The Incremental Cluster Threshold-Free Cluster Enhancement Algorithm for Functional Connectivity Analysis

Fabricio Cravo[1,3,*], Raimundo Rodriguez[4,5], Alfonso Nieto-Castañón[6,7,8,†], Stephanie Noble[1,2,3,9,†]

[1]Department of Psychology, Northeastern University, Boston, MA, USA
[2]Department of Bioengineering, Northeastern University, Boston, MA, USA
[3]Institute for Cognitive and Brain Health, Northeastern University, Boston, MA, USA
[4]Interdepartmental Neuroscience Program, Yale School of Medicine, New Haven, CT, USA
[5]Yale Biomedical Imaging Institute, Yale School of Medicine, New Haven, CT, USA
[6]Computational Neuroscience Research Lab, Santiago de Compostela, Spain
[7]Department of Speech, Language, and Hearing Sciences, Boston University, Boston, MA, USA
[8]McGovern Institute for Brain Research, MIT, Cambridge, MA, USA
[9]Department of Radiology and Biomedical Imaging, Yale School of Medicine, New Haven, CT, USA
[*] Corresponding author
[†] These authors contributed equally

Abstract

Threshold-free cluster enhancement (TFCE) is one of the most used statistical inference methods in neuroimaging, but its computational cost limits some of its applications. The current implementations recompute clusters at each threshold step, creating computation costs that poorly scale with precision increases. Furthermore, as larger samples and reduced noise increase maximum t-statistics, computational burden grows correspondingly. As the field moves towards finer parcellations, the number of FC edges grows quadratically with the number of ROIs, making TFCE computationally infeasible at the scales increasingly demanded by the field. We present Incremental Cluster TFCE (IC-TFCE), an algorithm that produces numerically equivalent results to standard TFCE while decoupling runtime from discretization precision. The IC-TFCE builds clusters incrementally from previous threshold steps rather than recomputing them, stores TFCE results on a region of interest (ROI) based structure instead of a functional connectivity (FC) edge structure for improved speed, and can be applied to voxel data through a novel graph transformation described and validated herein. This algorithm achieves a measured 3–93× speedup for FC TFCE depending on the precision parameter dh, making TFCE analyses with fine parcellations of 1000 or more ROIs computationally tractable for the first time. Finally, we validate correctness through mathematical proof and numerical comparison. The efficiency provided by IC-TFCE allowed a large-scale empirical power analysis across dh values to guide practitioners in parameter selection for their analyses.

1 Introduction

Cluster-based inference is one of the most widely used methods for detecting brain activation in functional magnetic resonance imaging (fMRI) [WKW14, Yeu18, LSL+18]. Real brain activations are spatially correlated, and cluster methods use this spatial property to improve detection [FWF+94, PWEF97, HSY+06, Fri00, WLK07, WKW14]. This clustering is observed in voxel activation patterns [WKW14] and, through their correlations, mathematically implied in functional connectivity data [ZFB12]. The high noise inherent to fMRI [Liu16, BFvG+09, VMD+21] requires improvements in statistical methods to more reliably detect true effects at realistic sample sizes [SN09, DG02, Bea10].

Two widely used cluster inference methods are cluster size [FWF+94] testing and threshold-free cluster enhancement (TFCE) [SN09, SKSN11, SSZ+19]. These methods operate on adjacent variables: voxels in activation analyses or edges in FC analyses. For both approaches, a cluster refers to a group of spatially adjacent voxels (for activation data) or connected edges (for FC data) that all exceed a statistical threshold. The cluster size method works as follows: First, a significance threshold is applied to individual variables, forming clusters. Then, a null distribution is estimated by computing

the maximum cluster size across permutations of the data. Observed clusters larger than those expected under the null distribution are deemed significant.

The TFCE [SN09] is similar to the cluster size, but it eliminates the need for an arbitrary threshold and has been shown to be a more powerful approach [NSC20, NMZS22]. Instead of testing at a single threshold, TFCE integrates across many thresholds. At each threshold level, it computes the size of the cluster that each variable is contained in. The final TFCE value is a combination of its own statistical strength and the cluster sizes of the clusters this variable is contained in at each threshold level in the form of an integral. In practice, this integral is computed using a discrete approximation to reduce computational burden [JBB+12, OFMS11, GLL+13, OCH16, WRW+14]. The approximation uses a step size parameter dh. The maximum t-statistic t_{max} divided by dh determines the number of threshold steps $n_{th} = t_{max}/dh$, with the clusters forming all variables being computed at each threshold step.

However, the TFCE is among the most computationally demanding statistical inference procedures in neuroimaging [SSZ+19, FCH+22], with documented computational burden limiting its application in some analyses [APE+14, SSZ+19, FCH+22]. Even though some of these limitations address voxel applications, voxel data has at most 26 neighbors per voxel, resulting in a bounded connectivity that greatly reduces computational complexity compared to FC data, where graphs are complete. This becomes particularly relevant for analyses with high t_{max} values, whether from large sample sizes, high-quality data with reduced noise, or strong effects [BIM+13, PBD+17, SI20], and for applications requiring finer precision (lower dh).

As the field moves towards finer parcellations with greater numbers of ROIs [SKG+18, EYG18], the number of FC edges grows quadratically with the number of nodes, rapidly making TFCE computationally infeasible. With 1000 ROIs, for instance, a single FC matrix already contains 499,500 edges. Since TFCE is a permutation-based method requiring on the order of 1000 permutations per analysis [WRW+14], the computational burden at this scale becomes prohibitive [SSZ+19, FCH+22]. If neuroimaging studies continue to move towards even finer parcellations, this problem will worsen, and it will restrict the TFCE application.

Another issue with the current discretized implementation is that it creates a speed-precision tradeoff [PLNR15]. Practitioners must choose between using larger dh values for faster computation or smaller dh values for higher precision. Smaller dh requires more threshold steps and therefore more cluster recomputations. While the impact of dh [SN09] was already partially analyzed, this analysis was of limited scope and did not access how it affects statistical power empirically. Some applications could require lower dh values than those commonly adopted, but the computational cost makes this impractical. The CONN toolbox [WGNC12] provides an Exact TFCE implementation that eliminates discretization entirely, making dh irrelevant. However, this implementation is slower than the discretized TFCE. For sufficiently small dh, the precision gains over the discretized TFCE become negligible, yet the Exact implementation cannot take advantage of this speed gain. The field needs an algorithm that provides both speed and precision without forcing a tradeoff.

In this work, we present the Incremental Cluster TFCE (IC-TFCE). The IC-TFCE produces identical results to standard TFCE with a computational speedup that increases as the dh becomes smaller. Rather than recomputing all clusters at each threshold step, IC-TFCE builds clusters incrementally from previous iterations. For FC data, we also introduce a node accumulation data structure that enables efficient storage and retrieval of TFCE values for each edge. These strategies reduce computational complexity from $O(N^2 n_{th})$ to $O(N^2 + n_{th}N)$, where N is the number of regions of interest (ROIs). Empirically, we observe 3×-93× speedups on FC data depending on the dh parameter, type of data, and parcellation used, with greater speedups at smaller dh values and with a greater number of nodes. Finally, by reducing computational complexity from $\mathcal{O}(N^2 n_{th})$ to $\mathcal{O}(N^2 + n_{th}N)$, IC-TFCE restores the viability of TFCE-based inference at the finer parcellation scales increasingly demanded by the field.

We provide rigorous validation through mathematical proof and numerical comparison with standard TFCE. To guide parameter selection, we conduct a power analysis across different dh values, quantifying the precision-power tradeoff that was previously uncharacterized. We provide pseudocode for IC-TFCE implementation, including a graph transformation for voxel data. Finally, we formally describe and validate the Exact TFCE algorithm used in the CONN toolbox, which has not been previously documented in the literature, and provide its pseudocode with incremental clustering strategies.

2 Results

2.1 The Algorithm

2.1.1 The Original TFCE Algorithm

The TFCE algorithm tests for the statistical significance of variables, either individual voxels or FC edges, based on their ability to form clusters. It does so by transforming each variable's test statistic (e.g., t-statistics from group comparisons or GLM contrasts) by integrating over cluster sizes at multiple thresholds. Each edge's TFCE value is computed by integrating the size of clusters it belongs to across threshold levels, giving higher values to edges that form larger connected components.

The TFCE integral requires two parameters: (1) E, which weights the extent of clustering, (2) H, the height exponent that weights the threshold levels up to each variable's statistic value. The algorithm's integral transformation is:

$$\mathrm{TFCE}(x) = \int_{h=0}^{h_{\max}} c(x,h)^E \cdot h^H \, dh \tag{1}$$

where $c(x,h)$ is the number of variables (fc edges or voxels) in the cluster containing variable x whose test statistic is above threshold h.

Normally, this integral is not computed in the previous form, and it is computed with an approximation as a discrete sum:

$$\mathrm{TFCE}(x) = \sum_{h=0}^{h_{\max}} c(x,h)^E \cdot h^H \cdot dh \tag{2}$$

To perform this integration numerically, a new parameter is introduced, dh, the step size for discretizing the integral. Like E and H, the choice of dh has historically been empirical, with numerical accuracy of the approximation being the only consideration.

2.1.2 Variables and Topology

The TFCE algorithm is used to two distinct types of neuroimaging data, voxel and functional connectivity, with varying ways of implementation. Furthermore, beyond the data types, the definition of what a cluster (topology) is varies substantially. In this section, we clearly define for which applications the IC-TFCE is oriented towards by explaining the variables and topology used herein.

For voxel activation data, adjacency is defined geometrically: Two voxels are neighbors if they occupy spatially contiguous positions in the brain volume, with varying degrees of neighborhood ranging from 6-connectivity ((side-sharing) to 26-connectivity (side-, diagonal-sharing) in three dimensions [SN09].

For FC data, the situation is more complex. Each variable is an edge in a connectivity graph and two edges can be declared neighbors in different ways. The Spatial Pairwise Clustering (SPC) [ZFB12] TFCE defines adjacency between edges based on the anatomical proximity of their endpoint ROIs: edges that are connected to spatially adjacent ROIs in the brain are considered neighbors. Another approach, introduced by the Network Based Statistic (NBS) [ZFB10] and later extended to TFCE by Baggio et al. [BAS+18a], defines two edges as neighbors if they share at least one ROI. Under this node-sharing adjacency, clusters are connected subgraphs of the original ROI graph, and cluster membership for any edge is fully determined by the cluster memberships of its two endpoint nodes.

The IC-TFCE is designed primarily for the NBS-TFCE and can be extended for the voxel TFCE. Therefore, in this manuscript, when we refer to topology for voxels, we refer to their spatial contiguity, and for FC, it signifies that both edges are connected to a common ROI.

2.1.3 The IC-TFCE Algorithm

The IC-TFCE algorithm is simply an alternative way to compute the same TFCE algorithm that yields equal results, but uses the clusters from previous integral thresholds to build the clusters from the following threshold, creating a faster algorithm. The approach processes the clusters from the highest threshold to the lowest, as at the maximum threshold, every variable test statistic is below the threshold, so all clusters are empty. At each step, we add the variables whose test statistics fall between $h_{n-1} = (n-1)dh$ and $h_n = ndh$ to compute the next cluster. The incremental merging can

be implemented with Union-Find. However, Union-Find provides no complexity gain, as it is required to check the sizes of each cluster for the TFCE computation at each discrete step.

For FC data, to achieve faster complexity, we introduce a node accumulation structure comprising two matrices: (1) **S**, storing cluster sizes as the number of edges at each threshold for each node, and (2) **F**, storing the cumulative TFCE integral values. The structure **S** is populated during cluster merging, then **F** is computed from **S** through a sum. Finally, each edge retrieves its TFCE value in O(1) time by indexing **F** at its threshold bin and either endpoint node. The pseudocode for this implementation is provided in Algorithm 1 with the supporting Figure 1. In Supplementary Section 5, we also prove the following proposition that assures correctness:

Proposition 2.1. *For all discrete integration steps, the IC-TFCE algorithm computes equal clusters to the TFCE.*

For voxel analysis, we provide a graph transformation that converts spatial adjacency into an edge-based representation regardless of dimension. Each edge connects two spatially adjacent voxels and receives the minimum of their test statistics. This allows IC-TFCE to be computed on voxel data, with the only difference being that the TFCE can be calculated directly after each bin is added, without the need for the extra accumulation and edge retrieval steps. The graph transformation is provided in Algorithm 2 and the proof that transformation is indeed correct can be found in the Supplementary Section 6.

For the code implementation of the graph transformation , IC-TFCE for FC, and IC-TFCE voxels, please refer to the PRISME power calculator [CFS+26].

2.1.4 The Exact TFCE Implementation

The CONN toolbox provides an exact TFCE implementation for FC data. Herein, we provide a pseudocode implementation of it using an incremental cluster strategy.

The Exact TFCE works by sorting the variables' test statistics and varying the dh step according to the sorted variables. Since for h any between the two test statistics of two sorted variables, the function $c(x, h)$ remains constant for all variables, one can perform the following integration in piecewise steps.

Let $h_{s(0)} = 0 < h_{s(1)} < h_{s(2)} < h_{s(3)} < .... < h_{s(n_v)}, h_{s(i)} \in \mathbb{R}$ be the sorted test statistics of the variables and $n_v \in \mathbb{N}$ the number of variables:

$$\text{TFCE}(x) = \sum_{i=1}^{n_v} c(x, h_{s(i)})^E \int_{h_{s(i-1)}}^{h_{s(i)}} \cdot h^H \, dh \tag{3}$$

Essentially, the cluster size is constant at those sequential intervals, which allows its removal from the integral and its calculation.

The pseudocode for the Exact TFCE algorithm is presented in Algorithms 4 and 3. Rather than sweeping over all nodes at each edge step, the Exact TFCE maintains a running score per cluster, updated whenever a cluster is accessed. Each cluster tracks its last processed height, so the accumulated TFCE contribution over any interval can be computed in $O(1)$ via FlushCluster. When an edge enters, its starting value is recorded as the current cluster score and the final TFCE for that edge is then recovered by walking up the parent chain from its entry cluster and subtracting this starting value. This approach stores one score value per edge rather than an $n_{th} \times N$ accumulation matrix, making it more memory efficient than IC-TFCE when $n_{th} > N$. However, each cluster merge creates a new cluster object, and all node labels in the merged cluster must be updated to point to the new cluster, carrying a higher per-merge cost than the union-by-size strategy used in IC-TFCE. The IC-TFCE can be implemented with a similar strategy.

2.2 Computational Complexity

2.2.1 The IC-TFCE Implementation

Note as N the number of ROIs. Since a functional connectivity graph is complete, bidirectional and there are no edges from nodes to themselves, the number of edges E is $E = \frac{N(N-1)}{2}$. Let n_{th} be the number of threshold bins which is dh divided by the maximum t-stat in the graph.

From the pseudocode, the IC-TFCE complexity is the following:

Algorithm 1 IC-TFCE for FC Data

Require: Adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ ▷ n is the number of nodes
Require: Parameters: dh (step size), H (height exponent), E (extent exponent)
Ensure: TFCE-transformed matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$

$t_{\max} \leftarrow \max(\mathbf{A})$
$\mathcal{T} \leftarrow \{0, dh, 2dh, \ldots, t_{\max} + dh\}$ ▷ Threshold levels
$m \leftarrow |\mathcal{T}|$ ▷ Number of thresholds
Initialize $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m \leftarrow \emptyset$ ▷ Edge sets by threshold

for each edge (i, j) where $i < j$ and $\mathbf{A}[i, j] > 0$ **do**
  $r \leftarrow \lfloor \mathbf{A}[i, j]/dh \rfloor + 1$ ▷ Round index
  $\mathcal{E}_r \leftarrow \mathcal{E}_r \cup \{(i, j)\}$ ▷ Assign edges to threshold rounds
end for

for $k = 1$ to n **do**
  $c_k \leftarrow (\text{nodes} : \{k\}, \text{nodeSize} : 1, \text{edgeSize} : 0)$ ▷ Initialize Clusters
  $clusterLabel[k] \leftarrow c_k$ ▷ Labels to find nodes
end for

$\mathbf{S} \leftarrow \mathbf{0}_{m \times n}$ ▷ Matrix storing TFCE for m bins and n nodes

// Main loop: Process thresholds from highest to lowest
for $h = m$ downto 2 **do**
  // Process new edges at threshold h
  for each edge $(i, j) \in \mathcal{E}_h$ **do**
    Let c_i, c_j be the clusters containing nodes i, j ▷ Get from clusterLabel
    if $c_i \neq c_j$ **then** ▷ Merge different clusters
      $\textsc{Merge}(c_i, c_j)$ ▷ Merge smaller into larger
      $\textsc{UpdateLabel}(clusterLabel, i, j, c_i, c_j)$ ▷ Update labels for nodes
    else ▷ Same cluster, just increment edge count
      $c_i.\text{edgeSize} \leftarrow c_i.\text{edgeSize} + 1$
    end if
  end for

  // Record cluster sizes for all nodes
  for $v = 1$ to n **do**
    $\mathbf{S}[h, v] \leftarrow clusterLabel[v].\text{edgeSize}$ ▷ Store cluster sizes for each threshold
  end for
end for

$\mathbf{F} \leftarrow \mathbf{0}_{m \times n}$ ▷ Cumulative TFCE per node
for $h = 2$ to m **do**
  $\mathbf{F}[h, :] \leftarrow \mathbf{F}[h-1, :] + \mathbf{S}[h, :]^E \cdot \mathcal{T}[h]^H \cdot dh$ ▷ Calculate cumulative TFCE values
end for

// Each edge finds its TFCE value from its respective activation point and node
Initialize $\mathbf{T} \in \mathbb{R}^{n \times n}$ to zeros
for $h = 2$ to m **do**
  for each edge $(i, j) \in \mathcal{E}_h$ **do**
    $\mathbf{T}[i, j] \leftarrow \mathbf{F}[h, i]$
    $\mathbf{T}[j, i] \leftarrow \mathbf{F}[h, i]$ ▷ Ensure symmetry
  end for
end for
return T

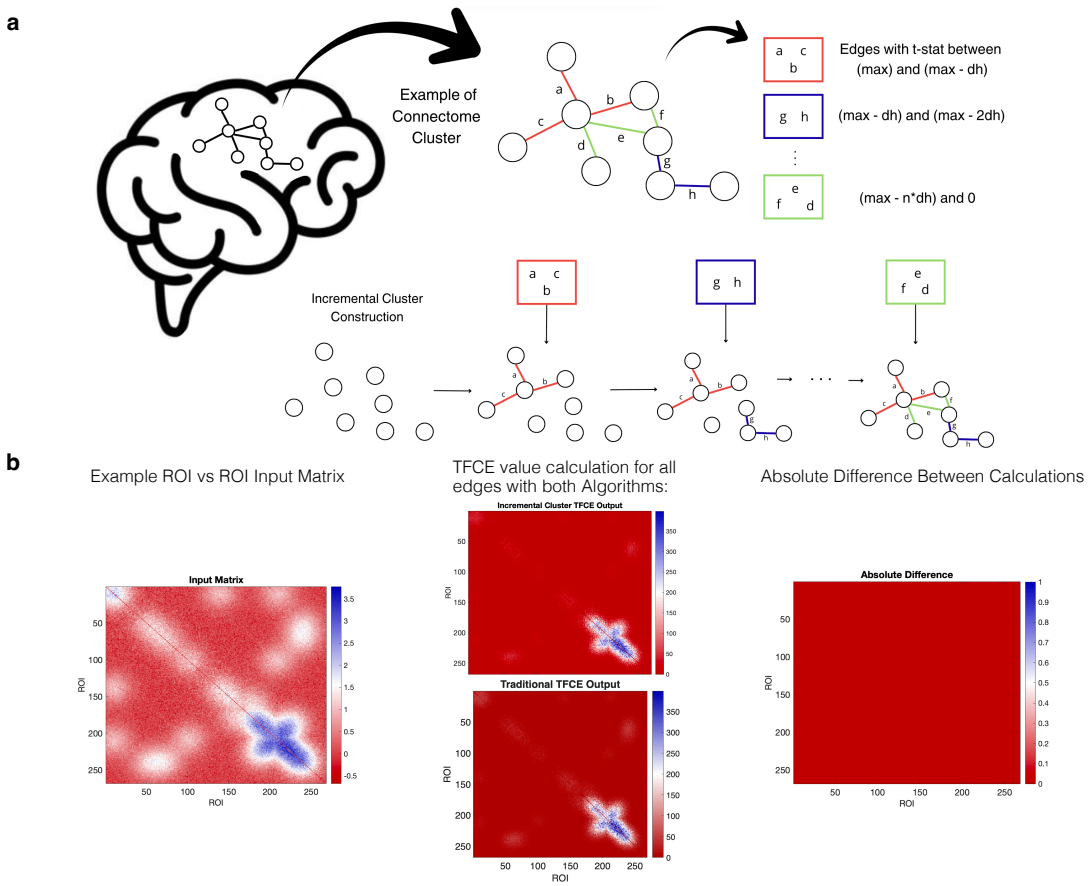

Figure 1: **TFCE algorithm comparison and validation.** **(a)** Figure describing the IC-TFCE Algorithm through a visual application using an example connectome cluster from functional connectivity data. Edges are categorized by t-statistic threshold ranges (red: maximum to max-dh; blue: max-dh to max-2dh; green: max-n*dh to 0). The edges then form a cluster incrementally for the TFCE computation. **(b)** Numerical comparison between TFCE implementations. Left: Example ROI-to-ROI input matrix from resting-state functional connectivity. Middle: TFCE values computed using the IC-TFCE and the TFCE show nearly identical results. Right: Absolute difference between calculations demonstrates numerical equivalence (difference < 0.001 for all edges).

- Assigning each edge to their respective bin $\mathcal{O}(N^2)$
- The IC-TFCE algorithm processes each edge once and it is of $O(N^2)$ order. For each edge, either: the edge connects nodes already in the same cluster and is processed in $O(1)$ time, or it connects nodes in different clusters, triggering a merge. Since each merge reduces the cluster count by one and we begin with N singleton clusters, at most $N-1$ merges occur across the entire algorithm. Since we merge smaller clusters into larger ones, in the best case the largest cluster absorbs a single node at each step for a total merge cost of $O(N)$.

  Differently than the Union-Find algorithm [Tar75, TvL84, CF07], we update cluster labels for the entire cluster at each merger as they are necessary for the cluster size calculation for each threshold.

  On the worst case merge, clusters start as single nodes and double in size every time they merge. A level number l indicates the number of the stage where clusters have size 2^l. At level l, there are $N/2^l$ clusters of size 2^l, which go through $N/2^{l+1}$ merges where each merge costs $O(2^l)$. The cost at each level is therefore $\frac{N}{2^{l+1}} \cdot 2^l = \frac{N}{2}$. Since there are $\log_2 N$ levels, the total merge cost is:

$$\sum_{h=0}^{\log_2 N} \frac{N}{2} = \frac{N}{2} \cdot \log_2 N = O(N \log N) \tag{4}$$

Algorithm 2 Graph conversion

Require: Voxel activation matrix V
Require: Logical mask with active voxel positions M
Ensure: Sparse graph representation $\mathbf{G}$ of brain voxel data

```
n_act_v ← |{v : M[v] = 1}|                    ▷ Number of populated indexes in the voxel matrix
G ← ∅                  ▷ Sparse representation of the graph - First Node, Second Node, Edge Value
for i = 1 to n_act_v do
    i_m ← convert_linear_to_spatial_index(i, V)      ▷ Bijective function - converts linear to spatial
    v_i ← V[i_m]

    for i_adj each adjacent index to i_m do
        j ← convert_spatial_to_linear_index(i_adj, V)      ▷ Inverse of convert_linear_to_matrix_index
        if j > i then
            continue      ▷ Skip repeated edges, notice that i = j saves the each voxel's own t-stat
        end if
        v_j ← V[i_adj]
        v ← min(v_i, v_j)
        G ← G ∪ {(i, j, v)}
    end for
end for
return G
```

Since we process each edge once, the total cost over all edges is:

$$\sum_{e=1}^{N^2} \Big(O(1) + O(\text{partial merge cost for } e) \Big) = O(N^2) + O(N \log N) = O(N^2) \tag{5}$$

where the partial cost for merging is zero when an edge e simply gets added to the cluster with no merger.

At the end of each threshold step, we record the cluster size for each of the N nodes, for a total cost of $O(n_{th} \cdot N)$. Therefore, the cost of the

$$\mathcal{O}(N^2 + n_{th} \cdot N) \tag{6}$$

- Accumulating the TFCE value for each node for each threshold bin is performed in $\mathcal{O}(n_{th}N)$ time.
- Finding the TFCE value for each edge for each edge is performed with a single check per edge by checking the structure that accumulated the cluster size in $\mathcal{O}(N^2)$.

Therefore, by summing through all complexities, the overall complexity of the IC-TFCE is $\mathcal{O}(N^2 + n_{th}N)$. On the other hand, the traditional TFCE algorithm computational complexity is $\mathcal{O}(N^2 n_{th})$.

2.2.2 The IC-TFCE Exact Implementation

The IC-TFCE accumulation strategy can also be applied to compute the Exact TFCE. Rather than using the discretized thresholds, the edges are sorted by their test statistics and each edge defines a single threshold bin. The number of threshold bins is therefore equal to the number of edges, $n_{th} = \frac{N(N-1)}{2}$, which is of $O(N^2)$ order. Since each edge's TFCE value depends only on the cumulative integral at its two endpoint nodes, the accumulation structure can be replaced by storing the initial TFCE value at the time each edge enters and subtracting it from the final cumulative value at the corresponding node. The overall complexity then follows directly from the IC-TFCE formula:

$$\mathcal{O}(N^2 + n_{th}N) = \mathcal{O}(N^2 + N^2 N) = \mathcal{O}(N^3) \tag{7}$$

Algorithm 3 Exact TFCE Helper Functions

function NewCluster(nodes, h)
  c.nodes $\leftarrow$ nodes; c.edgeSize $\leftarrow 0$; c.lastHeight $\leftarrow h$; c.score $\leftarrow 0$; c.parent $\leftarrow$ `None`
  return c
end function
function FlushCluster(c, h)
  $c.\text{score} \mathrel{+}= c.\text{edgeSize}^{E} \cdot \dfrac{c.\text{lastHeight}^{H+1} - h^{H+1}}{H+1}$; c.lastHeight $\leftarrow h$
  return c
end function
function AddToNewCluster(i, j, h, clusterLabel)
  $c \leftarrow$ NewCluster($\{i, j\}$, h); c.edgeSize $\leftarrow 1$
  clusterLabel[i] $\leftarrow c$; clusterLabel[j] $\leftarrow c$
  return $\big((c,\ 0),\ \text{clusterLabel}\big)$
end function
function AddToExistingCluster(i, j, c_i, c_j, h, clusterLabel)
  $c \leftarrow c_i$ if $c_i \neq$ `None` else c_j; $v \leftarrow j$ if $c_i \neq$ `None` else i
  $c \leftarrow$ FlushCluster(c, h)
  c.nodes $\leftarrow c.\text{nodes} \cup \{v\}$; c.edgeSize $\mathrel{+}= 1$
  clusterLabel[v] $\leftarrow c$
  return $\big((c,\ c.\text{score}),\ \text{clusterLabel}\big)$
end function
function MergeClusters(c_i, c_j, h, clusterLabel)
  $c_i \leftarrow$ FlushCluster(c_i, h); $c_j \leftarrow$ FlushCluster(c_j, h)
  $c \leftarrow$ NewCluster($c_i.\text{nodes} \cup c_j.\text{nodes}$, h)
  c.edgeSize $\leftarrow c_i.\text{edgeSize} + c_j.\text{edgeSize} + 1$
  c_i.parent $\leftarrow c$; c_j.parent $\leftarrow c$
  clusterLabel[v] $\leftarrow c$ for all $v \in c$.nodes
  return $\big((c,\ 0),\ \text{clusterLabel}\big)$
end function
function AddToSameCluster(c_i, h)
  $c_i \leftarrow$ FlushCluster(c_i, h); c_i.edgeSize $\mathrel{+}= 1$
  return $\big((c_i,\ c_i.\text{score}),\ c_i\big)$
end function

Algorithm 4 Exact TFCE

Require: Adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
Require: Parameters: H (height exponent), E (extent exponent)
Ensure: TFCE-transformed matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$

```
ℰ ← all edges (i, j) where i < j and A[i, j] > 0, sorted by A[i, j]
m ← |ℰ|
clusterLabel[k] ← None for k = 1 to n;
start[i, j] ← (None, 0) for all i, j
for r = m downto 1 do
    (i, j) ← ℰ[r]
    h ← A[i, j]
    c_i ← clusterLabel[i];    c_j ← clusterLabel[j]
    if c_i = None and c_j = None then
        (start[i, j], clusterLabel) ← AddToNewCluster(i, j, h, clusterLabel)
    else if c_i = None or c_j = None then
        (start[i, j], clusterLabel) ← AddToExistingCluster(i, j, c_i, c_j, h, clusterLabel)
    else if c_i ≠ c_j then
        (start[i, j], clusterLabel) ← MergeClusters(c_i, c_j, h, clusterLabel)
    else
        (start[i, j], c_i) ← AddToSameCluster(c_i, h)
    end if
    start[j, i] ← start[i, j]
end for

for each unique cluster c in clusterLabel do
    FlushCluster(c, 0)                                   ▷ Flush to h = 0
end for

Initialize T ∈ ℝ^{n×n} to zeros
for r = 1 to m do
    (i, j) ← ℰ[r]
    (c_entry, s_entry) ← start[i, j]                     ▷ Cluster and its score when edge entered
    total ← 0;    c ← c_entry
    while c ≠ None do
        total += c.score;    c ← c.parent
    end while
    T[i, j] ← total − s_entry
    T[j, i] ← T[i, j]
end for
return T
```

2.2.3 Voxel Transformation Complexity

The graph transformation from voxel data requires iterating over all voxels and their adjacency relationships, yielding a complexity of:

$$O(N + M) \tag{8}$$

where N is the number of voxels and M is the number of adjacency links between neighboring voxels. In voxel data, each voxel has a bounded degree with at most 26 neighbors under full 3D adjacency. Therefore, $M = O(N)$, and the transformation complexity becomes:

$$O(N + M) = O(N) \tag{9}$$

For voxel data, the accumulation structure is no longer necessary. Cluster sizes are defined by the nodes themselves and the TFCE values are updated after each merge. Following analogous analysis to the FC case, the IC-TFCE complexity for voxel data comprises:

- Edge bin assignment: $O(M) = O(N)$
- Processing edges and merges: $O(M + N \log N) = O(N \log N)$
- Recording cluster sizes at each threshold step: $O(n_{th} \cdot N)$

The overall complexity of IC-TFCE for voxel-based analysis is therefore:

$$O(N \log N + n_{th} N) \tag{10}$$

Due to the bounded number of edges, traditional TFCE on voxel data has complexity $O(N \cdot n_{th})$, which is asymptotically equivalent as $\log_2 N \leq n_{th}$ for the majority of cases. For example, with $N = 500\,000$ voxels, $\log_2 N \approx 19$, while even with coarse precision ($t_{max} = 10$, $dh = 0.25$), $n_{th} = 40$. The graph transformation therefore provides a unified implementation framework for both FC and voxel analyses using the IC-TFCE. However, the IC-TFCE benefit is clearly stronger in FC-based analysis.

From the mathematical proofs in Supplementary Section 6, the transformation still remains valuable to allow for graph based algorithms to be applied directly to voxel data, as clusters in this graph have exactly the same voxels as the clusters as defined from voxel data. Furthermore, this allows the Exact TFCE to be used in voxel analysis.

2.3 Numerical Simulations

Even though the IC-TFCE has better computational complexity than the traditional TFCE implementation, it does not necessarily imply the algorithm is faster in practical applications. The IC-TFCE has more overhead costs related to preparing data structures in the form of the edge bin assignment and preparing the node accumulation structure.

To guarantee the IC-TFCE provides a substantial speed gain, specially for cases where dh is smaller, we measure the computational gain in speed under the same populational conditions. Since the t-statistics of the edges themselves affects the execution speed of the TFCE algorithm, we perform an analysis for multiple subject group numbers (20, 80, 200) over five different tasks of the HCP dataset: working memory, motor, language, social, and relational [CFS+26]. As the idea is to focus on speed difference instead of inference, only 50 permutations are used and 10 repetitions for each task for each subject group number.

As we can see (Table 1, the IC-TFCE implementation results has the lowest mean time execution across all tasks on the HCP dataset and this result can be observed over multiple subject sizes and over the range of currently recommended dh values used in literature. Even for the highest dh of 0.25, we still observe a significant $x3$ times speedup. The Exact TFCE, although it yields the most precise computation of all TFCE algorithms, is by far the slowest one.

To evaluate how computational gains scale with the number of ROIs, we conducted an additional benchmark using synthetic data. The ROI-by-ROI t-statistic matrices were generated for varying numbers of ROIs by sampling t-statistics from a standard normal distribution and applying a 2D Gaussian filter to introduce spatial correlation structure expected in fMRI. Table 2 presents the runtime comparison between IC-TFCE and traditional TFCE across ROI counts ranging from 200 to 1000, demonstrating that IC-TFCE's computational advantage grows with the number of ROIs.

Table 1: Average runtime comparison of TFCE implementations across varying sample sizes for a group-level analysis. All algorithms were implemented in C++ to ensure fair performance comparison. Values represent mean execution times per permutation, averaged across five tasks and 10 repetitions from the HCP dataset (500 total executions per condition).

(a) Subject Numbers = 20

dh	IC-TFCE (ms)	TFCE (ms)	Speedup
0.01	4.024	133.186	33.10x
0.05	2.347	30.037	12.80x
0.1	2.243	16.424	7.32x
0.25	2.238	7.228	3.23x

(b) Subject Numbers = 80

dh	IC-TFCE (ms)	TFCE (ms)	Speedup
0.01	3.890	121.781	31.30x
0.05	2.395	27.794	11.61x
0.1	2.115	14.607	6.90x
0.25	2.013	6.587	3.27x

(c) Subject Numbers = 200

dh	IC-TFCE (ms)	TFCE (ms)	Speedup
0.01	3.946	121.847	30.88x
0.05	2.522	27.601	10.94x
0.1	2.264	14.578	6.44x
0.25	2.027	6.606	3.26x

Table 2 shows that IC-TFCE benefits scale with parcellation sizes. Speedups increase substantially with the number of nodes, reaching up to 93.9× at $N = 1000$ ROIs with $dh = 0.01$, compared to 7.6× at $N = 200$ with $dh = 0.25$. As expected from the complexity analysis, finer dh values yield the greatest relative gains, since the traditional TFCE runtime grows proportionally with n_{th} while IC-TFCE does not. Even the most conservative case ($dh = 0.25$) delivers consistent speedups over all analysis. Given that larger parcellations and finer precision are precisely the conditions under which TFCE has historically been computationally slower, IC-TFCE makes previously infeasible analyses tractable without any sacrifice in numerical accuracy.

2.4 Power Analysis

With our algorithmic optimizations, we conducted a large scale empirical power analysis using data from the Human Connectome Project [VESB+13]. For functional connectivity analysis, we tested five task contrasts across varying sample sizes ($n_{sub} = 20, 40, 80, 120, 200$) and four dh values ($0.01, 0.05, 0.1, 0.25$). Statistical power was calculated using the PRISME [CFS+26] power calculator. We examined two power metrics: (1) overall average power, calculated as the mean power across all edges in the connectivity matrix, and (2) network-based average power, calculated as the mean power across-task of the top ranked networks in in-task power. For voxel activation analysis, we performed similar comparisons using the same dh parameters and 3 sample sizes ($n_{sub} = 20, 80, 200$). Results demonstrated that statistical power increased with sample size for all dh values, but differences between dh parameters were non-existent, particularly at larger sample sizes. Both overall average power and network-based average power showed consistent patterns across different dh values at $n_{sub} = 80$ and $n_{sub} = 200$, with only modest variations observed at the smallest sample size ($n_{sub} = 20$).

3 Methods and Data Processing

The power analysis was performed using the PRISME empirical power calculator [CFS+26] with $E = 0.5$ and $H = 2$ for voxel data and $E = 0.4$ and $H = 3.0$ for FC data. Power analysis used 800 permutations for the TFCE calculations and 100 repetitions for calculating power. The 800 permutation usage is close enough in precision to the 1000 normally used and it is used to speed up

Table 2: Average runtime comparison of TFCE implementations across varying numbers of ROIs on synthetic data. Synthetic t-statistic matrices were generated by sampling from a standard normal distribution, applying a 2D Gaussian filter (zero-padded, σ scaled proportionally to N) to introduce spatial correlation structure, and symmetrizing the result. Times represent mean execution across 500 repetitions. Spatial correlation structure size was kept constant across different parcellations. All algorithms were implemented in C++.

(a) N = 200 ROIs

dh	IC-TFCE (ms)	TFCE (ms)	Speedup
0.01	0.9	33.2	35.3x
0.05	0.3	6.6	20.9x
0.1	0.2	3.3	14.4x
0.25	0.2	1.4	7.6x

(b) N = 500 ROIs

dh	IC-TFCE (ms)	TFCE (ms)	Speedup
0.01	2.8	183.7	64.5x
0.05	1.3	36.9	28.9x
0.1	1.1	18.7	17.4x
0.25	0.9	7.5	7.9x

(c) N = 1000 ROIs

dh	IC-TFCE (ms)	TFCE (ms)	Speedup
0.01	7.5	703.6	93.9x
0.05	4.1	135.6	33.0x
0.1	3.6	68.3	18.7x
0.25	3.4	28.7	8.3x

the large scale analysis. The 100 repetitions usage provides approximately ±5% precision on power estimates, sufficient for practical sample size planning. From PRISME algorithm, all TFCE implementations were compared across identical permutation and subject data for each repetition, relative power differences between dh values are estimated with higher precision.

Functional images were motion-corrected using SPM5. The data were then iteratively smoothed to an equivalent smoothness of a 2.5 mm Gaussian kernel in order to ensure uniform smoothness across the dataset. White matter and CSF were defined on a MNI-space template brain and eroded in order to minimize inclusion of grey matter in the mask. The template was then warped to subject space using a series of transformations described in the next section. This ensured that mainly grey matter voxels were used in subsequent analyses. The following noise covariates were regressed from the data: linear, quadratic, and cubic drift, a 24-parameter model of motion, mean cerebrospinal fluid signal, mean white matter signal, and mean global signal. Finally, data were temporally smoothed with a zero mean unit variance Gaussian filter (cutoff freq=0.19 Hz). Anatomical data were first skull-stripped using FSL. Functional data for each subject, scanner, and session were linearly registered to the corresponding FLASH images. FLASH images were then linearly registered to MPRAGE images. Next, an average MPRAGE image for each subject was created by linearly registering and averaging all 4 anatomical images (from 2 scanner × 2 sessions) for each subject. These average MPRAGE images were used for an iterative nonlinear registration to MNI space. The use of the average anatomical images and a single nonlinear registration for each subject ensures that any potential anatomical distortions due to the different scanners does not introduce a systematic bias into the registration. The average MPRAGE images were nonlinearly registered to an evolving group average template in MNI space as described previously. All transformation pairs were calculated independently and then combined into a single transform that warps single participant results into common space. From this, all subjects' images can be transformed into common space using a single transformation, which reduces interpolation error. [NST+17]

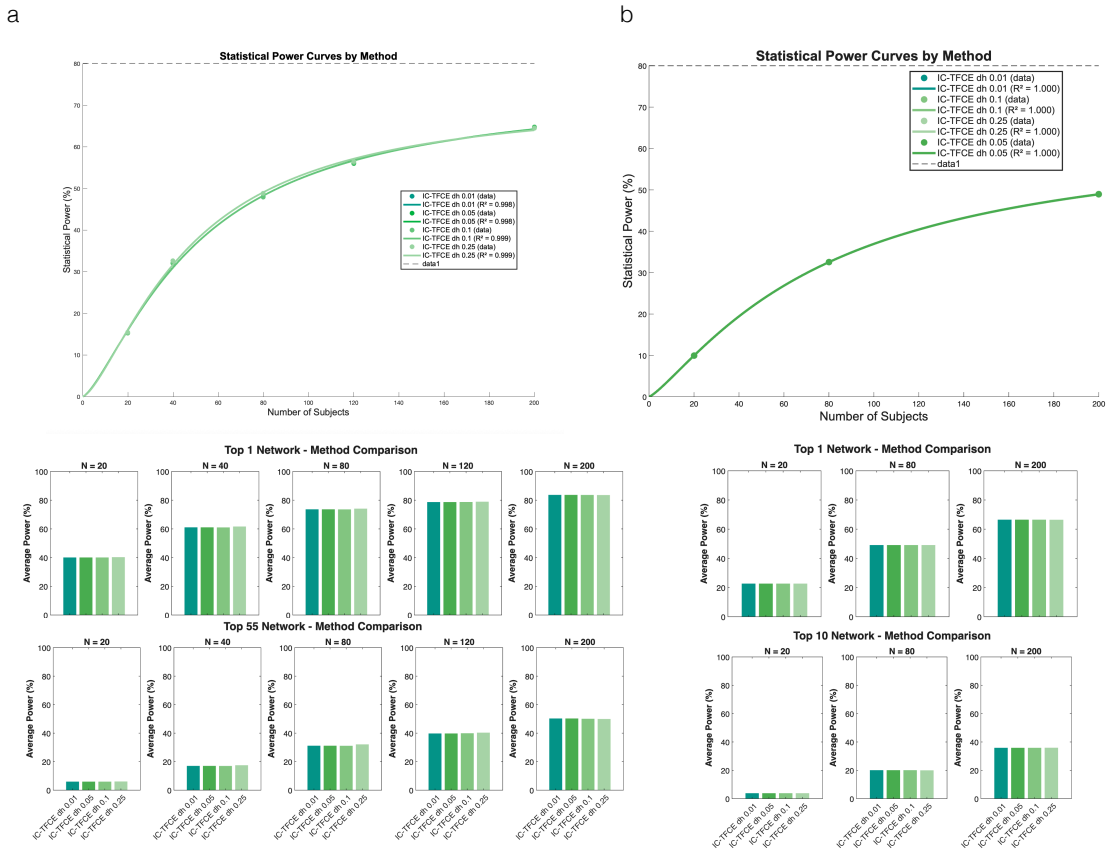

Figure 2: **Statistical power comparison of IC-TFCE implementations across different step sizes (dh).**: (a) Functional connectivity power analysis showing statistical power curves (top) and average power for top networks (middle and bottom) across varying sample sizes (N = 20, 40, 80, 120, 200) and dh values (0.01, 0.05, 0.1, 0.25). (b) Voxel-based power analysis demonstrating similar trends for IC-TFCE with different dh parameters (0.01, 0.05, 0.1, 0.25). Both analyses show that power increases with sample size, with minimal differences between dh values, particularly at larger sample sizes. Bar charts display average power for top-ranked networks/clusters at N = 20, 80, and 200.

4 Discussion

We present IC-TFCE, an incremental cluster algorithm that produces results equal to those of the TFCE, but with improved computational efficiency. The algorithm builds clusters incrementally across threshold levels by reusing clusters from previous steps instead of recomputing them at each threshold. For FC, the traditional TFCE has computational complexity $O(N^2 n_{th})$, where N is the number of ROIs or voxels, and $n_t h$ is the number of threshold bins (the maximum t statistic divided by the TFCE integration step size t_{max}/dh). IC-TFCE reduces this to $O(N^2 + n_{th}N)$, making the runtime less dependent on discretization precision. On FC data, this translates to measured speedups of 3–93× across multiple dh values, with greater advantages at finer precision. The method extends to voxel activation data through a graph-based transformation that unifies both data types.

Due to the increased computational efficiency of the IC-TFCE for multiple values of dh, we can now perform empirical power analysis on different values of dh, which were computationally impractical with the traditional TFCE. In this power analysis, we compared the overall average power and the average power of the top-most powered network and found that increasing the precision (decreasing dh) does not provide meaningful increases in average power and per network average power. Our power analysis found no meaningful benefit to increasing precision beyond $dh = 0.1$, consistent with current recommendations [SN08]. For large-scale analyses, the value $dh = 0.25$ could be justified. Furthermore, even in analysis using $dh = 0.25$, the speed up provided by the IC-TFCE (3×) is still

substantial compared to the traditional implementation.

While our power analysis supports $dh = 0.25$, we focus on recommendation to large-scale applications. First, because this evidence was generated on specific HCP task contrasts and may not generalise to all study types. Second, unless computational constraints make finer precision impossible or impractical, we recommend maintaining $dh = 0.1$ or lower, as there is no reason to sacrifice precision unnecessarily. For novel applications where the appropriate dh is uncertain, practitioners who wish to justify increased values of dh could compare results with the Exact TFCE for validation on a small subset of data to assess if the added error provides a meaningful difference.

For group-level analysis, since higher n_{sub} increases the observed t_{max}, which in turn increases n_{th} with fixed dh, one could expect IC-TFCE's advantage over traditional TFCE to grow with sample size. Table 1 shows that in practice this effect is minimal: execution times remain nearly flat across $n_{sub} = 20, 80, 200$ for both algorithms, and the speedup ratios are correspondingly stable. This can be explained by the permutations, as the computation time of the TFCE is usually governed by the 1000 permutations, as opposed to the single data computation. The permutation null distribution, rather than the observed statistics, has a lower effective t_{max} than the significant data, and the increase of t_{max} for the permutation data is not as great as the increase of the data itself, as evidenced by the increased rejections with increased sample sizes. The label shuffling from the permutation distribution redistributes the signal across subjects, preventing the permutation t_{max} from growing substantially with the sample size. Practitioners can therefore expect consistent speedups from IC-TFCE regardless of cohort size.

We can estimate the practical impact of IC-TFCE with two examples. For an individual analysis with $N = 1000$ ROIs and 10,000 permutations, traditional TFCE requires $10000 \times 68.3\,\text{ms} \approx 11\,\text{min}$ per run and the IC-TFCE reduces this to $10000 \times 3.6\,\text{ms} \approx 36\,\text{s}$. In practice, analyses must be rerun multiple times to verify preprocessing steps, check parameter choices, and correct errors, turning what would be a half-day of iteration into minutes. On larger systematic sensitivity analyses, this implementation is vital. Baggio et al. [BAS+18b] tested 260 combinations of E and H parameters for FC-TFCE; completing such a search at $N = 1000$ ROIs with 10,000 permutations would require $260 \times 11\,\text{min} \approx 49\,\text{hours}$ under traditional TFCE, but only $260 \times 36\,\text{s} \approx 2.6\,\text{hours}$ under IC-TFCE.

Although IC-TFCE improves sequential code execution, traditional TFCE offers potential parallelization advantages in specific scenarios. Traditional TFCE can parallelize cluster computation across thresholds as each is computed independently. However, as TFCE is used as a permutation test requiring thousands of executions, parallelizing across permutations is the current standard. The IC-TFCE supports the permutation-level parallelization strategy. Furthermore, parallelizing both within-execution cluster computation and across permutations would require computational resources beyond what is typically available. The IC-TFCE's faster single-execution time still provides practical advantages even when parallelization is available.

The IC-TFCE addresses a scaling challenge and a computational resource challenge. It is already outperforming commonly used TFCE implementations and is necessary for conducting large-scale studies. By reducing computational complexity from $\mathcal{O}(N^2 n_{th})$ to $\mathcal{O}(N^2 + n_{th}N)$ for FC analysis, IC-TFCE delivers better computational efficiency while maintaining numerical equivalence to traditional implementations. Our power analysis validates that the standard precision used for integration ($dh = 0.1$) provides adequate statistical power. As neuroimaging continues toward larger sample sizes and better signal quality, IC-TFCE's computational advantages are predicted by statistical theory to increase, making it the appropriate implementation for TFCE-based inference in both functional connectivity and voxel activation analyses. An implementation of the algorithm is available in the PRISME toolbox, and the code can be adapted to existing toolboxes.

References

[APE+14] Alexandre Abraham, Fabian Pedregosa, Michael Eickenberg, Philippe Gervais, Andreas Mueller, Jean Kossaifi, Alexandre Gramfort, Bertrand Thirion, and Gaël Varoquaux. Machine learning for neuroimaging with scikit-learn. *Frontiers in neuroinformatics*, 8:14, 2014.

[BAS+18a] Hugo C Baggio, Alexandra Abos, Barbara Segura, Anna Campabadal, Anna Garcia-Diaz, Carme Uribe, Yaroslau Compta, Maria Jose Marti, Francesc Valldeoriola, and Carme

Junque. Statistical inference in brain graphs using threshold-free network-based statistics. *Human Brain Mapping*, 39(6):2289–2302, 2018.

[BAS+18b] Hugo C Baggio, Alexandra Abos, Barbara Segura, Anna Campabadal, Anna Garcia-Diaz, Carme Uribe, Yaroslau Compta, Maria Jose Marti, Francesc Valldeoriola, and Carme Junque. Statistical inference in brain graphs using threshold-free network-based statistics. *Human Brain Mapping*, 39(6):2289–2302, 2018.

[Bea10] Erik B Beall. Adaptive cyclic physiologic noise modeling and correction in functional mri. *Journal of neuroscience methods*, 187(2):216–228, 2010.

[BFvG+09] Marta Bianciardi, Masaki Fukunaga, Peter van Gelderen, Silvina G Horovitz, Jacco A de Zwart, Karin Shmueli, and Jeff H Duyn. Sources of functional magnetic resonance imaging signal fluctuations in the human brain at rest: a 7 t study. *Magnetic resonance imaging*, 27(8):1019–1029, 2009.

[BIM+13] Katherine S Button, John PA Ioannidis, Claire Mokrysz, Brian A Nosek, Jonathan Flint, Emma SJ Robinson, and Marcus R Munafò. Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376, 2013.

[CF07] Sylvain Conchon and Jean-Christophe Filliâtre. A persistent union-find data structure. In *ACM SIGPLAN Workshop on ML*, Freiburg, Germany, October 2007.

[CFS+26] Fabricio Cravo, Alex Fischbach, Hallee Shearer, Matt Rosenblatt, Dustin Scheinost, and Stephanie Noble. Prisme: A matlab toolbox for large data-driven multimodal power benchmarking. *bioRxiv*, pages 2026–01, 2026.

[DG02] John E Desmond and Gary H Glover. Estimating sample size in functional mri (fmri) neuroimaging studies: statistical power analyses. *Journal of neuroscience methods*, 118(2):115–128, 2002.

[EYG18] Simon B Eickhoff, BT Thomas Yeo, and Sarah Genon. Imaging-based parcellations of the human brain. *Nature Reviews Neuroscience*, 19(11):672–686, 2018.

[FCH+22] Lennart Frahm, Edna C Cieslik, Felix Hoffstaedter, Theodore D Satterthwaite, Peter T Fox, Robert Langner, and Simon B Eickhoff. Evaluation of thresholding methods for activation likelihood estimation meta-analysis via large-scale simulations. *Human brain mapping*, 43(13):3987–3997, 2022.

[Fri00] Karl J Friston. Experimental design and. *Brain Mapping: The Disorders: The Disorders*, page 33, 2000.

[FWF+94] Karl J Friston, Keith J Worsley, Richard SJ Frackowiak, John C Mazziotta, and Alan C Evans. Assessing the significance of focal activations using their spatial extent. *Human brain mapping*, 1(3):210–220, 1994.

[GLL+13] Alexandre Gramfort, Martin Luessi, Eric Larson, Denis A Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkkonen, et al. Meg and eeg data analysis with mne-python. *Frontiers in neuroscience*, 7:267, 2013.

[HSY+06] Ruth Heller, Damian Stanley, Daniel Yekutieli, Nava Rubin, and Yoav Benjamini. Cluster-based analysis of fmri data. *NeuroImage*, 33(2):599–608, 2006.

[JBB+12] Mark Jenkinson, Christian F Beckmann, Timothy EJ Behrens, Mark W Woolrich, and Stephen M Smith. Fsl. *NeuroImage*, 62(2):782–790, 2012.

[Liu16] Thomas T Liu. Noise contributions to the fmri signal: An overview. *NeuroImage*, 143:141–151, 2016.

[LSL+18] Gabriele Lohmann, Johannes Stelzer, Eric Lacosse, Vinod J Kumar, Karsten Mueller, Esther Kuehn, Wolfgang Grodd, and Klaus Scheffler. Lisa improves statistical analysis for fmri. *Nature communications*, 9(1):4014, 2018.

[NMZS22] Stephanie Noble, Amanda F Mejia, Andrew Zalesky, and Dustin Scheinost. Leveling up: improving power in fMRI by moving beyond cluster-level inference. *bioRxiv*, 2022.

[NSC20] Stephanie Noble, Dustin Scheinost, and R Todd Constable. Cluster failure or power failure? Evaluating sensitivity in cluster-level inference. *NeuroImage*, 209:116468, 2020.

[NST+17] Stephanie Noble, Marisa N. Spann, Fuyuze Tokoglu, Xilin Shen, R. Todd Constable, and Dustin Scheinost. Influences on the test–retest reliability of functional connectivity MRI and its relationship with behavioral utility. *Cerebral Cortex*, 27(11):5415–5429, 2017.

[OCH16] Nikolaas N. Oosterhof, Andrew C. Connolly, and James V. Haxby. Cosmomvpa: multi-modal multivariate pattern analysis of neuroimaging data in matlab / gnu octave. *Frontiers in Neuroinformatics*, 10:27, 2016.

[OFMS11] Robert Oostenveld, Pascal Fries, Eric Maris, and Jan-Mathijs Schoffelen. Fieldtrip: open source software for advanced analysis of meg, eeg, and invasive electrophysiological data. *Computational intelligence and neuroscience*, 2011(1):156869, 2011.

[PBD+17] Russell A Poldrack, Chris I Baker, Joke Durnez, Krzysztof J Gorgolewski, Paul M Matthews, Marcus R Munafò, Thomas E Nichols, Jean-Baptiste Poline, Edward Vul, and Tal Yarkoni. Scanning the horizon: towards transparent and reproducible neuroimaging research. *Nature reviews neuroscience*, 18(2):115–126, 2017.

[PLNR15] Cyril R Pernet, Marianne Latinus, Thomas E Nichols, and Guillaume Alexis Rousselet. Cluster-based computational methods for mass univariate analyses of event-related brain potentials/fields: A simulation study. *Journal of neuroscience methods*, 250:85–93, 2015.

[PWEF97] Jean-Baptiste Poline, Keith J Worsley, Alan C Evans, and Karl J Friston. Combining spatial extent and peak intensity to test for activations in functional imaging. *Neuroimage*, 5(2):83–96, 1997.

[SI20] Denes Szucs and John PA Ioannidis. Sample size evolution in neuroimaging research: An evaluation of highly-cited studies (1990–2012) and of latest practices (2017–2018) in high-impact journals. *NeuroImage*, 221:117164, 2020.

[SKG+18] Alexander Schaefer, Ru Kong, Evan M Gordon, Timothy O Laumann, Xi-Nian Zuo, Avram J Holmes, Simon B Eickhoff, and BT Thomas Yeo. Local-global parcellation of the human cerebral cortex from intrinsic functional connectivity mri. *Cerebral cortex*, 28(9):3095–3114, 2018.

[SKSN11] Gholamreza Salimi-Khorshidi, Stephen M Smith, and Thomas E Nichols. Adjusting the effect of nonstationarity in cluster-based and tfce inference. *Neuroimage*, 54(3):2006–2019, 2011.

[SN08] Stephen M Smith and Thomas E Nichols. Threshold-free cluster enhancement: addressing problems of smoothing, threshold dependence and localisation in cluster inference. Technical Report TR08SS1, FMRIB, Oxford University, 2008.

[SN09] Stephen M Smith and Thomas E Nichols. Threshold-free cluster enhancement: addressing problems of smoothing, threshold dependence and localisation in cluster inference. *Neuroimage*, 44(1):83–98, 2009.

[SSZ+19] Tamás Spisák, Zsófia Spisák, Matthias Zunhammer, Ulrike Bingel, Stephen Smith, Thomas Nichols, and Tamás Kincses. Probabilistic tfce: A generalized combination of cluster size and voxel intensity to increase statistical power. *Neuroimage*, 185:12–26, 2019.

[Tar75] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[TvL84] Robert E. Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, 1984.

[VESB+13] David C Van Essen, Stephen M Smith, Deanna M Barch, Timothy EJ Behrens, Essa Yacoub, Kamil Ugurbil, and WU-Minn HCP Consortium. The wu-minn human connectome project: an overview. *Neuroimage*, 80:62–79, 2013.

[VMD+21] Luca Vizioli, Steen Moeller, Logan Dowdle, Mehmet Akçakaya, Federico De Martino, Essa Yacoub, and Kamil Uğurbil. Lowering the thermal noise barrier in functional brain mapping with magnetic resonance imaging. *Nature communications*, 12(1):5181, 2021.

[WGNC12] Susan Whitfield-Gabrieli and Alfonso Nieto-Castanon. Conn: a functional connectivity toolbox for correlated and anticorrelated brain networks. *Brain connectivity*, 2(3):125–141, 2012.

[WKW14] Choong-Wan Woo, Anjali Krishnan, and Tor D Wager. Cluster-extent based thresholding in fmri analyses: pitfalls and recommendations. *Neuroimage*, 91:412–419, 2014.

[WLK07] Tor D Wager, Martin Lindquist, and Lauren Kaplan. Meta-analysis of functional neuroimaging data: current and future directions. *Social cognitive and affective neuroscience*, 2(2):150–158, 2007.

[WRW+14] Anderson M. Winkler, Gerard R. Ridgway, Matthew A. Webster, Stephen M. Smith, and Thomas E. Nichols. Permutation inference for the general linear model. *NeuroImage*, 92:381–397, 2014.

[Yeu18] Andy WK Yeung. An updated survey on statistical thresholding and sample size of fmri studies. *Frontiers in human neuroscience*, 12:16, 2018.

[ZFB10] Andrew Zalesky, Alex Fornito, and Edward T Bullmore. Network-based statistic: identifying differences in brain networks. *NeuroImage*, 53(4):1197–1207, 2010.

[ZFB12] Andrew Zalesky, Alex Fornito, and Ed Bullmore. On the use of correlation as a measure of network connectivity. *NeuroImage*, 60(4):2096–2106, 2012.

5 TFCE and IC-TFCE Equality

Let $G = (N, E, W)$ be an undirected weighted graph generated from the calculated t-statistics from a GLM fit, with N being the set of nodes, E being the set of edges, and W a function from $E \to \mathbb{R}$ that returns a t-statistic associated with an edge $e \in E$.

We assume the existence of at least one positively weighted edge in G:

$$\exists\, e \in E \quad \text{such that} \quad W(e) > 0.$$

and that the number of edges is finite:

$$|E| \leq B_e \tag{11}$$

with $B_e \in \mathbb{N}_{>0}$ and $|E|$ the number of edges.

Furthermore, we assume the function W is bounded, so it maps all finite number of edges to finite values. Therefore, there exists a $B_w \in \mathbb{R}$ such that for all $e \in E$:

$$W(e) \leq B_w \tag{12}$$

As a final assumption, we assume the graph is undirected: if $(n_i, n_j) \in E$, then $(n_j, n_i) \in E$ and $W(n_i, n_j) = W(n_j, n_i)$. Consequently, paths in G are reversible. If there exists a path from node n_a to node n_b implies the reverse path from n_b to n_a is also in the graph. The assumption holds for the symmetric functional connectivity graphs. For simplification, this assumption will remain implicit for the extent of this proof and paths will be considered symmetric.

Let $\delta t > 0, \delta t \in \mathbb{R}$ be the integration step size of the TFCE algorithm.

The TFCE algorithm computes the following: the subgraphs given a cut off threshold and the clusters from that subgraph. Finally, with the cluster sizes, the TFCE integral is calculated. We start by defining the subgraphs calculated by the TFCE and then it's clusters.

TFCE subgraphs: Let $k \in \mathbb{N}$. The sequence of subgraphs $(G_k)_{k\in\mathbb{N}}$ with $G_k = (N, E_k, W)$ where E_k is a subset of edges such that

$$E_k = \{e \in E \mid W(e) \geq k \cdot \delta t\} \tag{13}$$

is the sequence of subgraphs calculated by the TFCE algorithm.

TFCE clusters: The clusters for the TFCE are the maximum connected induced subgraphs (MCISs) of the subgraph $G_k = (N, E_k, W)$. We now define a maximum connected induced subgraph.

If $G'_k = (N'_k, E'_k, W)$ is an MCIS from $G_k = (N, E_k, W)$, it is:

- For any two nodes N'_k, there exists a path between them using only edges in E'_k,
- No additional node from $N \setminus N'_k$ can be added to N'_k without violating this property (i.e. the number of nodes is maximal).
- E'_k contains all edges from E_k that connect pairs of nodes in N'_k (i.e., $E' = \{e_{i,j} \in E' \mid n_i, n_j \in N'\}$) (i.e. the number of edges is maximal).

To prove the IC-TFCE produces equal results, we need to define both the subgraphs and clusters generated by the IC-TFCE and then show that they are equal to the ones produced by the TFCE. We will start by defining the subgraphs generated by the IC-TFCE and show they are equal. Finally, we will do the same for the clusters.

IC-TFCE Subgraphs: Now, we define the subgraphs generated by the IC-TFCE implementation. We start with some set definitions.

Let $\mathcal{T} = (T_k)_{k\in\mathbb{N}}$ and $\mathcal{H} = (H_k)_{k\in\mathbb{N}}$ be sequences of sets indexed by $k \in \mathbb{N}$.

Let each $T_k \in \mathcal{T}$ be the set of edges defined as follows:

$$T_k = \{e \in E \mid k \cdot \delta t \leq W(e) < (k+1) \cdot \delta t\} \tag{14}$$

Further, let H_k be the sets defined at such:

$$H_k = \bigcup_{i \geq k, i \in \mathbb{N}} T_i \tag{15}$$

The sequence of subgraphs generated by the IC-TFCE algorithm $(I_k)_{k\in\mathbb{N}}$ are the subgraphs with $I_k = (N, H_k, W)$.

We first prove a Lemma that will be used by most induction steps in this proof. It is used to establish the base cases.

Lemma 5.1. *There exists a $K \in \mathbb{N}$ such that $E_k = H_k = \emptyset$ for $k \geq K$*

Proof. Let $w_{max} = max_{e\in E} W(e)$. The maximum is well defined because the graphs are finite and the function W is bounded. Let $K \in \mathbb{N}$ be the lowest number such that $K \cdot \delta t > w_{max}$.

From the definition of E_k, one has that

$$E_k = \emptyset \quad \text{for} \quad k \geq K \tag{16}$$

Furthermore, for all e, it is $W(e) < K\delta t$. Therefore, from the definition of T_k, one has that $T_k = \emptyset$ for $k \geq K$. It follows from the definition of H_k that $H_k = \emptyset$

It is $E_k = H_k = \emptyset$ for $k \geq K$ and the proof is concluded. □

Lemma 5.2. *For all $k \in \mathbb{N}$, we have $E_k = H_k$.*

Proof. To prove this, one can show that assuming an element belongs to E_k implies it must be in H_k, and vice-versa (i.e. mutual inclusion).

- **Let** $e \in H_k$: Since $e \in H_k$, there exists a T_i with $i \geq k$ such that $W(e) \in [i \cdot \delta t, (i+1) \cdot \delta t)$, therefore, we can conclude $W(e) \geq i \cdot \delta t$ and consequently $e \in E_k$.
- **Let** $e \in E_k$: Since $e \in E_k$, one has that $W(e) \geq k \cdot \delta t$. From Lemma 5.1, one can conclude $W(e) \in [k \cdot \delta t, K\delta t)$. Therefore, since the sets $\{[i \cdot \delta t, (i+1) \cdot \delta t)\}_{\{i\in\mathbb{N}|i\geq k, i\leq K\}}$ form a disjoint partition of $[k \cdot \delta t, K\delta t)$, there exists an index $i \in \mathbb{N}$ such that $W(e) \in [i \cdot \delta t, (i+1) \cdot \delta t)$, and $e \in T_i$ with $i \geq k$. Since $e \in T_i, i \geq k$, one has that $e \in H_k$.

It follows by subset inclusion that $H_k = E_k$. □

Corollary 5.3.

$$I_k = G_k \quad \textit{for all } k \in \mathbb{N} \tag{17}$$

Proof. This is a direct consequence of Lemma 5.2 that establishes an equality between edges. From the definition of I_k, it has the same nodes and the same weight function for edges as G_k. Therefore, the subgraphs are equal. □

Now all that is left to prove is that the IC-TFCE truly computes the correct clusters at each iteration. We start by defining the clusters outputted by the algorithm and then show they are indeed maximum connected induced subgraphs.

Definition 5.4 (IC-TFCE Cluster Construction)**.** Let $\mathcal{C}_k$ be the set of clusters produced by the IC-TFCE from the subgraph G_k, indexed by nodes $x \in N$:

$$c_k^x = (N_k^x, E_k^x, W) \in \mathcal{C}_k$$

constructed by backwards induction from K to 0 as follows:

- **Base case**: $c_K^x = (\{n_x\}, \emptyset, W)$, where n_x is a node in N indexed by $x \in \mathbb{N}$, and $K \in \mathbb{N}$ is as defined in Lemma 5.1.
- **Inductive step**: The cluster $c_{k-1}^x = (N_{k-1}^x, E_{k-1}^x, W)$ is defined as:
  - A node n is in N_{k-1}^x if there exists $m \in \mathbb{N}$ and cluster indices $y_1, \ldots, y_m \in N$ such that:
    1. $y_1 = x$
    2. $n \in N_k^{y_m}$
    3. For each $i \in \{1, \ldots, m-1\}$, there exists an edge $e \in T_{k-1}$ with one endpoint in $N_k^{y_i}$ and the other in $N_k^{y_{i+1}}$
  - An edge e is in E_{k-1}^x if both endpoints of e are in N_{k-1}^x and $e \in H_{k-1}$

Note that when clusters c^x_{k+1} and c^z_{k+1} merge at step k, we have:

$$c^x_k = c^z_k \tag{18}$$

Furthermore, note that $m = 1$ is permitted, in which case the condition is satisfied by individual nodes with a path with no necessary edges. From this, we conclude $N^x_{k+1} \subseteq N^x_k$.

The proof strategy now is to prove the clusters produced by the IC-TFCE are MCIS. As the MCIS from a subgraph G_k are unique and the TFCE computes the MCIS of subgraphs G_k, if the IC-TFCE computes the MCIS of G_k, they will be consequently equal from their uniqueness.

Lemma 5.5. *The clusters c^x_k constructed by the IC-TFCE are maximally connected induced subgraphs*

Proof. To prove the nodes and edges are maximal and the graph is connected, one can use reverse induction from K to 0. If those three requirements are satisfied, one has that the cluster is a MCIS by the definition of MCIS.

Let c^x_k be a cluster generated by the IC-TFCE algorithm.

Base case: From Lemma 5.1, one has that the set of edges is empty for the graph G_K. Therefore, as the clusters are initially defined containing the nodes alone, the clusters in $\mathcal{C}_K$ are indeed maximum connected induced subgraphs.

Induction Step:

- **Nodes are Maximal**: Assume by contradiction that the number of nodes is not maximal: there exists a node $n_1 \in N^x_{k-1}$ with an edge $e = (n_1, n_2) \in E_{k-1}$ such that $n_2 \notin N^x_{k-1}$.

  Since $e \in E_{k-1} = H_{k-1}$ (Lemma 5.2) and $E_K = \emptyset$, there exists some $K > i \geq k-1$ such that $e \in T_i$.

  At threshold i, let c^z_{i+1} be the cluster containing n_1 and let c^w_{i+1} be the cluster containing n_2. Since $e = (n_1, n_2) \in T_i$, the edge e is a bridging edge between c^z_{i+1} and c^w_{i+1} at step i. This implies $n_1 \in N^z_i$ and $n_2 \in N^z_i$ and they are both in the same cluster (indexed with z). By cluster construction, one has that $n_1, n_2 \in N^z_{k-1}$ as $N^z_i \subseteq N^z_{k-1}$. Since $n_1 \in N^x_{k-1}$, from the definition in Equation (18), one has that $N^x_{k-1} = N^z_{k-1}$ and that $n_2 \in N^x_{k-1}$ a contradiction to the assumption that $n_2 \notin N^x_{k-1}$.

  It follows that the number of nodes is maximal.

- **Edges Are Maximal**: Let us assume by contradiction that the number of edges is not maximal: there exists an edge e from $n_1 \in N^x_{k-1}$ to $n_2 \in N^x_{k-1}$ such that $e \notin E^x_{k-1}$. To avoid a contradiction with the algorithms definition, one must have $e \notin H_{k-1}$. However, from Lemma 5.2, $H_{k-1} = E_{k-1}$, so if $e \notin H_{k-1}$, it is not in the edge E_{k-1} set of the subgraph G_{k-1}. Since the cluster is from the subgraph G_{k-1} at iteration $k-1$ within the set of edges E_{k-1}, this is a contradiction and no such e is outside of this scope of edges, and consequently, the number of edges is maximal.

- **Graph is connected**: Let two nodes in $n_1, n_2 \in N^x_{k-1}$. From the definition of the set of nodes for each cluster in iteration of the algorithm, for n_1 to be in N^x_{k-1}, there exists a node $n_3 \in N^x_k$ (can be n_1) that is connected to n_1 through a path of bridging edges from the previous iteration step. Again, repeating the same argument, from the definition of the algorithm, there exists a node $n_4 \in N^x_k$, such that there is a path from n_2 to n_4. Since, one has that $n_3 \in N^x_k$ and $n_4 \in N^x_k$, there exists a path from n_3 to n_4 as N^x_k as is a set of nodes from the cluster c^x_k must be a connected graph from the induction assumption.

  Therefore, the path:

  $$n_1 \to n_3 \to n_4 \to n_2 \tag{19}$$

  proves that the nodes n_1 and n_2 are connected by a path.

With this induction, one can conclude c^x_{k-1} is a MCIS and the proof is concluded.

□

From the results from Corollary 5.3, we establish that the edge allocation procedure from the IC-TFCE creates subgraphs equal to the TFCE. On Lemma 5.5, we establish that the clusters from IC-TFCE are indeed MCIS. Since its folklore that MCIS are unique for a given graph, it follows that IC-TFCE generates the same clusters as the traditional TFCE implementation.

5.1 Extension to Exact TFCE

It suffices to propose T_k such that

$$T_k = \{e \in E \,|\, w_i \leq W(e) < w_{i+1}\} \tag{20}$$

where $(w_i)_{i\in\mathbb{N}}$ is the ordered (monotonically increasing) sequence of weights for all images of $W(e)$ with $e \in E$. The conclusions and reasoning from proofs from the IC-TFCE would follow.

6 Voxel Graph Transformation

In this section, we prove that the graph-based transformation yields clusters equivalent to the traditional voxel cluster definition used in neuroimaging.

6.1 Voxel Data and Traditional Clusters

Let $\mathcal{V}$ be a finite set of voxels and let $\sim$ denote spatial adjacency (voxels neighboring each other) on $\mathcal{V}$, where we include self-adjacency (i.e., $v \sim v$ for all $v \in \mathcal{V}$). Let $T : \mathcal{V} \to \mathbb{R}$ be a function mapping each voxel to its test statistic. We refer to the triple $(\mathcal{V}, \sim, T)$ as *voxel data*.

Given a threshold $h \in \mathbb{R}$, a voxel $v_x \in \mathcal{V}$, and voxel data $\mathcal{D} = (\mathcal{V}, \sim, T)$, the thresholded cluster definition which is used by the voxel-based TFCE and cluster size inference in neuroscience:

$$\mathcal{N}_h(v_x, \mathcal{D}) = \left\{ v \in \mathcal{V} \;\middle|\; \begin{array}{l} \exists \text{ sequence } (u_0, u_1, \ldots, u_k) \text{ with } u_0 = v_x,\ u_k = v, \\ u_i \sim u_{i+1} \text{ for all } i < k, \text{ and } T(u_i) \geq h \text{ for all } i \leq k \end{array} \right\} \tag{21}$$

Intuitively this means that if you take any voxel in the cluster, you can recreate the cluster by using adjacent paths above the desired threshold.

6.2 Graph Transformation

We define the equivalent graph transformation Γ that maps voxel data to a weighted graph. Given voxel data $\mathcal{D} = (\mathcal{V}, \sim, T)$, define $\Gamma : (\mathcal{V}, \sim, T) \to G = (N, E, W)$ where:

Nodes N: Each voxel becomes a node in the graph:

$$N = \mathcal{V} \tag{22}$$

Edges E: Two nodes are connected by an edge if and only if their corresponding voxels are spatially adjacent. Recall that we include self-adjacency ($v \sim v$ for all v), so each node also has a self-loop:

$$E = \{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid v_1 \sim v_2\} \tag{23}$$

Weights W: The weight function maps an edge to the minimum test statistic of its two endpoints:

$$W(v_1, v_2) = \min\{T(v_1), T(v_2)\} \text{ with } v_1, v_2 \in \mathcal{V} \tag{24}$$

Note that the self-adjacency condition ensures that the edges from a node to itself $(v, v) \in E$ result in $W(v, v) = T(v)$ for each voxel.

6.3 Graph-Based Clusters

Given a threshold $h \in \mathbb{R}$, define the following subset of edges from E:

$$E_h = \{e \in E \mid W(e) \geq h\} \tag{25}$$

For a node $v_x \in N$ and voxel data $\mathcal{D}$, define the *graph cluster* $\mathcal{S}_h(v_x, \Gamma(\mathcal{D}))$ as the set of nodes reachable from v_x via edges in E_h:

$$\mathcal{S}_h(v_x, \Gamma(\mathcal{D})) = \left\{ v \in N \;\middle|\; \begin{array}{l} \exists \text{ sequence of edges } (e_1, \ldots, e_k) \subseteq E_h, \text{with } k \geq 1, \text{ forming} \\ \text{a path from } v_x \text{ to } v \end{array} \right\} \tag{26}$$

6.4 Equivalence of Cluster Definitions

Theorem 6.1. *Let $G = \Gamma(\mathcal{D})$ be the equivalent graph transformation applied to voxel data $\mathcal{D} = (\mathcal{V}, \sim, T)$. For any threshold $h \in \mathbb{R}$ and any voxel $v_x \in \mathcal{V}$:*

$$\mathcal{N}_h(v_x, \mathcal{D}) = \mathcal{S}_h(v_x, \Gamma(\mathcal{D})) \tag{27}$$

Proof. One can show that the sets are equal through mutual inclusion.

- **Let** $v \in \mathcal{S}_h(v_x, \Gamma(\mathcal{D}))$: From the definition of $\mathcal{S}_h(v_x, \Gamma(\mathcal{D}))$, there exists a path of edges in E_h from v_x to v, corresponding to a sequence of voxels $(u_0, u_1, \ldots, u_k)$ with $u_0 = v_x$ and $u_k = v$.

  For each edge (u_i, u_{i+1}) in this path, one has $W(u_i, u_{i+1}) \geq h$, which implies $\min\{T(u_i), T(u_{i+1})\} \geq h$. Consequently, for all $i \leq k$, it is $T(u_i) \geq h$.

  Since each $(u_i, u_{i+1}) \in E_h \subseteq E$, it follows from the definition of E that $u_i \sim u_{i+1}$ for all $i < k$. Considering the following statements: it is $u_0 = v_x$, $u_k = v$, and $T(u_i) \geq h$ for all $i \leq k$, the sequence $(u_0, \ldots, u_k)$ satisfies all conditions in the definition of $\mathcal{N}_h(v_x, \mathcal{D})$, so $v \in \mathcal{N}_h(v_x, \mathcal{D})$.

- **Let** $v \in \mathcal{N}_h(v_x, \mathcal{D})$: From the definition of $\mathcal{N}_h(v_x, \mathcal{D})$, there exists a sequence of adjacent voxels $(u_0, u_1, \ldots, u_k)$ with $u_0 = v_x$, $u_k = v$, and $T(u_i) \geq h$ for all $i \leq k$.

  Since $u_i \sim u_{i+1}$, the edge $(u_i, u_{i+1}) \in E$ for all $i \leq k$.

  Since one has both $T(u_i) \geq h$ and $T(u_{i+1}) \geq h$ for each consecutive pair, one has:

  $$W(u_i, u_{i+1}) = \min\{T(u_i), T(u_{i+1})\} \geq h \tag{28}$$

  From $(u_i, u_{i+1}) \in E$ and Equation (28), it follows from $(u_i, u_{i+1}) \in E_h$ for all $i \leq k$.

  It follows there exists a path in E_h from v_x to v, so $v \in \mathcal{S}_h(v_x, \Gamma(\mathcal{D}))$.

One can conclude that the sets are indeed equal.

□