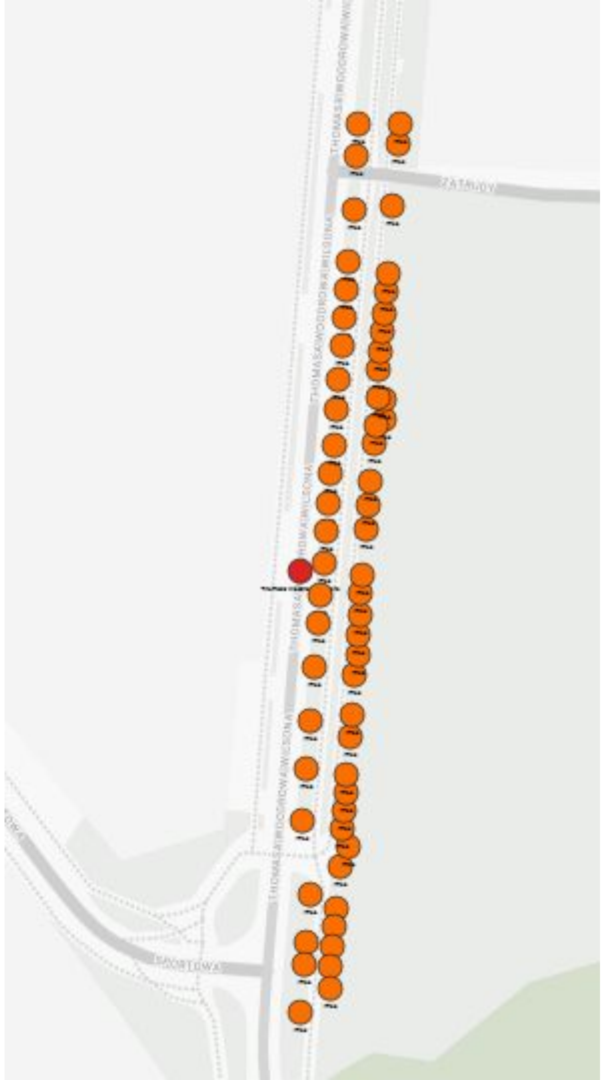
Example Querying

Query 10 - Trees within 15 meters of a road, at least 10



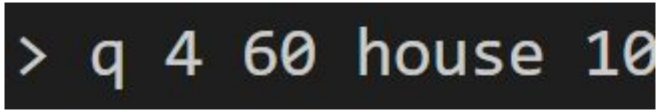
```
> q 10 10 15
```

```
MATCH (road:Road)
MATCH p=(tree:Tree)-[e:CLOSE_TO]->(road)
WHERE e.distance <= {max_distance}
WITH COLLECT(tree.id) as trees, road
WHERE size(trees) >= {min_count}
RETURN road.id as road_id, road.name as road_name, trees as tree_ids
```



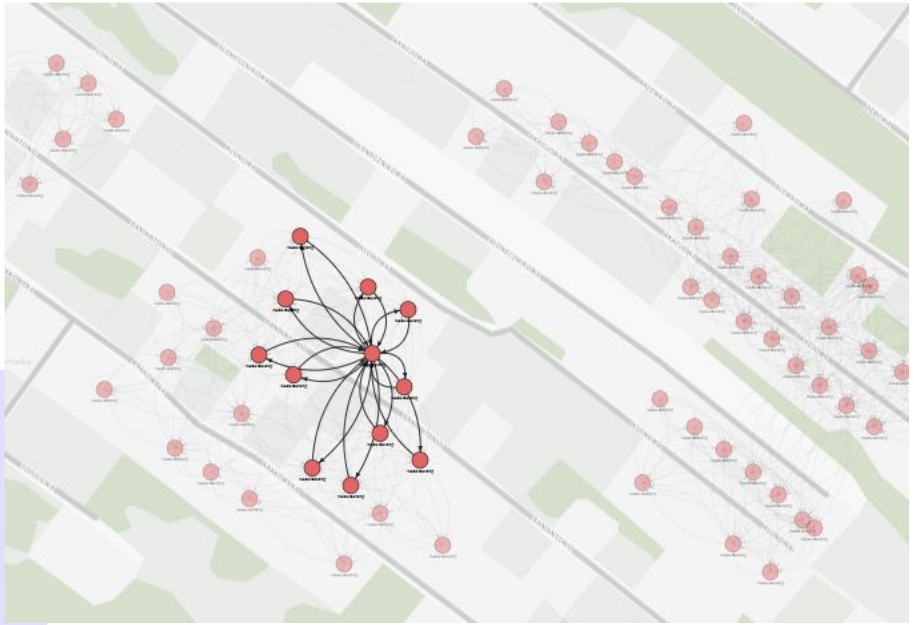
Example Querying

Query 4 - Clusters of building of type “house”, distance 60 meters, at least 10 buildings for a cluster



```
> q 4 60 house 10
```

```
MATCH p=(:Building {{building:"{building_type}"}})-[e:CLOSE_TO]
-(:Building {{building:"{building_type}"}})
WHERE e.distance <= {max_distance}
WITH project(p) AS subgraph
CALL nxalg.strongly_connected_components(subgraph)
YIELD components
UNWIND components as c
WITH c
WHERE size(c) >= {min_count}
RETURN  EXTRACT(n in c | n.id) as ids
```



MEMGRAPH

Thank you!

Paulina Gacek, Jakub Hulek