MethodsX 7 (2020) 101072

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

Computationally scalable geospatial network and routing analysis through multi-level spatial clustering.

Jonathan Chambers

University of Geneva, Switzerland

A B S T R A C T

Network analysis finds natural applications in geospatial information systems for a range of applications, notably for thermal grids, which are important for decarbonising thermal energy supply. These analyses are required to operate over a large range of geographic scales. This is a challenge for existing approaches, which face computational scaling challenges with the large datasets now available, such as building and road network data for an entire country.

This work presents a system for geospatial modelling of thermal networks including their routing through the existing road network and calculation of flows through the network. This is in contrast to previous thermal network analysis work which could only work with simplified aggregated data.

- We apply multi-level spatial clustering which enables parallelisation of work sets.
- We develop algorithms and data processing pipelines for calculating network routing.
- We use cluster-level caching to enable rapid evaluation of model variants.

A R T I C L E I N F O

Method name: Geospatial multi-level clustering and graph theoretical analysis
Keywords: Energy, Thermal networks, Geospatial, Graph theory, Data science
Article history: Received 22 June 2020; Accepted 17 September 2020; Available online 21 September 2020

E-mail address: jonathan.chambers@unige.ch

https://doi.org/10.1016/j.mex.2020.101072

Specifications Table

Subject Area	Energy
More specific subject area:	Geospatial analysis for thermal grids
Method name:	Geospatial multi-level clustering and graph theoretical analysis
Resource availability	**Software**
	Numpy: numpy.org
	Scipy: scipy.org
	Pandas: pandas.pydata.org
	Xarray: xarray.pydata.org
	Matplotlib: matplotlib.org
	Cartopy: scitools.org.uk/cartopy/
	Numba: numba.pydata.org
	Postgres: postgresql.org
	PostGIS: postgis.net
	PgRouting: pgrouting.org
	Data
	Swisstopo: https://www.swisstopo.admin.ch/en/maps-data-online

Method details

Background

Geospatial analysis is being increasingly used for analysis and planning of cities and districts, in particular use of network routing algorithms for a range of applications. Thermal grids for heating and/or cooling have been promoted as an approach to decarbonising heat supply for building [1]. To-date, analysis of thermal grids has mainly focused on local studies of either thermal grids alone or as a part of a multi-energy system. These generally follow an energy-system optimisation approach, where a solver is used to optimise the energy sources and flows at all points in time [2,3]. While optimisation approaches are powerful, their computational intensity limits their application to small datasets covering generally only a town or city district. Similarly, it has been noted that thermal grids follow existing roads [4,6]. In the past, this has been applied in limited areas such as city districts and small towns [2]. However, there is no fundamental barrier to performing this calculation at national scale using road data and routing algorithms.

This work presents a system for detailed geospatial analysis of thermal grids including aspects such as location, selection of buildings to connected, and network routing. The system makes use of state-of-the-art data processing software methods to enable country-scale analysis with building-scale spatial resolution. By applying optimisations and data pipeline caching strategies, the method presented allows a level of performance that enables interactive analysis, i.e. different model parameters (such as building selection strategies or network costs) can be set and the results produced ‘on-the-fly’.

This approach enables much high spatial resolution as well as detailed network layout analysis, while previous thermal network work instead worked with raster aggregates such as sums of heat demand by hectare as a method for simplifying the calculations and approximating the areas where thermal networks may be built [5,7,8]. A further advantage of our method to make exploration of large scenario/parameter spaces feasible.

A nation-wide dataset of potential thermal grid routing through the existing road network was developed through the creation of a topologically correct road routes database. From this, a road distance network for the connection between neighbouring buildings was derived. Multi-level spatial clustering was applied to enable subdivision of the national datasets into subsets that can be processed using a massively parallel computing architecture. Within spatial clusters, dynamic filtering and re-processing of the spatial datasets was performed to study various connection and routing strategies.

While the method presented was applied with the concrete example of thermal networks, it should be readily adaptable to many kinds of network infrastructure such as electricity grids, gas networks, fresh water and wastewater lines, communication lines, etc.

Materials

The method is implemented in Python 3.7, the follow scientific software packages with academic references used were: Numpy 1.17 [9], Scipy 1.4 [10], Pandas 1.0 [11], Xarray 0.15 [12], Numba 0.49 [13], Matplotlib 3.2 [14]. Cartopy 0.17 [15].

Data was stored and processed using the Postgres 10 SQL database, with the Postgis 2.5 extension for geospatial computation and the pgRouting 2.6 extension to provide geospatial routing, topology, and network functionality.

Computer resources consisted of an HP Z8 workstation with a 28 core CPU (Xeon Gold 6132), 128GB RAM, and 512GB SSD.

Datasets used in this work were drawn primarily from Swiss national geospatial datasets provided by Swisstopo [16]. These included road data from the SwissTLM3D dataset and building location data and metadata from the Swiss building registry (Registre de Batiments et Logements (RegBL)) dataset. Additional data on energy demand in buildings was derived from previous work [5,7,17,18].

Process overview

The flowchart in Fig. 1 gives an overview of the method. Subsequent sections will describe elements of this in more detail.

Road network topology and routing

Calculating the shortest path through the road network is a computationally intensive task, if it was necessary to re-calculate the road-based distances between sets of buildings for every potential network configuration, performing the operation at scale would not be feasible. We therefore present an algorithm that allows more efficient calculations (Fig. 2). For this section, the location points of all buildings in Switzerland as given by the RegBL dataset was used together with the road network information provided by Swisstopo.

First, the building and road data was pre-processed in order to generate points on the roads for corresponding to building, because the routing algorithm in pgRouting operates between points located directly on the road geometry while building locations are not directly on the street. Therefore, 'routing points' for each building were generated by finding the closest point on the closest road using the PostGIS 'st_closestpoint' function.

In order to calculate routing distances for a network connecting buildings, we must define a road network topology and routing distances between buildings through the road network. The routing algorithm in pgRouting requires the he geospatial line objects that form the road network data are processed to form a topology, in this case a network graph in which each point is a node and lines from road segments make up the graph edges. pgRouting validates this graph to ensure it is topologically correct by ensuring e.g. that roads lines which cross have corresponding intersection points. This is handled by the 'pgr_createtopology()' function, which was run in parallel on batches of 100 000 road segments and completed in under 15 min.

Weighted Delaunay graph using road routing

The spatial connectivity (i.e. nearest neighbourhood) of buildings was modelled as a graph defined by the Delaunay triangulation of building locations. Delaunay triangulation builds network edges between nodes and their neighbours without generating any overlapping or intersecting edges. This enables efficient calculation as only road routing distances between buildings that are neighbours in the graph need to be calculated. The Delaunay graph was stored in the Postgres database as a table of source and target building IDs.

For each edge in the Delaunay graph, we calculate the distance through the road network using Dijkstra's algorithm through the function 'pgr_dijkstraCost' and store this as an edge weight in the graph. To calculate the total distance between an arbitrary pair of buildings, we calculate the shortest path through the Delaunay graph using the pre-calculated distance weights. As the Delaunay graph is

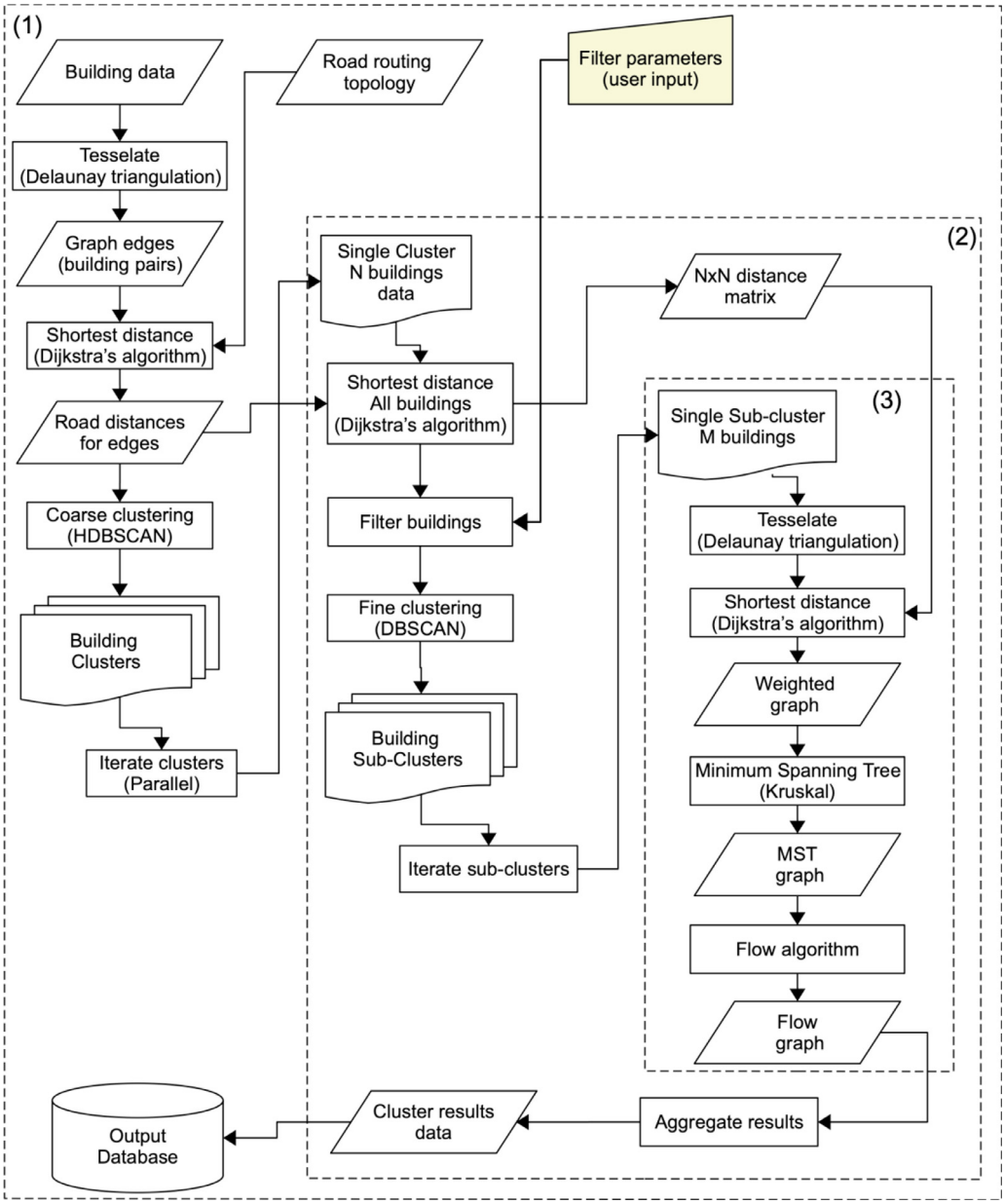

Fig. 1. Flowchart of re-clustering method. Box (1) encompasses the whole process including the pre-processing stages and the generation of the first level clusters. Box (2) is repeated for each cluster and includes the second-level clustering. Box (3) is repeated for each sub-cluster within each cluster.

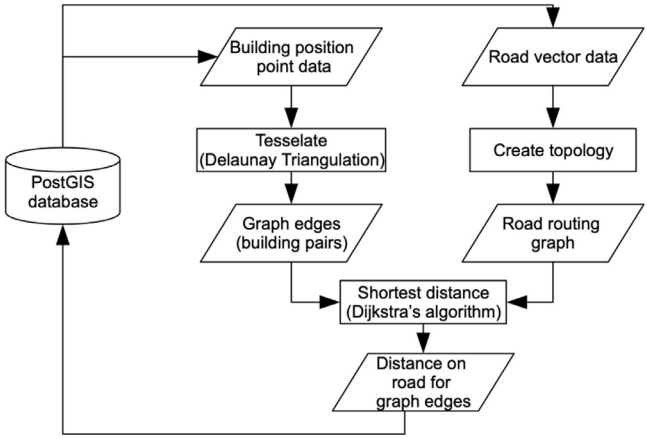

Fig. 2. Flowchart of method for calculating road distances for building Delaunay triangulation.

```
select delaunay_edges.*,
    a.id as route_source_id,
    b.id as route_target_id,
    st_asewkt(st_expand(point, 1100)) as search_zone
from delaunay_edges
join building_id a on a.id=source
join building_id b on b.id=target
```

Fig. 3. Postgres SQL query to generate a table of Delaunay graph node pairs and search radius geometry.

much simpler than the road network, this latter operation is much faster than calculating distances using the original road network. This made subsequent network analysis was considerably faster.

Assigning road distances to Delaunay graph edges was the slowest operation of this process, but only needed to be performed once. To improve performance, we supply the pgr_dijkstraCost function with a subquery that first selects only road network data within a set distance (in our case 1100 m), this reduces the amount of data that is processed for each edge – this is query is presented in Fig. 3. The search zone is stored as a Well Known Text (WKT) standard string, which is then injected into the distance calculation query using a string templating engine (e.g. Python format strings).

For each row in the Delaunay edge table, the weight field was assigned using the query in Fig. 4. The row queries were run as 48 parallel jobs and completed in approximately 72 h. Fig. 5 illustrates the output of this process.

Multi-level clustering approach

We observe that buildings are very unevenly spatially distributed into dense clusters, and furthermore that network analysis only requires buildings that are close by. This allows us to split the building stock into spatially distinct and disconnected clusters that can be analysed independently. The much smaller analysis units significantly reduce the computational complexity and enable parallelism over the clusters.

```
update delaunay_edges
   set weight=(select agg_cost from pgr_dijkstraCost(
      'select id, source, target, length as cost from jc_energy_solutions.routing_edges where
st_contains(st_geomfromewkt("{search_zone_wkt}"), geom) ',
       :route_source_id , :route_target_id, false
       ))
where index=:delaunay_row_id
join building_location on a.id = building_location.id

where route  distance is null
```

Fig. 4. Postgres SQL query for each row in the Delaunay edge table to assign the road distance between the two edge nodes as the edge weight. Note that while some query parameters may be supplied using standard Postgres query parameter syntax (variables with the ':' prefix), the search zone data indicated by the placeholder {search_zone_wkt} must be inserted using an external string templating engine.

For each cluster generated by the top-level clustering, we apply processing and filtering followed by a second stage of clustering – making this a 'multi-level' clustering approach. It must be noted that this is not the same as a hierarchical clustering, because we apply further processing on each cluster which changes the dataset for the second level of clustering (e.g. by selecting a subset of buildings). This introduces a data discontinuity which invalidates the sub-level clusters that would be produced by hierarchical clustering.

To perform spatial clustering, DBSCAN is a well-established algorithm particularly suitable for geospatial applications, as it is able to generate non-convex clusters, unlike other common algorithms such as Euclidean distance K-means or K-medoids [19]. Generating non-convex clusters is an essential requirement for this work and precludes the application of many existing algorithms. DBSCAN builds clusters from 'core' points defined as having at least a minimum number of points within a given distance. 'Peripheral' points added to the core based on an allowable distance term. The Euclidean distance is used as the distance metric between points.

HDBSCAN [20] is a further enhancement of the DBSCAN approach that allows the generation of variable density clusters. This characteristic is useful when clustering buildings at the national scale, as urban areas display a lot of variety. The implementation was also selected because of its high performance relative to other implementations and support for multi-threaded processing. While HDBSCAN also allows for hierarchical clustering, this functionality was not used for the reasons outlined above.

It should be noted that other spatial clustering algorithms could be used for the spatial clustering [21], including OPTICS [22] and FSDP [23]. As the spatial clustering is low-dimensional (2D), it is not expected that the different algorithms would produce significantly different cluster patterns. Comparison of the relative merits of different clustering algorithms is beyond the scope of this work, algorithm choice is instead based on the availability of a well-tested and high-performance implementation in Python.

Coarse spatial clustering

Using HDBSCAN, a dataset of 2 million building locations we clustered into 16,900 clusters in approximately 6 min. The cluster ID, building ID pairs were stored for each building in the database.

It has been noted that there is no formal way to set spatial clustering parameters (distance for DBSCAN and OPTICS, density threshold for FSDP) [21]. In this case, clustering distance parameter was set to 100 m following empirical observations of the feature scale for urban centres, as well as the common practise of analysis urban spaces at hectare (100 m $\times$ 100 m) resolution.

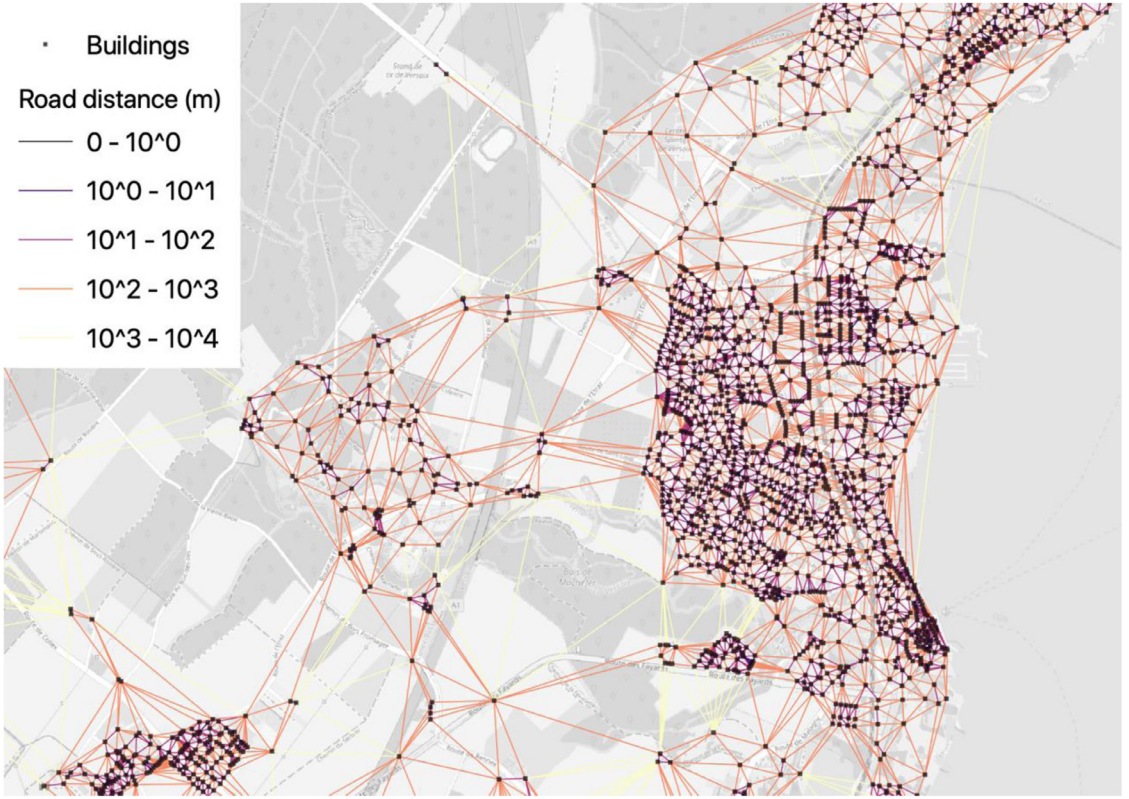

Fig. 5. Illustration of road-distance weighted Delaunay graph of buildings overlaid on building locations and a base map (base map source: OpenStreetMap).

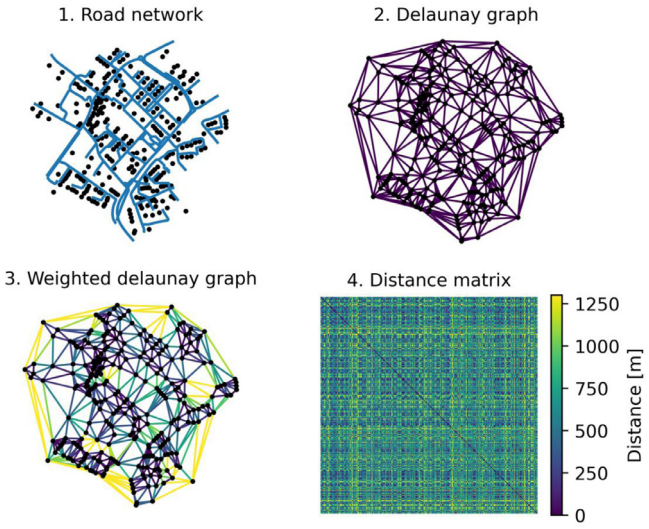

Fig. 6. Illustration of a building cluster processing. 1. The road network for the cluster is retrieved. 2. A Delaunay triangulation of the buildings in the cluster is performed. 3. Weights are calculated for Delaunay edges as the road network distance between the buildings for the corresponding edge. 4. A full distance matrix for the shortest difference between every pair of buildings is calculated by calculating the shortest path through the weighted Delaunay graph.

Changing the cluster distance parameter does impact the cluster output. A simple assessment was performed by clustering with a range of values ±40% relative to the 100 m basis, resulting in a change in the number of clusters by ±20%.

Cluster processing

The road distances between every building pair in a cluster is pre-calculated so that arbitrary subsets of buildings and the corresponding distances between them can be extracted without re-calculation. Each cluster generated by the coarse clustering algorithm can be processed in parallel.

This is performed efficiently using the pre-calculated Delaunay triangulation mesh and road distances. The Delaunay graph connects each building with its neighbours and has edge weights set to the road distances as calculated by routing through the real streets network. Therefore, a path through the Delaunay graph (which is much simpler than the road network) can be found between any two buildings and the path length is the sum of the edge weights (i.e. road distances).

The shortest distance via roads for every pair of buildings in a cluster is calculated using Dijkstra's algorithm subject to the road distance edge weights. This produces a dense N by N matrix where N is the number of buildings in the cluster and the value at index *[i, j]* is the distance via road between buildings *i* and *j*. The distance matrix is along with node metadata is stored for use in later processing as a netCDF file, one for each cluster, allowing rapid retrieval of results for further processing.

The process is illustrated in Fig. 6. A cluster of 276 buildings is shown with the road network (6.1). The Delaunay triangulation produces 809 edges (6.2). The distances through the road network are calculated for each edge (3). The full distance matrix is calculated by from the shortest distance between every building pair, for a median inter-building distance of 74 m.

Sub-clustering

Experimental work on thermal networks general involves studying different networks that result from including different sub-sets of buildings in the networks. Selecting different subsets of buildings

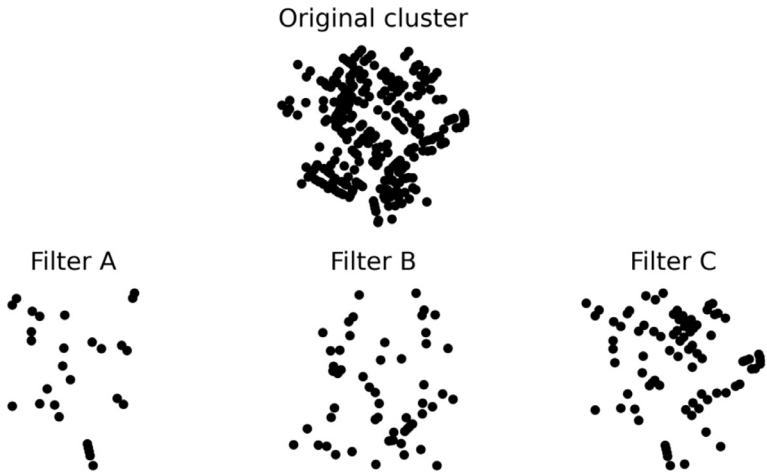

Fig. 7. Illustration of different subsets of buildings (Filter A, B, & C) generated by filtering buildings from an arbitrary cluster.

results in new layouts which invalidates the original Delaunay graph generated in previous steps and further changes the spatial density with respect to clustering.

An illustrative example is given in Fig. 7. Three building filters are applied for this example to simulate different building connection strategies. Filter A and C select 10% and 30% of the largest buildings respectively, while filter B selects 20% of buildings at random, resulting in subsets of 27, 54, and 81 buildings for A, B, and C. The change in spatial density clearly illustrates why a single step hierarchical clustering would not be appropriate, since new building density patterns vary significantly depending on the sub-cluster processing and filtering.

Therefore, after the building filtering/selection process – which may be arbitrary depending on the requirements of a specific study - the DBSCAN spatial clustering algorithm is applied to the building subset, which usually generates more than one new sub-cluster per 'macro' cluster. This is illustrated in Fig. 8 using the 81 buildings selected by Filter C (Fig. 7). 4 new clusters are found that can be processed in parallel.

Note that if desired, the clustering criteria (e.g. cluster distance threshold) can be altered in this step. In this work, we maintain the threshold of 100 m as this is a) representative of the typical scale of urban area features and b) a good approximation for the maximum extension distance for a thermal network (i.e. the distance to the most isolated building in a thermal network). In general, values should be chosen based on domain knowledge of the real spatial data used and the desired application.

Network definition

Each new sub-cluster contains a number M of buildings, and we require a weighted Delaunay graph for these buildings. Generating a Delaunay graph is straightforward, however re-calculating the road distances for the edge weights would be prohibitively computationally expensive. Therefore, we use the distance values extracted from the full distance matrix for the first level cluster and set the edges weights of the Delaunay graph to these values. Since this matrix contains the full distance between any pair of buildings in the original cluster, these distances are pre-computed once per cluster, so operation takes only ~10 ms. This also makes it trivial to generate weighted graphs for different building subsets.

Within each sub-cluster, we define the simplest heat network as one the minimal network that is able to reach all buildings. This is the Minimum Spanning Tree (MST) graph, which is derived from the

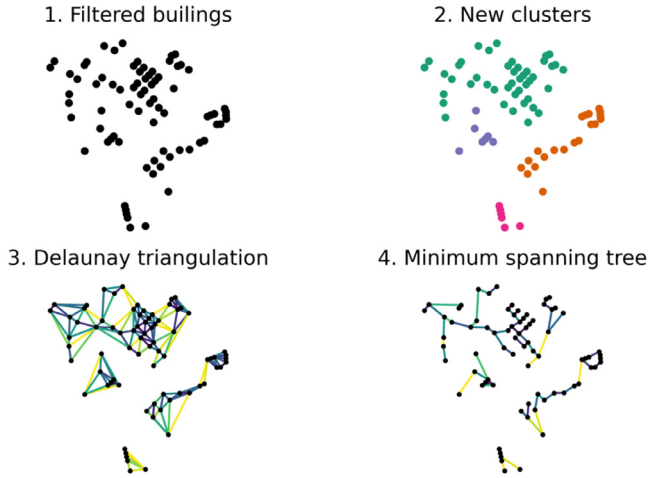

Fig. 8. Illustration of steps for processing of sub-cluster. For a given selected subset of buildings (1), DBSCAN is applied to generate new clusters (2). The Delaunay triangulation is applied to each new sub-cluster (3) and the MST generated for each sub-cluster (4).

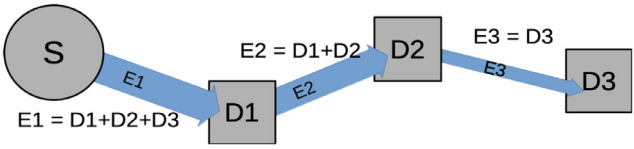

Fig. 9. Illustration of total flow through edges E1, E2, E3 from source S to demand sites D1, D2, & D3.

sub-cluster Delanuay graph using the Kruskal algorithm and the road network distances as weights. The implementation of Kruskal's algorithm in Scipy was used.

Flow analysis

It is useful for a range of analysis (e.g. connection sizing) to estimate the thermal flow through the MST graph for the sub-cluster of size M. For this, a simplifying approximation is made that all flows are from a given graph node (building) out to each building. Therefore, flows through any graph edge are the sum of the flow to the graph node (building) at the end of the edge and any further downstream nodes (Fig. 9). The flow to a given node in this case is set by the heat demand of the building, which is part of the building data table.

In order to calculate these flows, we required the path through the MST from each node to the root node in terms of the actual sequence of nodes that must be visited to reach the root. This information is provided by SciPy the "shortest_path" function which implements Dijkstra's algorithm. This function generates a vector of *predecessors* of length M. Each element of the vector (*predecessors[j]*) gives the index of the previous node in the path from the root node to point *j*. If no path exists between *i* and the root node, the value is set to −9999.

The implementation, illustrated in a simplified form in Fig. 10, is as follows. We initialise a square (*M*x*M*) graph matrix to store the flows as edge weights, and set all the initial weights to zero. For each graph node we traverse the path back to the route node and accumulate the demand flows as the edge weights in the graph matrix.

For example, if we are starting with node *j*, we start by fetching the node data D_j (in our case, energy demand data) from a vector of node data. We fetch the predecessor of node *j* from the

```
def calculate_flow_matrix(M, root_node, node_data_vector, predecessors):
    flow_matrix = array(shape=(M,M))

    for current_node in range(M):
        data_value = node_data_vector[current_node]

        while current_node != -9999 and current_node != root_node:
                previous_node = current_node
                current_node = predecessors[current_node]
                if current_node == -9999:
                    break
                flow_matrix[previous_node, current_node] += data_value

    return flow_matrix
```

Fig. 10. Simplified Python code of the flow calculation algorithm.

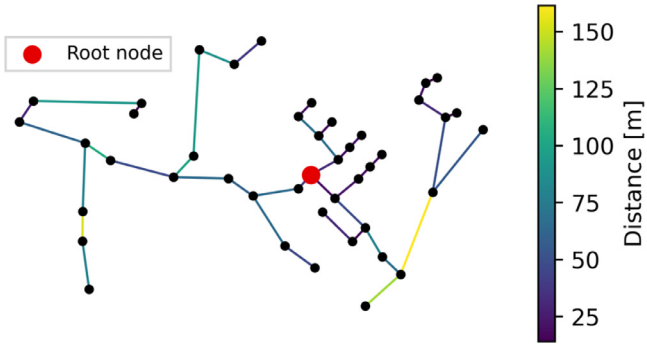

Fig. 11. Simplified graph of inter-building road distances. Note that the displayed edges indicate Delaunay triangulation connectivity rather than physical routes through the road network.

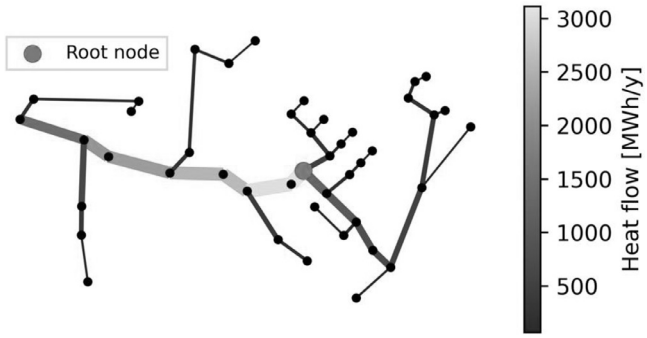

Fig. 12. Simplified graph of heat flow calculation through the graph edges from the root node to all other nodes. Flow amount is indicated by edges colour and width. Note that the displayed edges indicate Delaunay triangulation connectivity rather than physical routes through the road network.

predecessors vector, *predecessors[j]*, which gives a new node ID *k*. The node ID *k* is the next node on the path back to the root node. We add the data value D_j to the edge between the node and its predecessor *(j,k)* in the flow matrix. We then get the next node ID on the path using *predecessors[k]*, and add the end node demand D_j to that edge. This is repeated until we reach the root node.

This is process is repeated for each node. Since we accumulate the data values *D* by summation for each path, the resulting flow graph will contain the sum of the flow through each root->node path, and each graph edge weight will be the sum of the downstream node of that edge and all other nodes further downstream from that node.

Note that to achieve desired performance, we implemented the algorithm using the Numba Python compiler (which can greatly increase the performance of a subset of numeric Python code). An example is given of the input road length graph in Fig. 11 and an arbitrarily selected source (root) node, while Fig. 12 Illustrates the flow amount per graph edge from the chosen root node. It demonstrates the large differences in flow in different sections, with a range of 65–3700 MWh/year. This would have significant implications for the sizing of flow elements in this network (pipe size).

Implementation notes

This work made extensive use of graph theory concepts and graph data structures. Two main representation of graph structures were used. The original Delaunay triangulation graph that covered the whole map was stored as building ID pairs (source, target) in an SQL table. When processing

individual clusters and performing the second-level Delaunay triangulation and graph processing, the graphs were represented using sparse matrices from the 'scipy.sparse' module. This made it simple to also use Scipy's implementations of graph algorithms (such as Kruskal's minimum spanning tree algorithm). Using sparse graphs guaranteed low memory usage, which furthermore simplified parallel processing as it allowed more processes to run in parallel without exhausting system memory.

Conclusion

The method presented allows the generation of district thermal network routings through multi-level clustering of buildings. The method has been demonstrated to be readily parallelizable, enabling rapid re-calculation under different conditions, such as different building selections.

The method was demonstrated using datasets provided by the swiss geospatial data agency. These present advantages in terms of high quality and internal consistence. The same methods could be applied using a range of datasets, including national datasets similar to the Swiss ones or using open datasets such as Open Street Map [24].

While the work presented focused on thermal networks, the principals used are valid for any kind of urban resource distribution network that would be expected to be installed under existing roads including water, electricity, telecoms etc. It could also be adapted to apply to transportation flows (e.g. distribution from a warehouse).

Declaration of Competing Interest

None of the authors of this paper has a financial or personal relationship with other people or organizations that could inappropriately influence or bias the content of the paper.

Acknowledgements

This research project is financially supported by the Swiss Innovation Agency Innosuisse as part of the Swiss Competence Centre for Energy Research SCCER FEEB&D (Future Energy Efficient Buildings & Districts) as well as of SCCER CREST (Competence Centre for Research in Energy, Society and Transition).

References

[1] D. Connolly, et al., Heat roadmap Europe: combining district heating with heat savings to decarbonise the EU energy system, Energy Policy 65 (Feb. 2014) 475–489.
[2] L. Girardin, F. Marechal, M. Dubuis, N. Calame-Darbellay, D. Favrat, EnerGis: a geographical information based system for the evaluation of integrated energy conversion systems in urban areas, Energy 35 (2) (Feb. 2010) 830–840.
[3] M. Mohammadi, Y. Noorollahi, B. Mohammadi-ivatloo, H. Yousefi, Energy hub: from a model to a concept – A review, Renew. Sustain. Energy Rev. 80 (Dec. 2017) 1512–1527.
[4] J. Unternährer, S. Moret, S. Joost, F. Maréchal, Spatial clustering for district heating integration in urban energy systems: application to geothermal energy, Appl. Energy 190 (2017) 749–763.
[5] J. Chambers, K. Narula, M. Sulzer, M.K. Patel, Mapping district heating potential under evolving thermal demand scenarios and technologies: a case study for Switzerland, Energy 176 (Jun. 2019) 682–692.
[6] K.N. Finney, V.N. Sharifi, J. Swithenbank, A. Nolan, S. White, S. Ogden, Developments to an existing city-wide district energy network – Part I: identification of potential expansions using heat mapping, Energy Convers. Manag. 62 (Oct. 2012) 165–175.
[7] J. Chambers, et al., Spatiotemporal analysis of industrial excess heat supply for district heat networks in Switzerland, Energy 192 (Feb. 2019) 116705.
[8] J.D. Chambers, S. Cozza, M.K. Patel, Applications of graph theory in district heat network analysis at national scale, in: Proceedings of CISBAT 2019, 2019.
[9] S. van der Walt, S.C. Colbert, G. Varoquaux, The NumPy array: a structure for efficient numerical computation, Comput. Sci. Eng. 13 (2) (. 2011) 22–30.
[10] E. Jones, T. Oliphant, P. Peterson, and others, SciPy: open source scientific tools for Python. 2015.
[11] W. McKinney, Pandas: a foundational python library for data analysis and statistics.
[12] S. Hoyer, J. Hamman, xarray: N-D labeled arrays and datasets in Python, J. Open Res. Softw. 5 (1) (2017) p. 10, doi:10.5334/jors.148.
[13] S.K. Lam, A. Pitrou, S. Seibert, Numba: a LLVM-based Python JIT compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, 2015.
[14] J.D. Hunter, Matplotlib: a 2D graphics environment, Comput. Sci. Eng. 9 (3) (2007) 90–95.

[15] Met Office, Cartopy: a cartographic python library with a matplotlib interface. Exeter, Devon, 2016.
[16] Federal Office of Topgraphy, "Swisstopo homepage," 2018. [Online]. Available: https://www.swisstopo.admin.ch/. [Accessed: 29-Jan-2018].
[17] S. Schneider, P. Hollmuller, J. Chambers, M. Patel, A heat demand load curve model of the swiss national territory, IOP Conf. Ser. Earth Environ. Sci. 290 (1) (2019) 12107.
[18] K.N. Streicher, M. Berger, J. Chambers, S. Schneider, M.K. Patel, Combined geospatial and techno-economic analysis of deep building envelope retrofit, J. Phys.: Conf. Ser. 1343 (1) (2019) 12028.
[19] M. Ester, H.-.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996.
[20] C. Malzer and M. Baum, A hybrid approach to hierarchical density-based cluster selection, Nov. 2019.
[21] G. Schoier, C. Gregorio, Clustering algorithms for spatial big data, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10407, Springer, 2017, pp. 571–583. LNCS.
[22] M. Ankerst, M.M. Breunig, H.P. Kriegel, J. Sander, Optics: ordering points to identify the clustering structure, in: SIGMOD Record (ACM Special Interest Group on Management of Data), 28, Jun. 1999, pp. 49–60.
[23] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science (80-.). 344 (6191) (Jun. 2014) 1492–1496.
[24] OpenStreetMap contributors, "OpenSteetMap data." 2017.