Spectral MST-Based Graph Outlier Detection With Application to Clustering of Power Networks

Tyuryukanov, Ilya; Popov, Marjan; Van Der Meijden, Mart A.M.M.; Terzija, Vladimir

DOI
10.23919/PSCC.2018.8442671

Publication date
2018

Document Version
Final published version

Published in
20th Power Systems Computation Conference (PSCC)

Spectral MST-based Graph Outlier Detection with Application to Clustering of Power Networks

Ilya Tyuryukanov
Marjan Popov
Faculty of EEMCS
Delft University of Technology
Delft 2628CD, Netherlands
ilya.tyuryukanov@ieee.org

Mart A.M.M. van der Meijden
TenneT TSO B.V.
Arnhem 6812AR, Netherlands

Vladimir Terzija
School of Electrical and Electronic Engineering
The University of Manchester
Manchester M13 9PL, U.K.

***Abstract*—An increasing number of methods for control and analysis of power systems relies on representing power networks as weighted undirected graphs. Unfortunately, the presence of outliers in power system graphs may have a negative impact on many of these methods. In addition, detecting outliers can be a relevant task on its own. Motivated by the low number of outlier detection algorithms focusing on weighted undirected graphs, this paper proposes an efficient and effective method to detect loosely connected graph clusters below a certain number of nodes. The essence of the method lies in the efficient examination of the spectral minimal spanning tree of the input graph. The obtained results on several large test power networks validate the high outlier detection performance of the proposed method and its high computational efficiency.**

***Index Terms*—outliers, graph outlier detection, power network partitioning, power system analysis computing**

I. Introduction

The ongoing modernisation of electric power grids caused by the bulk integration of renewable energy sources requires new adaptive control strategies capable of countering the expected increased operational uncertainties and faster dynamics. Owing to the inherent graph nature of electric power networks, graph partitioning becomes an important tool for many novel control solutions. Partitioning of large-scale electric power networks into zones or areas is relevant for power system operations and planning [1], zone-based voltage control schemes [2], power system emergency control [3], power network reduction [4] and some other applications.

Spectral clustering [5] is a family of computationally efficient methods based on the eigendecomposition of certain graph matrices that plays a fundamental role in graph partitioning and cluster analysis. Despite its advantages and widespread usage, spectral clustering has several known deficiencies, one of which is its sensitivity to outliers [6]. This problem is usually tackled by applying a robust post-processing method on top of spectral clustering [3] or by removing the outliers in advance [6]. The first option limits the set of methods to produce the final partitioning, and severe outliers may still be able to impair the partitioning result [7]. Thus it is strongly recommended to filter out the severe outliers before applying spectral clustering [6]. The relevance of graph outlier detection in the power system context was illustrated in [3] for the problem of generator coherency constrained graph partitioning.

The main contribution of this paper is a computationally efficient outlier detection method for graph data. Compared to outlier detection in Euclidean point cloud data, there are noticeably less methods to find outliers in graphs [8]. The majority of existing graph outlier detection methods focuses either on *community outliers* [9] (i.e., residual nodes loosely coupled to the identified network communities), or on identifying anomalous elements in graph matrices via matrix factorization techniques [10]. These methods are not tailored to find *outlier clusters* (i.e., loosely connected groups of two or more nodes). Additionally, many state-of-the-art outlier detection methods have a higher time complexity than spectral clustering [10], [11], which may result in a computational bottleneck. Our method avoids this problem by having about the same time complexity as spectral clustering.

The main idea of the proposed method is to embed the graph in a low-dimensional Euclidean space and detect outlier clusters by analysing the minimal spanning tree (MST) of the embedded graph. The graph embedding is constructed in a well-established manner by finding several largest eigenvectors of the normalised adjacency matrix. The use of MST computed for such embedded graph was previously reported in [12] for the purpose of graph clustering. However, an output of graph clustering may not reveal all of the outliers as they may mask each other. That is, some outliers may look similar to the normal data in the presence of a substantially more severe outlier [8]. For this reason, our method searches the entire graph for loosely connected nodes and clusters below certain size to return a collection of such nodes and clusters together with their severity ranking. This information can be used for various purposes, including the robustification of graph clustering, which is detailed in this paper.

The remainder of the paper is organised as follows. Section II outlines the necessary theoretic background. Based on this information, Section III introduces the used graph outlier cluster metrics. The proposed graph outlier mining algorithm is described in Section IV, and its performance is demonstrated in Section V. Finally, Section VI concludes this work.

II. Spectral Graph Embedding

For the purpose of graph clustering or outlier detection, the network consisting of n nodes can be represented as an edge-weighed undirected graph G represented by its adjacency matrix $\mathbf{A} = [a_{ij}]$. The edge weights a_{ij} should correspond to the quantities representing the similarity between the pairs of buses in the network (e.g., branch admittance or average power flow). The set of all nodes of the graph G is denoted as V, while the set of its edges is denoted as E. For the rest of the paper, we denote matrices by bold uppercase letters. In order to avoid an excessive stacking of matrix indices, an alternative matrix indexing is occasionally used in the algorithm pseudocodes (e.g., $\mathbf{A}[i, j]$ is equivalent to a_{ij}).

Given the adjacency matrix $\mathbf{A}$, it is possible to compute the transition probability matrix $\mathbf{P}$ of the Markov chain associated with the graph G.

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A} \tag{1}$$

where d_i is the (weighted) degree of node i

$$d_i = \sum_{j=1}^{n} a_{ij} \tag{2}$$

and $\mathbf{D} = \mathrm{diag}(d_1, \dots, d_n)$ is the degree matrix. The transition probability matrix (1) describes random walks on the graph G, and its row sums are equal to one. The random walk process also relates to the graph clustering structure [13]. Namely, starting in a dense and well-isolated graph cluster, the random walk is expected to take many steps before leaving it [5], [13]. The eigenvectors corresponding to k largest eigenvalues of matrix $\mathbf{P}$ can be used to optimally embed the nodes of graph G in a low-dimensional Euclidean space $\mathbb{R}^k$ [12], [14].

As the matrix $\mathbf{P}$ is not symmetric, the symmetric matrix $\mathbf{A_n}$ that is similar to $\mathbf{P}$ is actually used to produce the graph embedding.

$$\mathbf{A_n} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \tag{3}$$

where $\mathbf{D}^{-\frac{1}{2}} = \mathrm{diag}(\frac{1}{\sqrt{d_1}}, \dots, \frac{1}{\sqrt{d_n}})$. While $\mathbf{A_n}$ has the same eigenvalues as $\mathbf{P}$, computing the eigendecomposition of a symmetric matrix is more numerically efficient [6]. The top k eigenvectors of $\mathbf{A_n}$ can be combined into the matrix $\mathbf{X} \in \mathbb{R}^k$, with each of its n rows describing the k coordinates of the corresponding node in the network. It is also common to normalize the rows of $\mathbf{X}$ to have the length one [5] to produce the row-normalized eigenvector matrix $\mathbf{Y}$

$$Y_{ij} = X_{ij}/(\textstyle\sum_{j=1}^{k} X_{ij}^2)^{1/2} \tag{4}$$

The rows of $\mathbf{Y}$ are used in this paper to produce the geometric graph embedding. While the eigenvectors of $\mathbf{A_n}$ are related but not equal to the eigenvectors of $\mathbf{P}$ (see e.g. [5] for the relationship), the corresponding row-normalized eigenvectors of these two matrices are the same. The row normalization

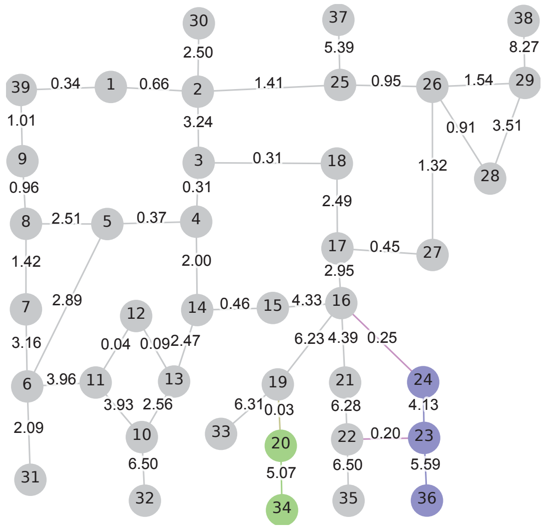


Figure 1: Active power flow graph of the IEEE 39 bus test system obtained by perturbing the loads around the nominal value. The edge weights represent the active power flows in p.u. on the 100 MVA base.

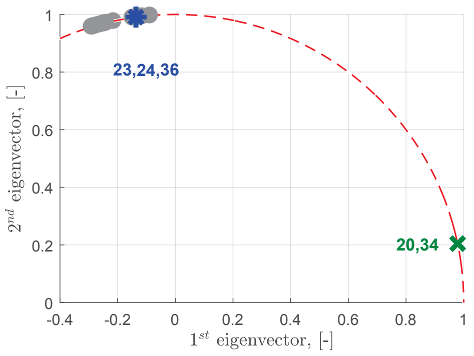


Figure 2: Row-normalized spectral embedding of the graph in Figure 1. The 2D unit sphere is shown in red.

step projects the original node coordinates into the unit hypersphere. This removes the radial variation between the points, which results in using only the angle proximities between the rows of $\mathbf{Y}$ to cluster them.

The final step is to reconstruct the structure of the graph G from the computed geometric embedding. Effectively, a new graph is created that has the same nodes and edges as the original graph G, but with its edge weights being the spherical distances between the rows of the matrix $\mathbf{Y}$ corresponding to the original edge ends. The idea to construct such graph appeared previously in [14], together with the recommendation to use spherical distances instead of Euclidean ones. The distance graph induced by the spectral embedding $\mathbf{Y}$ of the nodes of graph G is further referred to as *embedded graph*.

An illustrating example is given in Figures 1–2, which represent an active power flow graph of the IEEE 39 bus test system (as given in the MATPOWER toolbox [15]) and its spectral embedding with the two largest row-normalised eigenvectors of (3). Figure 1 highlights the node groups {20, 34} and {23, 24, 36} due to their relatively weak connection to the rest of the network. Figure 2 illustrates that nodes {20, 34} indeed form an outlier cluster that is distinctly separate from all other nodes in the spectral embedding. Because nodes {20, 34} are so distinctly apart from all other nodes, they are *masking* the presence of a less severe weakly connected cluster {23, 24, 36}. Thus, running an outlier detection algorithm for Euclidean point clouds (e.g., [11]) would not necessarily detect the cluster {23, 24, 36}, as it would become more visible only after removal of the cluster {20, 34}. While clusters of two or three nodes may not look as very small for a 39 bus network, similar situations can be even more common in realistically-sized networks.

III. Outlier Metrics

As the proposed method itself, the used outlier metrics are based on spectral graph theory and its links to random walks on graphs that were briefly touched in Section II. In addition, single outlier nodes and outlier clusters are evaluated with two different metrics.

Let the total weight of edges inside cluster C be denoted as $links(C, C)$ [16]

$$links(C, C) = \sum_{i \in C, j \in C} a_{ij}, \tag{5}$$

and the sum of weighted degrees of cluster C be denoted as $vol(C)$ [5], [16]

$$vol(C) = \sum_{i \in C} d_i \tag{6}$$

Then the cluster outlier score is adopted as

$$\phi(C) = 1 - \frac{links(C, C)}{vol(C)} \tag{7}$$

The objective function of normalized spectral clustering with multiple eigenvectors aims to minimise the sum of cluster scores (7) [5]. For this reason, a cluster with a low score value (7) is more likely to be separated. Meila and Shi [13] introduced the probabilistic interpretation of score (7): $\phi(C)$ describes the probability of the random walk to leave cluster C in one step starting from its stationary distribution, if the current walk state is in C. The possible values of (7) lie in the range $[0, 1]$, with lower values corresponding to more severe outliers.

It is problematic to extend (7) to a single node cluster, as the corresponding score value will be one even if the node is very weakly connected to the rest of the network. This is consistent with the probabilistic interpretation, since, starting at any node, random walk leaves it on the next step with the probability one. From the random walk perspective, a single node outlier in a weighted undirected graph can be understood as a node that is unlikely to be visited from its neighbors. The corresponding score for node i can be introduced as

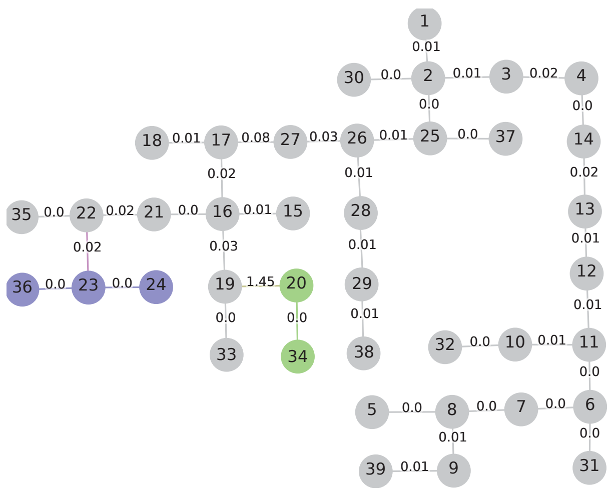


Figure 3: SpMST of the embedded graph obtained from the spectral embedding in Figure 2, rooted at node 1.

$$p_{max}(i) = \max_{\{j|a_{ij} \neq 0\}} p_{ji} = \max_{\{j|a_{ij} \neq 0\}} \frac{a_{ij}}{d_j} \tag{8}$$

where p_{ij} are the elements of the transition probability matrix $\mathbf{P}$ (1). Score (8) can be efficiently evaluated for every graph node, and it attains a low value for single graph nodes that are weakly connected to any of their neighbours.

IV. MST-based Outlier Mining

The proposed outlier detection method is based on the systematic examination of the MST of the spectral embedded graph (further referred to as SpMST). Nodes that are similar to each other in the original graph G should have close coordinates in the spectral embedding, while loosely connected clusters are placed far apart (see Figure 2 for an example). At the same time, a long edge may be included into an MST if and only if this edge is still the shortest one to connect a node or group of nodes to the rest of the network [17]. These principles play the major role in the MST-based clustering [12].

A. MST Construction

A SpMST can be initially constructed by a standard algorithm, such as Prim or Kruskal [17]. After the SpMST has been obtained, some preparations should be made to examine it in an efficient manner [18].

First, one graph node is selected as the tree root, and the tree is traversed from this root with the breadth first search (BFS) algorithm [17]. The BFS algorithm is normally capable of returning the predecessor vector, the n entries of which contain the indices of parent nodes for each node. For example, the fifth and ninth entries of the predecessor vector of the SpMST shown in Figure 3 should both contain the number 8. In our algorithm, the predecessor vector is used to orient the SpMST edges from parent nodes to their successors. That is, if the edges are stored as a list of 2-tuples, the first tuple entry should be the index of the parent (or *from*) node.

Algorithm 1 Obtain descendants of each node

```
Input: Map from nodes to their direct children child; tree root node $root$.
Output: Map from nodes to all their descendants desc.
 1: for $i \leftarrow [1, \dots, n]$ do
 2:     desc{$i$} $\leftarrow i$ // Initialization
 3: end for
 4: desc $\leftarrow$ GETDESC($root$, desc, child)
 5: return desc

 6: function GETDESC($node$, desc, child)
 7:     if child[$node$] $= \emptyset$ then return
 8:     else
 9:         for $k \leftarrow$ child{$node$} do
10:             desc $\leftarrow$ GETDESC($k$, desc, child)
11:             desc{$node$} $\leftarrow$ desc{$node$} $\cup$ desc{$k$}
12:         end for
13:     end if
14: end function
```

Algorithm 2 Outlier mining with fundamental cutsets

```
Input: SpMST as matrix $[f;\ t;\ w]$; map to nodes' descendants desc; vector of numbers of descendants of each node $nd$; adjacency matrix $\mathbf{A}$; vector of weighted degrees of each node $d$; limit on cluster cardinality $s^*$; limit on outlier score $\phi^*$.
Output: Set of outlier clusters $S$; outlier scores.
 1: for $k \leftarrow [1, \dots, n-1]$ do
 2:     if $(nd[t_k] < s^*) \vee (n - nd[t_k] < s^*)$ then
 3:         if $nd[t_k] < s^*$ then
 4:             $clu \leftarrow$ desc{$t_k$} // cluster nodes
 5:             $siz \leftarrow nd[t_k]$ // cluster size
 6:         else
 7:             $clu \leftarrow V \setminus$ desc{$t_k$}
 8:             $siz \leftarrow n - nd[t_k]$
 9:         end if
10:         Compute $vol$ (6) for nodes in $clu$ using vector $d$.
11:         $\phi_{est} \leftarrow \mathbf{A}[f_k, t_k]/vol$ // Lower bound on $\phi$
12:         if $\phi_{est} < \phi^*$ then // this spares some computations
13:             Compute $links$ (5) for the nodes in $clu$.
14:             $\phi \leftarrow 1 - links/vol$ // as (7)
15:             if $\phi < \phi^*$ then
16:                 $S \leftarrow S \cup clu$ // score (7) is also stored
17:             end if
18:         end if
19:     end if
20: end for
21: return $S$, outlier scores.
```

Secondly, we have modified the standard BFS algorithm to additionally return the direct children of each node. By its principle, the BFS algorithm starting at the top of the tree hierarchy will not leave the current node before visiting all its children. Therefore, additionally returning a key-value *map* from each node to the list of its direct children is straightforward with BFS.

Next, another map is created to store all the descendant nodes of each node in the tree. This key-value map can be created effectively by recursion as shown in Algorithm 1. With this map, it is easy to get the number of nodes below each node (including the node itself) as an extra vector of n elements.

B. Outlier Identification with Fundamental Cutsets

Since a spanning tree contains the minimal number of edges to interconnect the vertices of the graph, removing an edge from any spanning tree (including MST) induces two connected components. The set of edges that is needed to produce the same connected components in the initial graph is called *fundamental cutset*. Since any spanning tree interconnects the n graph nodes with $n-1$ edges, a spanning tree defines $n-1$ fundamental cutsets in the graph [19].

A classical spanning tree clustering algorithm (e.g. [12], [20]) would disconnect the $k-1$ longest edges of the MST to produce k clusters, thus making use of the $k-1$ fundamental cuts induced by those MST edges. Since our set goal is not to partition the graph into k clusters, but to detect loosely connected subgraphs below certain size, we are going to examine all $n-1$ fundamental cuts of the SpMST. The outline of our implementation is shown in Algorithm 2.

The SpMST is provided to Algorithm 2 as an $(n-1) \times 3$ matrix, the first two columns of which contain the *from* and *to* nodes of each directed SpMST edge, while the third column contains the edge length. Due to the availability of precomputed descendant nodes for any node, the cardinalities of connected components of each fundamental cut can be found in constant time. Then the algorithm simply checks if one of the components satisfies the cluster size criterion and, if it does, the algorithm further computes its outlier score. As the sought clusters are small, the set difference operation will only be computed a few times to return the limited number of valid outlier candidates containing the SpMST root node. Components that are smaller than the predefined size s^* and posses a low enough outlier score are saved as outlier clusters.

C. Outlier Identification with Single-Link Clustering

Section IV-B contained an example of *divisive clustering* on MST, as each iteration of Algorithm 2 divided the whole tree into two parts and evaluated them. However, some clusters with relatively high scores (7) can only be discovered by removing more than one long edge in the SpMST. This may also depend on the number of used eigenvectors (e.g., a cluster may become detectable with Algorithm 2 if more eigenvectors of (3) are considered). Nevertheless, introducing an additional bottom up (or *agglomerative*) clustering mechanism makes outlier mining more robust in general.

The agglomerative outlier mining uses the known analogy between single-link hierarchical clustering and the Kruskal algorithm for MST [20]. Algorithm 3 is largely similar to the Kruskal algorithm using the union-find data structure (as given

Algorithm 3 Outlier mining with single-link clustering

```
    Input: SpMST as matrix $[f; t; w]$; adjacency matrix $\mathbf{A}$;
    vector of weighted degrees of each node $d$; limit on cluster
    cardinality $s^*$; limit on outlier score $\phi^*$.
    Output: Set of outlier clusters $S$; outlier scores.
 1: Reorder the column vectors $f$, $t$ and $w$ in the increasing
    order of values of SpMST edge weights $w$.
 2: for $i \leftarrow [1, \ldots, n]$ do
 3:     comp$\{i\} \leftarrow i$ // each node is in its own component
 4:     $siz[i] \leftarrow 1$ // each component has size one
 5: end for
 6: $pred \leftarrow [1, \ldots, n]$ // each component is its own parent
 7: for $k \leftarrow [1, \ldots, n-1]$ do
 8:     $c1 \leftarrow$ GetRoot$(f_k, pred)$ // component id of node $f_k$
 9:     $c2 \leftarrow$ GetRoot$(t_k, pred)$ // component id of node $t_k$
10:     if $siz[c1] + siz[c2] < s^*$ then
11:         if $siz[c1] < siz[c2]$ then
12:             $pred[c1] \leftarrow c2$ // Set $c2$ as parent of $c1$
13:             $siz[c2] \leftarrow siz[c2] + siz[c1]$ // update sizes
14:             comp$\{c2\} \leftarrow$ comp$\{c2\} \cup$ comp$\{c1\}$
15:             comp$\{c1\} \leftarrow \emptyset$
16:             $clu \leftarrow$ comp$\{c2\}$ // the component to test
17:         else
18:             The same operations as in the true branch,
                but with setting $c1$ as the parent of $c2$.
19:         end if
20:         Compute $\phi$ (7) for the nodes in $clu$.
21:         if $\phi < \phi^*$ then
22:             $S \leftarrow S \cup clu$ // score (7) is also stored
23:         end if
24:     end if
25: end for
26: return $S$, outlier scores.

27: function GetRoot$(node, pred)$ // Get component's id
28:     while $node \neq pred[node]$ do
29:         $node \leftarrow pred[node]$
30:     end while
31: return $node$
32: end function
```

Algorithm 4 Resolve cluster intersections

```
    Input: Vector of cluster outlier scores $\varphi$ (7); map to node
    indices of each cluster clu; vector of cluster cardinalities
    $siz$; initial number of clusters $n_{ou}$.
    Output: Set of clusters $S$ with resolved node overlaps.
 1: Reorder the clusters clu and their sizes $siz$ in the increas-
    ing order of the cluster outlier scores $\varphi$ (7).
    // Transform the node indices of each cluster into the cluster
    indicator row vectors combined into the matrix $\mathbf{T}_{n_{ou} \times n}$.
 2: for $i \leftarrow [1, \ldots, n_{ou}], j \leftarrow [1, \ldots, n]$ do
 3:     if $j \in$ clu$\{i\}$ then $T_{ij} = 1$
 4:     end if
 5: end for
 6: Set the status of each cluster indicator row of $\mathbf{T}$ to true.
 7: $current \leftarrow 1$
 8: while status of some row of $\mathbf{T}$ is true do
 9:     Find clusters with indicator rows containing any, but
        not all ones in the non-zero columns of the current
        row. These clusters partially intersect with the current
        one.
10:     Find clusters with indicator rows containing all ones in
        the non-zero columns of the current row and having
        the same entry in $siz$ as the current cluster. These
        clusters are the duplicates of the current one.
11:     Mark clusters discovered with the above two state-
        ments for the deletion. Set the corresponding rows of
        $\mathbf{T}$ to zero and the status of these rows to false.
12:     Set the current row of $\mathbf{T}$ to zero and its status to false.
13:     $current \leftarrow$ index of the topmost indicator row with
        the true status.
14: end while
15: return $S$ as the set of clusters in clu not marked for the
    deletion, and (optionally) their scores.
```

in [17]) and is included here for the completeness and clarity of presentation.

As Algorithm 3 operates on the already constructed MST without any loops, it does not need to check if the two nodes are already in the same component. Instead, the cardinality of the resulting connected component is checked, and if a too large component results from adding the next edge, this edge is skipped. As in the original Kruskal algorithm, the edges are added between the graph nodes in the order of increase of their length. Thus, the union-find data structure, which is minimally represented by the component predecessor vector $pred$, component size vector siz, and the GetRoot function, is only used to keep track of the connected components resulting by sequentially adding the SpMST edges between the nodes. The additional variable `comp` is used to keep track of the current nodes in each connected component. It is shown in Algorithm 3 as a map from the component's id to the component's node indices, but this may be not the most efficient data structure. Finally, Algorithm 3 evaluates the outlier score of each newly obtained connected component and saves the component if the score is below the threshold.

D. Aggregation of Results

It is common in large-scale networks to detect several outlier clusters that overlap with each other, and especially if the limits on outlier score (7) and outlier size were set rather large. While this situation occurs naturally (e.g., the same tightly clustered core group of nodes can be isolated with several cutsets of varying quality), it is often desirable to meaningfully resolve the cluster intersections. Thus Algorithm 4 is proposed.

Algorithm 4 aims to remove clusters that partially overlap with a cluster having a lower score (7). This is done because combining two non-fully overlapping clusters would induce a new cluster, the properties of which (e.g., the size) are hard to control. However, Algorithm 4 would keep a less severe outlier cluster if it fully includes a more severe one, as the union of such clusters will still result in a valid cluster, with the smaller cluster being automatically integrated into the larger one. This integration step can be done by retrieving the edges running inside of each of the relevant clusters and using this set of edges to find the subgraphs induced by these edges (e.g., with a connected components algorithm).

The two steps (Algorithm 2 and Algorithm 3) of the outlier mining process usually detect many identical clusters. Thus, Algorithm 4 is especially relevant in the context of combining the results of multiple outlier detection techniques.

E. Algorithm Summary

After introducing all the necessary procedures, the proposed outlier mining algorithm can be summarized as follows:

1) Select k as the dimension of spectral embedding and find the first k eigenvectors of the matrix $\mathbf{A_n}$ (3).
2) Normalize the rows of the eigenvector matrix according to (4) and construct the embedded graph using the sets V and E of the original graph G and the spherical distances between the points in the spectral embedding [14].
3) Construct the SpMST as described in Section IV-A.
4) Use Algorithms 2 and 3 to detect the outlier cluster candidates.
5) Aggregate the results as described in Section IV-D and Algorithm 4.
6) Evaluate score (8) for every graph node to get the ranking of loosely connected single graph nodes.
7) Process the obtained clusters and single graph nodes as required by the application.

The first and last steps in the above list still require some clarification. Selecting a larger value of k is important if the method is needed to be more sensitive (i.e., to discover less pronounced small clusters). For detecting severe outliers (e.g., with score (7) below 0.01), setting of k to 3 or 4 should be enough for most cases. In what about the last step, this paper considers removing graph outliers in advance to prevent their separation by a spectral graph partitioning algorithm. We achieve this by merging each outlier cluster into a single node. As discussed in Section III, spectral clustering tends to avoid the separation of single nodes. To further avoid the separation of single node outliers (including the newly formed merged nodes), they are merged with a neighbour node to which they have the strongest connection. The results in Section V illustrate that this approach is quite efficient. In general, the processing of the returned outliers can be different for a different application.

V. Test Results

To test the detection performance of the proposed graph outlier mining method, we have applied it as a pre-processing step for the partitioning of randomly generated active power flow graphs of four networks from the MATPOWER toolbox [15], [21] ranging from 300 to 2869 nodes. In addition, we have tested the computational performance of the proposed method on the MATPOWER networks of up to 13659 nodes.

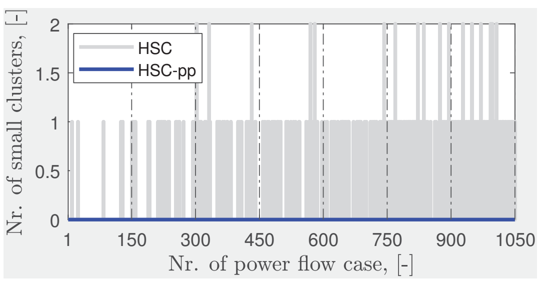
A. Sampled Random Power Flows

To generate a large number of tests, 150 random power flows were generated for each number of clusters for each network. The larger test networks *case1354pegase*, *case2383wp* and *case2869pegase* were partitioned into $k = 2, \dots, 12$ clusters, while the smaller network *case300* was partitioned into $k = 2, \dots, 8$ clusters. To generate the random power flows, the power demand of each load was modelled as a uniformly distributed random variable with the range of $\pm 50\%$ of the nominal power [7].

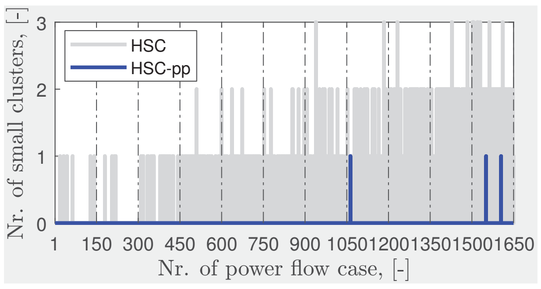
The baseline clustering algorithm was chosen to be the hierarchical spectral clustering method (HSC) [14] with the average linkage criterion. Using hierarchical clustering on the spectral embedded graph was shown to produce high-quality clusters, while ensuring each cluster to be a connected subgraph [14]. However, hierarchical clustering is generally known to be sensitive to outliers if the number of clusters is given as input [6], which was confirmed by our tests on randomly generated power flow graphs. Therefore, the goal of this case study is to show the ability of the proposed outlier detection method to prevent highly unbalanced graph partitioning by detecting and handling the small clusters before applying a graph partitioning algorithm.

For this case study, we have counted the occurrences of clusters that are smaller than 10% of the average cluster size (i.e., lower than $\mathrm{round}(0.1n/k)$ for k being the number of clusters) and tried to prevent such occurrences with our graph outlier detection method. To achieve this goal, it was necessary to increase the outlier cluster threshold ϕ^* as the number of clusters was increased. Bi-partitioning the network aims to select the two most pronounced clusters. However, the clusters that are not distinct enough for the bi-partitioning may become important if a larger number of clusters is requested, which explains the need to increase ϕ^* for the growing k. We have used the heuristic expression $\phi^* = 0.02 \log_2(k)$ for the cluster outlier score threshold. For the *case2383wp* test network, reduction of too many clusters has caused the network to become over-constrained, which resulted in frequent small clusters with high values of score (7) (the values of 0.1–0.5 or larger reflect poor clusters). For this reason, the value of ϕ^* for the *case2383wp* test network was constrained not to exceed 0.03. For the *case2869pegase* network, the value of ϕ^* was set to 0.02 because it was acceptable not to set it to a higher value. For the single node outliers, we have selected to keep the top 7% of the single node outlier candidates returned by Algorithm 2 instead of trying to guess a good absolute value for p^*_{max}. The number of eigenvectors to build the embedded graph and its SpMST was chosen to be equal to the requested number of partitions k because of the requirement to detect quite subtle small clusters for the higher values of k.

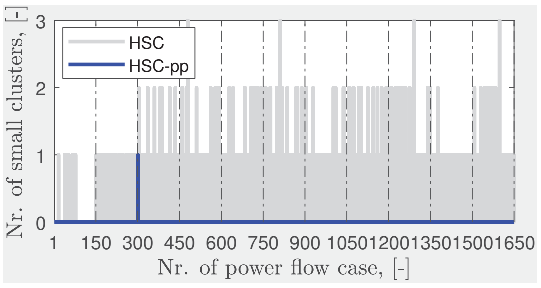
As Figure 4 shows, the described pre-processing caused the HSC method [14] to much less frequently return clusters below 10% of the average cluster size. While there still were a few occurrences of clusters below the specified size threshold, those could be avoided by tuning the parameter ϕ^* specifically



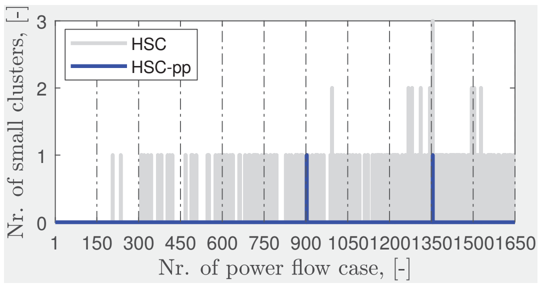
(a) case300 test network



(b) case1354pegase test network



(c) case2383wp test network



(d) case2869pegase test network

Figure 4: Occurrences of clusters below 10% of the average cluster size for the partitioning of randomly generated power flow graphs with an outlier-sensitive method, with (blue) and without (gray) pre-processing of potential small clusters.

TABLE I: Total numbers of small cluster occurrences

Network	*HSC*	*HSC**	*k-means*	*k-means**
case300	253	0	13	0
case1354pegase	755	3	116	0
case2383wp	668	1	10	1
case2869pegase	231	2	1	0

for these test cases. Thus, it can be concluded that the proposed graph outlier detection method is able to reliably estimate the outlier clusters for a given graph (otherwise those clusters would be returned by the graph partitioning method). We have also considered the spectral partitioning with the k-means algorithm [5] as an additional partitioning method. The aggregated results for the HSC method and spectral k-means are given in Table I, with the * superscript denoting the versions of the methods with outlier pre-processing.

B. Computational Performance

In order to test the computational performance of the proposed graph outlier detection method, we have executed it on the six MATPOWER networks ranging from 300 to 13659 nodes. For each network, the number of eigenvectors was varied from three to eight, while the upper cluster size limit was set to $\mathrm{round}(0.1n/5)$ and the threshold of score (7) was set to 0.03. For this case study, the algorithm was run on the active power flow graphs obtained from the nominal loading profile of each network. All results were obtained on MATLAB R2017a (64-bit) on a PC with an Intel® Xeon® E5 3.70 GHz CPU and 16 Gb of RAM on a single core.

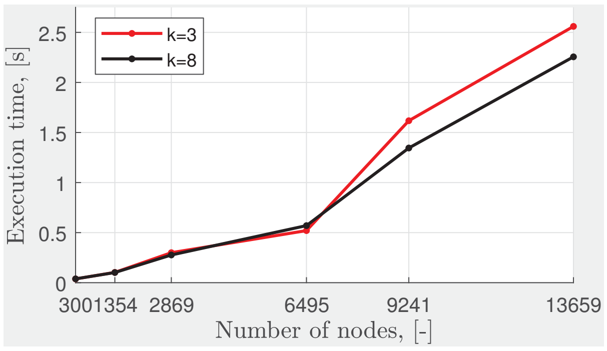


Figure 5: Execution time of the SpMST-based graph outlier detection method for the varying number of network nodes and dimension of spectral embedding.

Surprisingly, choosing a low value for the number of eigenvectors k may result in longer running times. This can be explained by the complex convergence mechanisms of eigenvalue solvers for sparse matrices (e.g., of the Lanczos algorithm [5]). Nevertheless, the execution times in Figure 5 justify the practicality of the proposed graph outlier detection method at least for the networks of several thousands of nodes.

The main computational requirement of the SpMST graph outlier detection method is in computing several largest eigenvalues and eigenvectors of the graph matrix $\mathbf{A_n}$ (3). This procedure is reported to be tractable for sparse graphs of at least tens of thousands of nodes [6]. In practice, computing the eigenvectors took around 50—70% of the total execution time for each network using the highly optimized eigenvalue solver available in MATLAB. Such disproportion in the computation time can also be explained by the efficient algorithmic design of the SpMST-based outlier mining described in Section IV. The disparity between the eigenvector computation time and the SpMST processing time could be even larger if the latter component were also implemented in a high-performance compiled language instead of the current MATLAB-based implementation.

VI. Conclusion

This paper has proposed a method for efficient detection of weakly connected small clusters in power network graphs and general similarity graphs. Such clusters can be considered as outliers in many contexts, including graph clustering, which is the application studied in this paper.

The SpMST graph outlier detection method is based on embedding of the input graph into a low-dimensional Euclidean space by the methods of spectral graph theory. The MST of this graph is examined with the efficient top-down and bottom-up graph clustering algorithms to reveal the weakly connected parts of the graph below certain size. As the final step, the revealed outlier candidates are systematically aggregated. The test results demonstrate the good performance of the proposed method in filtering out the graph outliers as well as its high computational speed.

The proposed algorithm can be useful as a preprocessing or analytic tool for similarity graphs of large-scale power networks. Example applications involving similarity graphs include definition of control zones, clustering of contingencies, power network reduction and others. Various additional criteria besides the maximal cluster size could be added to detect more specific cluster types.

Acknowledgement

This research is financially supported by the Dutch Scientific Council NWO-STW, under the project 408-13-025 within the program of Uncertainty Reduction of Sustainable Energy systems (URSES) in collaboration with TSO TenneT and the Dutch National Metrology Institute, van Swinden laboratory.

References

[1] E. Cotilla-Sanchez, Hines, Paul D. H., C. Barrows, S. Blumsack, and M. Patel, “Multi-attribute partitioning of power networks based on electrical distance,” *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4979–4987, 2013.

[2] H. Mehrjerdi, S. Lefebvre, M. Saad, and D. Asber, “A decentralized control of partitioned power networks for voltage regulation and prevention against disturbance propagation,” *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1461–1469, 2013.

[3] P. Demetriou, J. Quirós-Tortós, E. Kyriakides, and V. Terzija, “On implementing a spectral clustering controlled islanding algorithm in real power systems,” in *10th IEEE POWERTECH*, Grenoble, France.

[4] V. Quintana and N. Müller, “Partitioning of power networks and applications to security control,” *IEE Proceedings C - Generation, Transmission and Distribution*, vol. 138, no. 6, pp. 535–545, 1991.

[5] U. Von Luxburg, “A tutorial on spectral clustering,” *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[6] C. Hennig, M. Meila, F. Murtagh, and R. Rocci, *Handbook of cluster analysis*. CRC Press, 2015.

[7] I. Tyuryukanov, J. Quirós-Tortós, M. Naglic, M. Popov, M. A. van der Meijden, and V. Terzija, “A post-processing methodology for robust spectral embedded clustering of power networks,” in *17th IEEE EUROCON*, pp. 1–7, Ohrid, Macedonia, 2017.

[8] C. C. Aggarwal, *Outlier Analysis*. Springer, 2013.

[9] H. Sun, J. Huang, J. Han, H. Deng, P. Zhao, and B. Feng, “gskeletonclu: Density-based network clustering via structure-connected tree division or agglomeration,” in *10th IEEE ICDM*, pp. 481–490, 2010.

[10] L. Xiong, X. Chen, and J. Schneider, “Direct robust matrix factorizatoin for anomaly detection,” in *11th IEEE ICDM*, pp. 844–853, IEEE, 2011.

[11] H.-P. Kriegel, A. Zimek, *et al.*, “Angle-based outlier detection in high-dimensional data,” in *14th ACM KDD*, pp. 444–452, 2008.

[12] U. Brandes, M. Gaertler, and D. Wagner, “Experiments on graph clustering algorithms,” in *European Symposium on Algorithms 2003, Lecture Notes in Computer Science*, vol. 2832, pp. 568–579, Springer, 2003.

[13] M. Meila and J. Shi, “Learning segmentation by random walks,” in *13th NIPS*, pp. 873–879, 2001.

[14] R. J. Sanchez-Garcia, M. Fennelly, S. Norris, N. Wright, G. Niblo, J. Brodzki, and J. W. Bialek, “Hierarchical spectral clustering of power grids,” *IEEE Transactions on Power Systems*, vol. 29, no. 5, pp. 2229–2237, 2014.

[15] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, “Matpower: Steady-state operations, planning, and analysis tools for power systems research and education,” *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[16] I. S. Dhillon, Y. Guan, and B. Kulis, “Kernel k-means: spectral clustering and normalized cuts,” in *10th ACM KDD*, pp. 551–556, 2004.

[17] R. Sedgewick and K. Wayne, *Algorithms, 4th Edition.* Addison-Wesley, 2011.

[18] M. Laszlo and S. Mukherjee, “Minimum spanning tree partitioning algorithm for microaggregation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 902–911, 2005.

[19] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*. Wiley, 1997.

[20] O. Grygorash, Y. Zhou, and Z. Jorgensen, “Minimum spanning tree based clustering algorithms,” in *18th IEEE ICTAI*, pp. 73–81, 2006.

[21] C. Josz, S. Fliscounakis, J. Maeght, and P. Panciatici, “Ac power flow data in matpower and qcqp format: itesla, rte snapshots, and pegase,” *arXiv:1603.01533*, 2016.