https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

TriCCo v1.0.0 - a cubulation-based method for computing connected components on triangular grids

Aiko Voigt[1], Petra Schwer[2], Noam von Rotberg[2], and Nicole Knopf[3]

[1]Department of Meteorology and Geophysics, University of Vienna, Austria
[2]Institute for Algebra and Geometry, Department of Mathematics, Otto-von-Guericke University, Magdeburg, Germany
[3]Institute of Meteorology and Climate Research - Department Troposphere Research Karlsruhe Institute of Technology, Germany

Correspondence: Aiko Voigt (aiko.voigt@univie.ac.at)

Abstract. We present a new method to identify connected components on a triangular grid. Triangular grids are, for example, used in atmosphere and climate models to discretize the horizontal dimension. Because they are unstructured, neighbor relations are not self-evident and identifying connected components is challenging. Our method addresses this challenge by involving the mathematical tool of cubulation. We show that cubulation allows one to map the 2-d cells of the triangular grid onto the vertices of the 3-d cells of a cubic grid. The latter is structured and so connected components can be readily identified on the cubic grid by previously developed software packages. An advantage is that the cubulation, i.e., the mapping between the triangular and cubic grids, needs to be computed only once, which should be benifical for analysing many data fields for the same grid. We further implement our method in a python package that we name TriCCo and that is made available via pypi and gitlab, as well as archived on zenodo. We document the package, demonstrate its application using cloud data from the ICON atmosphere model, and characterize its computational performance. This shows that TriCCo is ready for triangular grids with 100,000 cells, but that its speed and memory requirements need to be improved to analyse larger grids.

1 Introduction

Climate and atmospheric modeling is experiencing a leap in its ability to represent Earth digitally (Satoh et al., 2019; Wedi et al., 2020). The leap is made possible by a drastic increase in spatial resolution and the development of global storm-resolving models that apply local differencing schemes and discretize the sphere by means of unstructured grids. An example of the latter is the triangular grid based on the icosahedron and applied in the ICON unified weather and climate model (Zängl et al., 2015; Giorgetta et al., 2018). The triangular grid is a defining difference of ICON to its predecessor models ECHAM and COSMO, which were based on latitude-longitude grids.

While having many numerical advantages, the change from a structured latitude-longitude to an unstructured triangular grid challenges established workflows and analysis methods. For some types of analysis one might accept to interpolate the model output to latitude-longitude coordinates. For others, however, an interpolation might be

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021
© Author(s) 2021. CC BY 4.0 License.

problematic as it artificially smoothes the boundary of objects such as clouds, thereby potentially introducing an ambiguity in object-based analyses.

In this paper we present a new method that lifts a triangular grid to a cubic grid by means of cubulation. The method takes data that is stored as an unordered 1-dimensional array and indexed in terms of triangles and makes it accessible as a three-dimensional matrix with self-evident neighbor relations. This means that analysis tools developed for structured latitude-longitude grids can be employed on the three-dimensional matrix representation of the data. This is the key advantage of the method and our motivation to develop it.

Connected-component labeling is an established tool in atmospheric sciences for object-based studies of atmospheric moisture and clouds, and their topology. It has been used to characterize large-scale moisture transport in the form of atmospheric rivers (Muszynski et al., 2019) and has been widely applied to characterize clusters of convective clouds (Neggers et al., 2003; Rieck et al., 2014; Rempel et al., 2017; Licón-Saláiz et al., 2020), whose size statistics and distance to neighbours impacts cloud behavior, cloud organization and cloud radiative effects (Schäfer et al., 2016; Jakub and Mayer, 2017). With climate models moving to storm-resolving resolutions of a few kilometers in the atmosphere and finer, three-dimensional radiative effects of clouds are becoming increasingly important. As the radiative properties of clouds differ strongly from those of their surrounding air, a connected-component labeling that respects the sharp boundaries between cloudy and cloud-free air seems especially important.

The method that we introduce in this paper makes it possible to identify connected components for data on a triangular grid by using an existing connected-component labeling algorithm for cubical data. The method combines our expertise in climate science and high-resolution modeling on the one hand, and pure mathematics and geometric group theory on the other hand. By doing so we aim to help address the need for new strategies in analyzing the big-data output from next-generation climate models. Fig. 1 illustrates this need by comparing the simulation of clouds over the North Atlantic for two model resolutions (Senf et al., 2020). The left panel shows the cloud field for the model run at a 80 km resolution, for which the triangular grid structure can be spotted by eye. The resolution is typical for contemporary global climate models used to anticipate how climate will evolve over the coming century (Eyring et al., 2016). The right panel shows the same cloud field simulated at a much finer resolution of 2.5 km and illustrates the rich patterns of clouds at the mesoscale that are becoming accessible in the new generation of storm-resolving models.

The purpose of the paper is threefold. In Sect. 2 we first introduce the basic idea of our method and to rigorously define its mathematical foundation. Secondly we practically implemented the method and developed an open-source python package named TriCCo. The implementation is described in Sect. 3, and an example for an application is presented. The third purpose is to characterize the strengths and weaknesses of TriCCo's current implementation in Sect. 4, so as to both demonstrate its feasibility as well as to point out how its computational performance can be improved. The reader mostly interested in the use of TriCCo might focus on Subsect. 2.1 and Sects. 3 and 4.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021
© Author(s) 2021. CC BY 4.0 License.

2 Methodology: Component labeling via cubulation

2.1 General idea

Before detailing the mathematical aspects of our method, we describe its general idea in this subsection. The method is based on the realization that a triangular grid can be embedded into a 3-dimensional cubical grid. On the triangular grid, the cell centers are indexed as a 1-dimensional array that on its own does not describe the neighbor relationships. On the cubical grid, in contrast, the triangles are indexed in terms of triples (x, y, z), and the neighbor relations become self evident.

The simultaneous, adjacency preserving translation of cell indices on the triangular grid into (x, y, z)-positions on the cubical grid is called cubulation. This method makes use of the three sets of parallel classes of lines (also called hyperplanes) in the grid that are formed by the edges of the triangles. Each set of parallel lines is consecutively numbered and the position of a triangle can be described by the three indices of the lines that contain the triangle's edges. This process leads to the cubical coordinates.

The concept is illustrated in Fig. 2. The edges of the triangle cells are shown in black, and the base triangle is highlighted in cyan. Each edge of the base triangle is contained in a unique line in the plane. The three lines obtained in this way are highlighted in red, green and blue. Each other line is parallel to one of the three lines. We enumerate these classes from one (green) to three (blue). Some examples of parallel lines are highlighted in a same color. We index each line by a number as shown in Fig. 2 by starting with the index 0 for the line containing the edge of the cyan triangle. The position of any triangle in the grid is described by means of the line indices of the hyperplanes. For example, the position of the highlighted triangle is $(0, 0, 0)$. The three neighbors that share an edge with the highlighted triangle have indices $(1, 0, 0)$ (lower-left triangle), $(0, 1, 0)$ (lower-right triangle) and $(0, 0, 1)$ (upper triangle). For a precise description, see Definition 2.6.

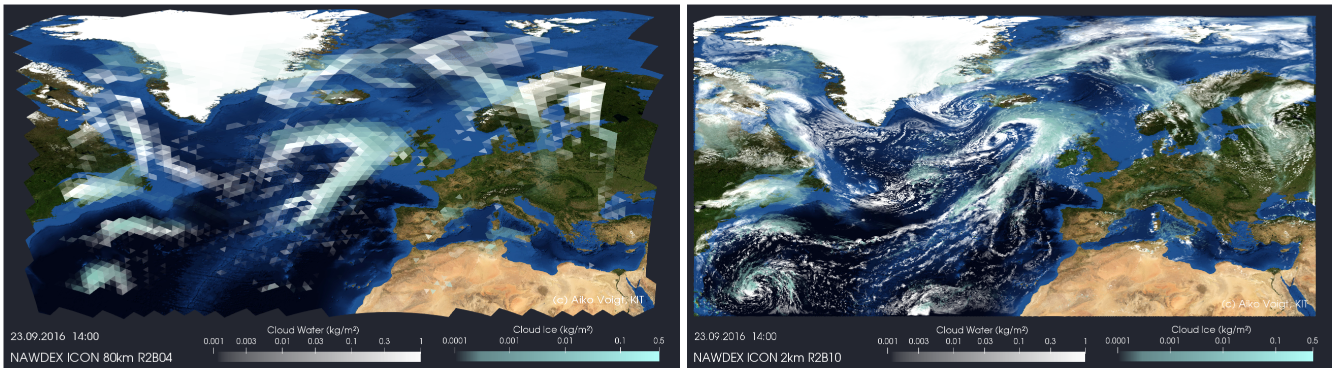

Figure 1. Illustration of clouds simulated by the ICON atmosphere model in a low-resolution version with 80 km horizontal grid spacing (left) and a high-resolution version with 2 km horizontal grid spacing (right). The triangular grid structure is visible in the left panel.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

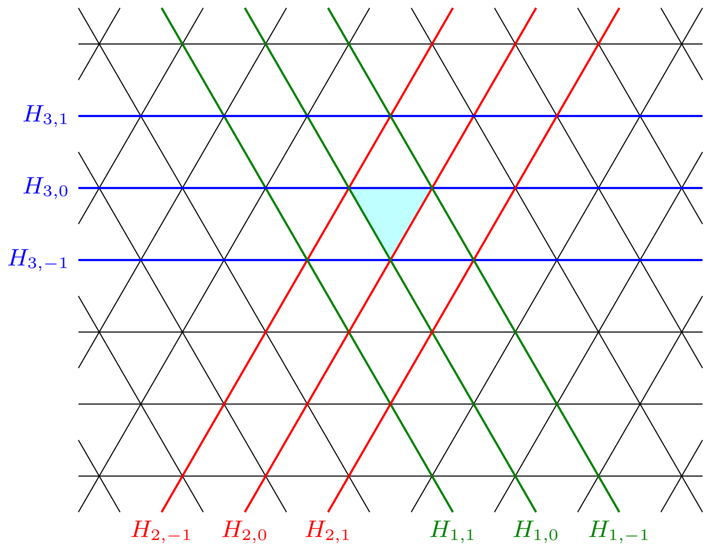

Figure 2. Illustration of the general idea of the cubulation method. The triangular grid is shown in terms of the triangle edges in black. Some parallel hyperplanes are shown in same colors.

As a result, the neighbors of a triangle are self-evident when the cubical positions are used, and the connected component labeling can be performed on a structured cubical grid.

2.2 Mathematics of the cubulation method

In this subsection we describe the algorithm that transforms (a connected subset of) the regular triangle tiling of the Euclidean plane into a subset of the standard subdivision of $\mathbb{R}^3$ into unit cubes. The method is a concrete example and implementation of Sageev's cubulation method introduced in Sageev (1995) for the Coxeter group of type $\tilde{A}_2$. The vertices of this cubulation will have integer-valued coordinates. We start with some characteristics concerning the structure of the triangular grid in Section 2.2.1, collect some necessary background on cube complexes in Section 2.2.2 and then carry out the construction in Section 2.2.3

2.2.1 Structure of the triangle tiling

The cubulation of the regular triangle tiling of the plane is the key tool that makes TriCCo work. The regular triangle tiling of the plane will be called Σ in the following. The space Σ carries the structure of a (metrized) simplicial complex whose maximal simplices are all 2-dimensional and in which all edges have the same length. A picture of this complex is provided in Figure 3 below. The 2-dimensional simplices are the triangles in this figure; edges are shown in black.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

The cubulation that we construct restricts to any (connected) subset of the triangles of the plane and hence automatically yields a method to cubulate local lattices of any granularity.

There are three parallel classes of lines in Σ. We have illustrated these classes in Figure 2 by highlighting some lines of a same class in a same color. These three classes will correspond to the three pairwise perpendicular coordinate axes of $\mathbb{R}^3$ in which the cubulation lives.

We will now define a graph associated with Σ.

Definition 2.1. The *dual graph* Γ_Σ of the triangle tiling Σ of the plane is defined as follows: The set of vertices V in Γ_Σ is the set of triangles in Σ. There exists an edge (u,v) between $u,v \in V$ if and only if the triangles u and v share a codimension one face, i.e., they have an edge in common.

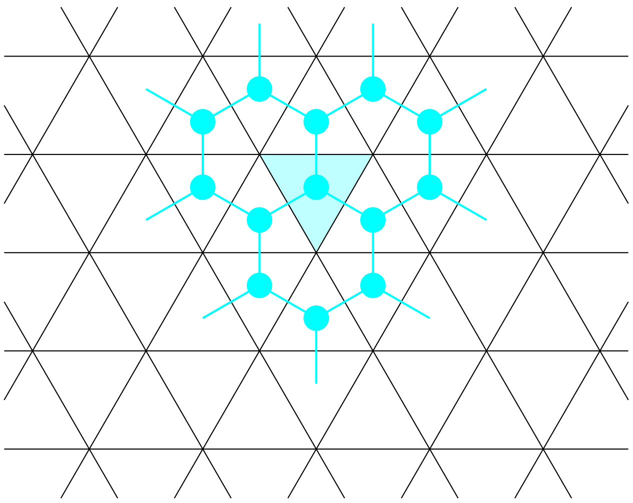

Figure 3. The figure shows a piece of the equilateral triangle tiling Σ of the plane. The turquoise vertices and edges represent the dual graph of the tiling as defined in Definition 2.1.

The dual graph can be pictured inside the tiled plane as follows. Draw a point in the center of each of the triangles. Each of these points represents a vertex of the graph Γ_Σ. Two points are connected by an edge whenever the corresponding triangles have a side in common. These edges may be drawn perpendicular to the common face. Figure 3 illustrates this correspondence.

Each vertex of the dual graph Γ_Σ by construction corresponds to a unique triangle in Σ. Every hexagon in Γ_Σ corresponds to a collection of six triangles in Σ sharing a common vertex.

2.2.2 Cubical complexes

Cubical complexes are spaces obtained by gluing unit cubes of various dimensions along isometric faces, i.e. faces of the same dimension. A unit cube is a cube in some Euclidean space of dimension k all of whose edges are of length one. More formally and in short, a *cubical complex* K is an M_k-polyhedral complex such that all the shapes are unit cubes, i.e. of the form $[0,1]^k$ for some $k \in \mathbb{N}$. For details on M_k complexes see Bridson and Haefliger (1999) or Schwer (2019).

We will provide an ad-hoc definition of a cubical complex below in order to allow for a treatment of the subject without the need of introducing general M_k-polyhedral complexes.

Definition 2.2 (Cubes)**.** An *n-cube* is a set C of the form $C = [0,1]^n \subset \mathbb{R}^n$. A *codimension one face* of C is given by $F_{i,\epsilon} := \{x \in C | x_i = \epsilon\}$ for $\epsilon \in \{0,1\}, i = 1,\dots,n$. All other (proper) faces of C are non-empty intersections of codimension 1 faces. We say that $x \in C$ is an *inner point* of C if x is not contained in any (proper) face of C.

These cubes will now be glued together to form larger complexes. For technical reasons we will assume that the intersection of two cells in a cubical complex is either empty or a face of both. Some, but not all, of the gluings that do not satisfy this assumption can be resolved by further subdividing the complex into smaller cubes.

Definition 2.3 (Cubical complexes)**.** Let C and C' be two cubes with faces $F \subseteq C$ and $F' \subseteq C'$ [1]. A *gluing* of C and C' is an isometry $\phi : F \to F'$, which provides an identification of two of the sides of the cubes.

Suppose $\mathcal{C}$ is a set of cubes and $\mathcal{S}$ a family of glueings of elements of $\mathcal{C}$, that is for all $C \in \mathcal{C}$ there is $n_C \in \mathbb{N}$ such that $C \cong [0,1]^{n_C}$ and every $\phi \in \mathcal{S}$ is an isometry $\phi : F \to F'$ where F, F' are faces of cubes $C, C' \in \mathcal{C}$. Assume further that $(\mathcal{C}, \mathcal{S})$ satisfies the following two conditions:

1. No cube is glued to itself.
2. For all $C, C' \in \mathcal{C}$ there is at most one gluing of C and C'.

Then $(\mathcal{C}, \mathcal{S})$ defines a *cubical complex* (X, d) by putting $X := (\bigsqcup_{C \in \mathcal{C}} C) / \sim$ where $\sim$ is the equivalence relation generated by putting $x \sim \phi(x)$ for $\phi \in \mathcal{S}$ and $x \in \mathrm{dom}(\phi)$. The metric d on X is the length metric induced by the restricted Euclidean metric on each cube in $\mathcal{C}$.

An example of a cubical complex is shown in Fig. 4.

One property of a cubical complex X is that the restriction of the quotient map $\mathrm{p} : \bigsqcup_{C \in \mathcal{C}} C \to X$ to one cube $C \in \mathcal{C}$ is injective. And that the intersection of two cubes in X is either empty or a face of both (here a face might be the whole cube). Hence we may identify a cube $C \in \mathcal{C}$ with its image in C and write $C \in X$.

One of the key features of a cube complex are *hyperplanes*. Hyperplanes are cubical complexes themselves which we may associate to the midcubes parallel to codimension-one faces of certain cubes. These hyperplanes then cut through the middle of adjacent cubes. Examples are shown in Figure 5.

In a cube complex that satisfies the additional curvature property of being CAT(0) every hyperplane cuts the complex into two disjoint pieces, called halfspaces. The partially ordered set of all these halfspaces allows to recover the cubical complex itself.

[1]Note that here possibly $F = C$ or $F' = C'$

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

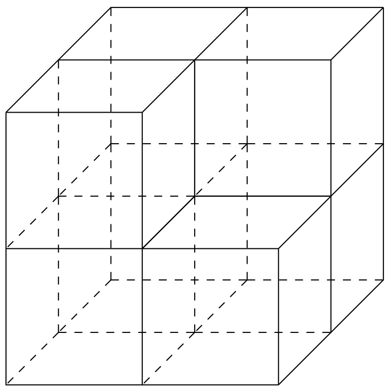

Figure 4. A cubical complex built out of seven 3-cubes.

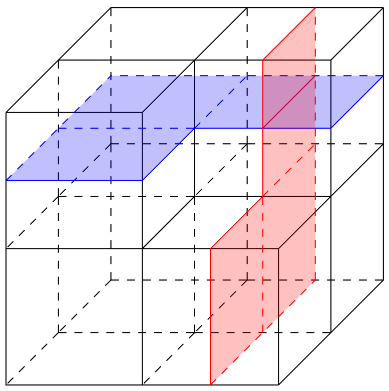

Figure 5. The blue and red 2-dimensional cubical complexes are examples of hyperplanes in the cubical complex we have already seen in Fig. 4.

In the next section we will cubulate the equilateral triangle tiling of the plane using the hyperplanes and half-spaces appearing in the tiling. More generally one can introduce an abstract notion of half-space systems and use those to cubulate more abstract spaces than the example we are considering here. See for example Schwer (2019).

Our main goal is the following.

Main Goal 2.4. *Construct from every edge-connected subcomplex A of Σ a subcomplex $X(A)$ of the standard cubulation X of $\mathbb{R}^3$. Adjacency of triangles in the plane should be equivalent to adjacency of the associated cubes in X.*

As mentioned above, Euclidean 3-space can be subdivided and equipped with the structure of a (metric) cubical complex. We call this cube complex the *standard cubulation* of $\mathbb{R}^3$ and denote it by X. Its vertices are the points in $\mathbb{R}^3$ whose coordinates are all integers with respect to the standard basis $\{(1,0,0),(0,1,0),(0,0,1)\}$. Denote this set of vertices by $X^{(0)}$. The edges of the cubes are the intervals between any pair of these integer valued vertices that differs in exactly one entry. The graph that is formed by these vertices and edges is called the *one-skeleton* of X and is denoted by $X^{(1)}$.

2.2.3 Construction of the cubulation of Σ

In fact we will not cubulate Σ but its dual graph Γ_Σ. To be precise: The goal is to define a simplicial map from Γ_Σ to the one-skeleton $X^{(1)}$ of the standard cubulation X of $\mathbb{R}^3$.

We now introduce a labeling of the lines in the tiling Σ which will allow us to define such a map.

Definition 2.5. A *consistent labeling* of the set of hyperplanes in Σ is the following procedure that assigns to every line the number of its class and an integer index as follows. Fix a base-triangle v_0 in Σ. There exist then three hyperplanes each containing one of the three sides of v_0. Call them $H_{1,0}, H_{2,0}$ and $H_{3,0}$, respectively. In addition there is a unique hyperplane parallel to $H_{i,0}$ whose intersection with v_0 is a single vertex. Call this hyperplane $H_{i,-1}$ and enumerate all other hyperplanes in the same parallel class periodically.

See Figure 2 for an illustration of the labeling we have just defined. In the next definition we obtain from the labeling defined in Definition 2.5 coordinates for the vertices in Γ_Σ. These can be used these to define a map from the vertex set V of Γ_Σ to the vertices $X^{(0)} \subset X$.

Definition 2.6 (The 3d-coordinates for triangles)**.** Recall that V is the set of vertices of the dual graph Γ_Σ. For each $v \in V$ define 3-dimensional *coordinates* (v_1, v_2, v_3) by putting $v_i := k$ if the triangle in Σ corresponding to v lies between the hyperplanes $H_{i,k}$ and $H_{i,k-1}$ in Σ.

In Figure 3 the dual graph Γ_Σ is shown in turquoise. The vertex of the dual graph inside the turquoise triangle has coordinates $(0,0,0)$. Vertices contained in a common hexagon of the dual graph will be mapped to the same 3-cube in the cubulation.

Definition 2.7 (Cubulation map)**.** The *cubulation map* $f : V \to X^{(0)}$ is defined by $v \mapsto (v_1, v_2, v_3)$ where the v_i are chosen as in Definition 2.6.

Figure 6 illustrates some of the images of vertices in Γ_Σ inside the 1-skeleton of X.

The cubulation map satisfies some properties and in particular preserves adjacency of vertices.

Proposition 2.8 (Properties of the cubulation map)**.** *Let f be the map defined in Definition 2.7. Then the following holds.*

1. *The map f preserves adjacency, that is, the coordinates of two adjacent vertices u, v in Γ_Σ differ in exactly one entry. Their images in $X^{(1)}$ under f are connected by an edge.*

2. *Every hexagon in Γ_Σ is mapped into a unique cube of X.*

3. *Triangles u, v that share a vertex in Σ are mapped to vertices that are contained in a same cube.*

Proof. To see the first item let u, v be adjacent vertices in Γ_Σ. They correspond then to two triangles that share a common side. This side is contained in a unique hyperplane $H_{i,k}$ for some parallel class i and some index k. So the

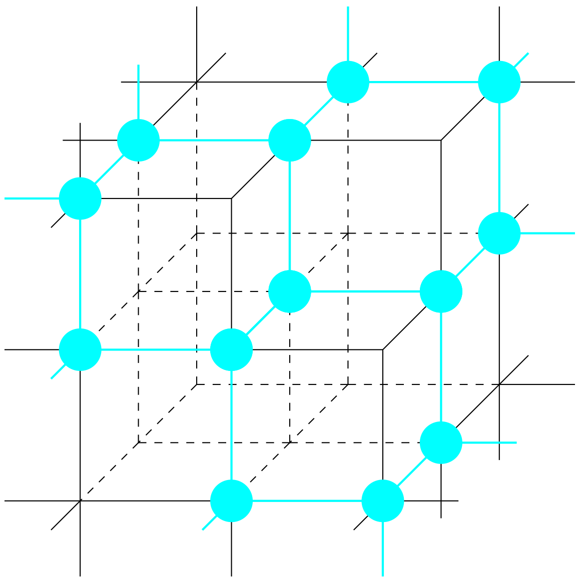

Figure 6. Illustration of the mapping from Γ_Σ to $X^{(1)}$ showing how the dual graph sits inside the 3-dimensional cubical complex.

coordinates v_i and u_i differ by one. If there was a second coordinate in which u and v would differ, there would be a second hyperplane in a different parallel class separating u and v. But this is impossible.

By checking one of the hexagons by hand one can verify that the second property is satisfied and all vertices of this hexagon are mapped to a common cube. The vertices in all other hexagons have hyperplane coordinates shifted by integer values in at least one of the three directions obtained from the parallel classes of hyperplanes. This yields the assertion.

The third item follows from the second by checking that triangles sharing a vertex are contained in a common hexagon in Γ_Σ. □

We can characterize the full image of f and describe which points in $X^{(1)}$ are part of the embedded graph Γ_Σ.

Proposition 2.9. *The image $f(V)$ of all the vertices in Γ_Σ is the collection of points in $X^{(1)}$ whose coordinates sum up to either 1 or 0. Each edge (u,v) has a vertex with coordinate sum 0 and one with coordinate sum 1.*

Proof. Let u and v be two triangles sharing an edge. For each i there is an index k_i such that u lies between H_{i,k_i} and H_{i,k_i-1} and u has coordinates (k_1,k_2,k_3). Observe that v lies between the same two parallel hyperplanes for two of the indices. Moreover, there is one index, say j, for which v is either between H_{j,k_j+1} and H_{j,k_j} or between the two hyperplanes H_{j,k_j-1} and H_{j,k_j-2}.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

Without loss of generality, let $j = 1$. Suppose first that v is between H_{1,k_1} and H_{1,k_1+1}. Then v has coordinates $(k_1 + 1, k_2, k_3)$ and the coordinate sum differs by one. In case v is between H_{1,k_1-1} and H_{1,k_1-2} it has coordinates $(k_1 - 1, k_2, k_3)$ and the coordinate sum differs by -1.

It remains to prove that coordinate sums alternate between 0 and 1. The base triangle v_0 has coordinates $(0,0,0)$ and hence coordinate sum 0. Its neighbors lie, by construction, between hyperplanes with indices 0 and 1 in one of the three directions and have hence coordinate sum 1.

One can proceed by induction on the distance to v_0 in Γ_Σ and prove that along any shortest path in Γ_Σ connecting an arbitrary vertex to v_0 the coordinates sums alternate between 0 and 1. □

2.3 Identifying connected components for 2-d data

After having computed the cubulation of the triangular grid we can use it to detect connected components.

There are two ways to define connectivity on a triangular grid: either via shared edges (edge-connectivity) or via shared vertices (vertex-connectivity). The two types of connectivity can also be defined more rigorously as follows. A subset of the triangles of the triangle tiling Σ is *edge-connected* if for any two triangles the corresponding vertices in the dual graph are connected by a path in the dual graph. We say that a set A of triangles is *vertex-connected* if for every pair of triangles u and v in A there exists a sequence of triangles $v_i \in A, i = 0, \ldots, n$ connecting u and v such that two subsequent triangles v_i and v_{i+1} share a vertex.

We need to to clarify how connectivity for cells on the triangular grid translates to connectivity for the cubical grid. One can show that edge connectivity on the triangular grid corresponds to face-connectivity (also known as 6-connectivity) on the cubic grid and that vertex-connectivity on the triangular grid translates precisely to vertex-connectivity (also known as 26-connectivity) on the cubic grid.

2.4 Identifying connected components of 3-d data

We now describe how connectivity is computed for 3-dimensional data. An example of 3-dimensional data is cloud fraction. Cloud fraction depends on geographical position, which is described by the triangular grid, and altitude. To represent the vertical dimension, the ICON model stacks layers of triangular grids. This is a standard in atmospheric modeling.

We understand connectivity in the vertical dimension to mean that neighboring layers share at least one triangle on the horizontal grid. E.g., if a cell i corresponding to latitude-longitude position (φ, λ) has the same value at model level k and model level below, $k+1$, then the grid volumes spanned by (k,i) and $(k+1,i)$ are connected. We note that we thus limit connectivity in the vertical to cell faces. This is consistent with the treatment of vertical exchange typical in atmospheric models, which occurs column-wise apart from a few exceptions such as 3-dimensional radiative transfer.

With this, connected components in 3-d can be computed from a three-step procedure. In a first step, 2-d components are identified for each model level separately following the method described in Subsect. 2.3. Each 2-d

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

component is considered a node of an undirected graph. In a second step, we identify all pairs of 2-d components that reside in neighboring model levels and share at least one triangle. The pairs are edges between the nodes formed by the 2-d components. In a third step, we apply a connected component analysis on these nodes and edges formed by the set of 2-d connected components. The overall result of this procedure is a list of 3-d connected components, where each 3-d connected component is given by a list of 2-d connected components. For the connected component analysis of step three we use the external library networkx implemented in Python, but our procedure would also work for other external network analysis libraries.

3 Software implementation and application examples

In this section we describe the software implementation and provide basic examples on how to apply the method to cloud data from the ICON atmosphere model. Our aim is to provide an orientation on the code structure and its usage. Version 1.0.0 of the implementation described and used here is available via gitlab and pypi, and long-term archived at zenodo (see code availablility; Voigt, 2021).

The implementation of TriCCo and its use consist of four steps. Each step is described in the following. Note that the terms triangle and cells are used interchangeably, with no risk of confusion as our method is designed for triangular cells.

3.1 Step 1: Preparing the horizontal grid

Regarding the horizontal triangular grid, information on the neighboring cells and the edges of each cell is needed, as well as information on the vertices that form a specific edge. The grid information is stored in an xarray dataset named `grid` using variable names that follow the convention of the ICON model grid. The variable naming is due to the fact that the ICON grid was used during the development and testing of code. The code includes routines specific to the ICON grid. It should be straightforward to adapt the routine to other grids.

Let us assume that the grid consists of n_c cells, n_v vertices and n_e edges. For a triangular grid that covers the entire sphere, n_v and n_e are given by n_c as described in Zängl et al. (2015); for a limited-area grid, the relationships hold in an approximate manner. The three variables required to describe the grid are:

- `neighbor_cell_index` defines the three neighbouring cells for each cell. The dimension is $(3, n_c)$.
- `edge_of_cell` defines the three edges for each cell. The dimension is $(3, n_c)$.
- `edge_vertices` defines the two vertices for each edge. The dimension is $(2, n_e)$.

The variables are indexed starting from 0. In ICON this requires a shift by -1 as the indexing starts with 1. The variables are accessed by three analogously-named functions that provide the variable values for a single grid cell or edge. For a limited-area grid, a missing neighbouring cell indicates the grid boundary and is assigned a value of -9999.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

3.2 Step 2: Computing the cubulation of the horizontal grid

The main function is `compute_cubulation`. This function implements the cubulation described in Section 2 and computes for each grid cell i the associated 3-d coordinate on the cubic grid (x,y,z). This information defines the cubulation and is stored in `cube_coordinates` as list of arrays of the form $(i;(x,y,z))$.

The function `compute_cubulation` starts at a user-specified `start_triangle`, and iteratively computes all cube coordinates within an expanding circle around the start cell. The iteration stops when the circle reaches a user-specified `radius`. The radius needs to be chosen according to the grid size, or alternatively can be set to a smaller value if only a specific part of the grid is of interest. If the radius is too small, the cubulation will not cover the entire grid. On the other hand, if the radius is too large, the algorithm will iterate over empty lists for the last steps. Setting `print_progress=True` outputs the progress of the iteration to screen, allowing one to monitor the number of new cells added in each round, which is helpful for choosing the radius. Also, even though iterating over empty lists comes with essentially no computational burden, the size of the cubulation increases with the radius, which in turn increases the memory demand of the connectivity analysis in Step 4. The radius thus should be as small as possible.

The following consideration is helpful when choosing the radius r. Each iteration adds $3 \cdot i$ cells, where $i \leq r$ is the number of the current iteration. The total number of visited cells is

$$n_c = 1 + \sum_{i=1}^{r} 3 \cdot i = 1 + 3 \cdot \frac{r(r+1)}{2} = 1 + 1.5r + 1.5r^2. \tag{1}$$

The sum begins with 1 due to the start cell. Thus, covering n_c cells requires a search radius of

$$r = -\frac{1}{2} + \sqrt{\frac{1}{4} + \frac{2}{3}(n_c - 1)} \tag{2}$$

The equation is exact as long as the iteration has not reached the grid borders, i.e., it works best for circle-shaped grids such as those used by Schemann and Ebell (2020). For other grids, the equation serves as a lower bound of the radius that one needs to cover n_c cells. Acknowledging that $n_c \gg 1$, the lower bound can effectively be approximated by $r \gtrapprox \sqrt{\frac{2}{3} n_c}$.

Another helpful approach to find the search radius is to start from a value somewhat larger than the lower bound and adapt the radius based on the diagnostic output of `compute_cubulation` that can be obtained by `print_progress=True`.

A few aspects of `compute_cubulation` warrant further description. The function begins by assigning the cube coordinate $(x,y,z) = (0,0,0)$ to `start_triangle`, but this is not the final coordinate of the start cell as explained further below. In each iteration, all 'new' cells that are adjacent to already visited cells are considered and their cube coordinates are calculated. Missing neighbors, as they occur for cells at the border of the grid, are identified by -9999 and ignored (cf. Section 3.1).

Moreover, the edges of a new cell need to be colored, and this needs to be done such that the edge colors are consistent with the edge colors of other cells. I.e., parallel edges need to have the same color as they belong to the same hyperplane. This is illustrated for two neighboring cells in Figure 7, where the left cell is `old` and the right cell is a so far unvisited neighbour `new`. The joint edge is already colored as `old` was visited in the preceding iteration. This leaves the task of coloring the two non-joint edges of `new`. If only one edge is uncolored, its color is given by the color which has not yet been used. If two edges of `new` need to be colored, their colors are deduced from the edge colors of `old`: a non-joint edge of `new` is colored with the same color as a non-joint edge of `old` if both edges share no vertices and hence are parallel.

The cube coordinates are computed in the following manner. As `old` and `new` are adjacent, their cube coordinates differs by ± 1 in exactly one entry of (x,y,z) by Proposition 2.9. The color of the joint edge between the two cells defines which entry needs to be changed. The decision of whether the entry differs by $+1$ or -1 follows from the constraint that the sum of cube coordinates, $x+y+z$, must be either 0 or 1 (see Proposition 2.9). That is,

$$\begin{aligned} \texttt{old}\text{ has coordinate sum } 0 \quad &\Rightarrow \quad \texttt{new}\text{ has coordinate sum } 1 \\ \texttt{old}\text{ has coordinate sum } 1 \quad &\Rightarrow \quad \texttt{new}\text{ has coordinate sum } 0 \end{aligned}$$

After all cells have been visited and the iteration is finished, the cube coordinates are shifted by `radius`/2 (rounded down to integer value) for all three dimensions to ensure that all coordinates are positive.

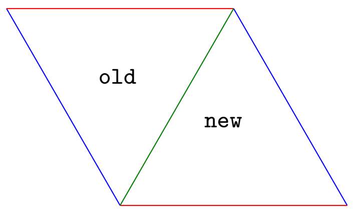

Figure 7. Illustration of how to color the edges of a newly visited cell (right) based on the edge colors of an old cell (left).

3.3 Step 3: Preparing the simulation data

The simulation data needs to be moved from the triangular grid to the cube coordinates of the cubulation. This is achieved by the function `prepare_field` for single-level data, and by the function `prepare_field_lev` for multi-level data. The functions are wrappers for model-specific functions. We include such functions for the ICON model; writing analogous functions for other models and their data format should be straightforward.

`prepare_field` and `prepare_field_lev` require as input the cubulation computed in Step 2 and a threshold value. The latter is used to convert the input data to values of either 0 and or 1, depending on whether the input data is smaller or equal-or-larger than the threshold. The functions return the thresholded input data on the triangular grid as well as on the cubical grid. In case of multi-level data, the first entry corresponds to the model level, and the following entry(ies) describe(s) the horizontal position. For the triangular grid, the horizontal position is given by the cell index i; and by three numbers (x,y,z) on the cubic grid.

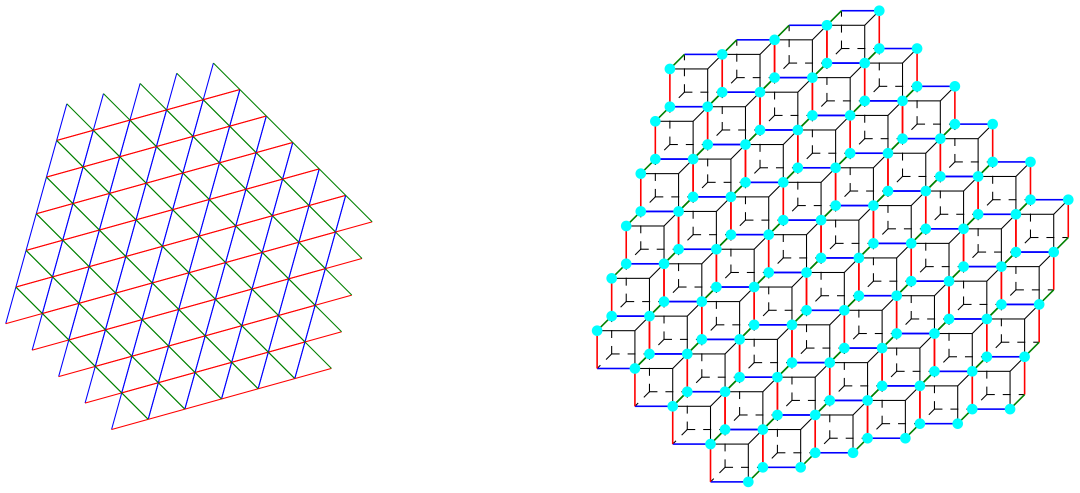

Figure 8. Illustration of `make_cubical_coordinates`. The triangular grid is shown in the left panel, the corresponding cubical grid in the right panel. In the right panel, cyan vertices correspond to triangle centers in the left panel, and analogously for colored edges. Vertices with no colors in the right panel do not correspond to triangle centers.

The first step `make_field_array` generates a 3-dimensional array `field_array` of size `radius`$+1$ in each direction. The size must be `radius`$+1$ because increasing the radius by 1 increases the range of each cube coordinate by 1, the $+1$ is necessary since we also need to store 0 in each coordinate. The field is then written into this matrix: If a triangle with cube coordinate (x,y,z) has a thresholded value of 1, then the 3-d matrix entry `field_array[x][y][z]` is set to 1 as well.

3.4 Step 4: Computing connected components

Once the cubulation is known and the simulation data is prepared, computing the connected components is done by the functions `compute_connected_components_2d` and `compute_connected_components_3d`, respectively. The functions require as input the cubulation (Step 2) and the prepared simulation data on the cube grid (Step 3). Two types of connectivity in the horizontal direction can be chosen: vertex or edge connectivity. For edge connectivity, cells in the horizontal belong to the same component if they share a triangle edge. For vertex connectivity, they also belong to the same component if they share only a triangle vertex. Vertex connectivity thus results in larger but fewer connected components. The default choice is vertex connectivity.

Examples illustrating edge- and vertex connectivity are provided in Figures 10 and 9.

The functions use the external library `cc3d` to identify connected components on a single model level. For multi-level data the external library `networkx` is used in addition to identify and merge connected components in the vertical. The final result of both functions is a list of connected components. For 2-d data a connected component is given by a list of triangular cell indices. For 3-d data, it is given by a list of tuples, with each tuple consisting of the model level and cell index.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021
© Author(s) 2021. CC BY 4.0 License.

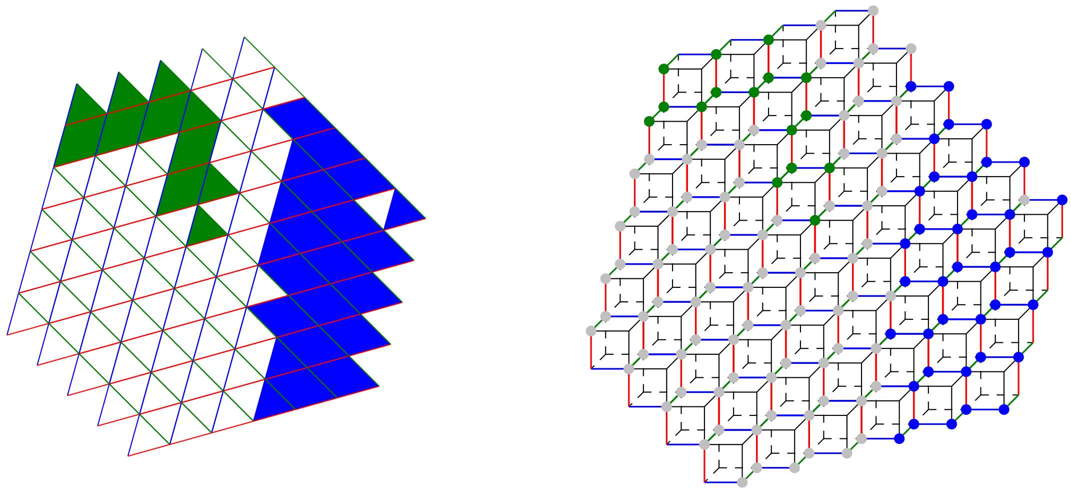

Figure 9. Illustration of the result of connected component labeling for 2-d data and vertex connectivity. In the left plot, connected components are formed by cells with the same face color. The colors of the cell edges that form the hyperplanes of the cubulation are also shown. The right plot illustrates the same data on the cubulation. Triangles on the left correspond to vertices on the right. A set of triangles on the triangular grid is vertex connected if in the set of vertices on the cubical grid any two vertices in the set can be connected by a sequence of vertices where subsequent ones are in a common 2-dimensional face (i.e. square) of a cube.

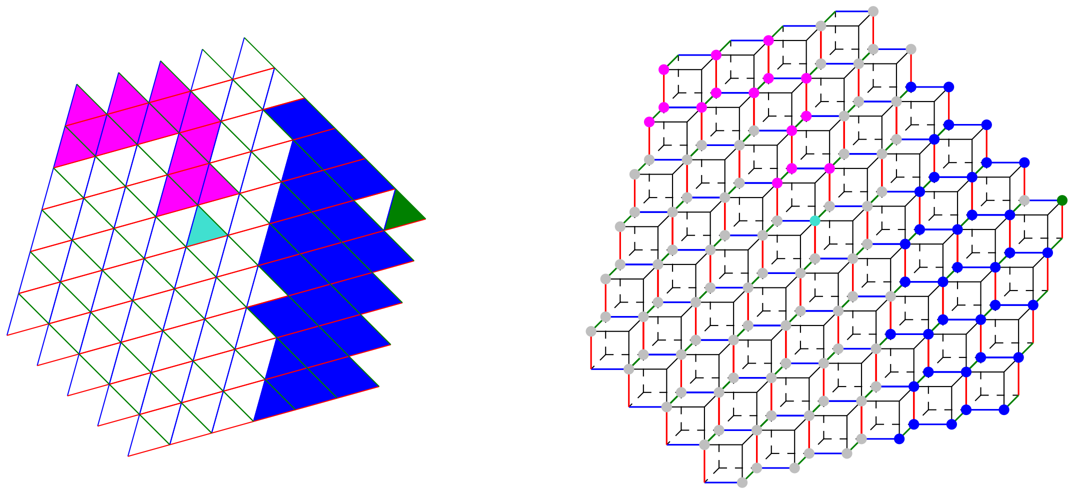

Figure 10. Same as in Figure 9 but for edge connectivity. A set of triangles on the triangular grid is edge connected if in the set of vertices on the cubical grid any two vertices in the set can be connected by a sequence of vertices such that two subsequent ones are connected by an edge of a cube.

3.5 Application examples

As an illustration of TriCCo's abilities, we analyze model output from the ICON atmosphere model in limited-area setup over the North Atlantic. The output is from a simulation that applies a triangular grid with 7,920 cells and 75 model levels and that is part of a larger set of simulations presented in Senf et al. (2020) and Stevens et al. (2020) from a scientific perspective. In our work here, the simulation output solely serves as technical input to test and illustrate the functionality and performance of the TriCCo routine. The grid has a nominal horizontal resolution of 80 km and its characteristics are listed in Tab. 1. We use total cloud cover for demonstrating the use for data on a single model level, and vertically-resolved cloud fraction on 75 levels as an example for multi-level data.

The simulation domain extends from 78 W to 40 E, and 23 N to 80 N, covering the North Atlantic, the Mediterranean, the larger part of Europe and parts of Northern Africa. To find the start cell we search for the cell with the smallest distance to the grid centre at 19 W and 51.5 N, where we measure the distance in terms of the great circle distance on the sphere using the haversine formulae. We then find the radius by using the lower bound and the diagnostic output of `compute_cubulation` as described in Sect. 3.2. The start cell and radius are given in Tab. 1.

Fig. 11 shows the result of connected component labeling for total cloud cover, which in ICON can have values between 0% (cloud-free) and 100% (completely cloud covered). Panel a shows total cloud cover from a single time step of the simulation. Centered at roughly 20 W and 50 N, there is a commashaped cloud band that is associated with a warm-conveyor belt of a North Atlantic extratropical cyclone. We threshold the data at 85%, as shown in panel b, and identify connected components for vertex (panel c) and edge connectivity (panel d). For vertex connectivity, the commashaped cloud band is connected to a cloud structure west of it. Using edge connectivity instead, the cloud band can be isolated. Overall, vertex connectivity leads to 31 connected components, compared to 74 components when the stricter criterion of edge connectivity is used.

We also present results for connected component labeling for vertically-dependent cloud fraction from the same time step, where we again apply a threshold of 85%. For vertex connectivity we identify 235 components, whereas edge connectivity results in 381 components. Fig. 12 shows the component that corresponds to the commashaped cloud band centered at 20 W and 50 N. Note that for displaying purposes, the horizontal grid is rotated and latitude decreases from left to right. The vertical structure of the cloud band is clearly visible, as is the fact that vertex connectivity associates more cells to the connected component than edge connectivity. This can be seen, for example, near the surface around 10 W and 40 N.

The python code of the examples presented here is included in TriCCo as jupyter notebooks.

4 Benchmarks and computational challenges

The ICON simulation analyzed in Sect. 3.5 is part of a larger set of simulations that includes triangular grids with much finer resolution. In this section, we use this larger set to characterize the computational aspects of TriCCo, and to identify limitations of the current implementation. The limitations result, for example, from the current

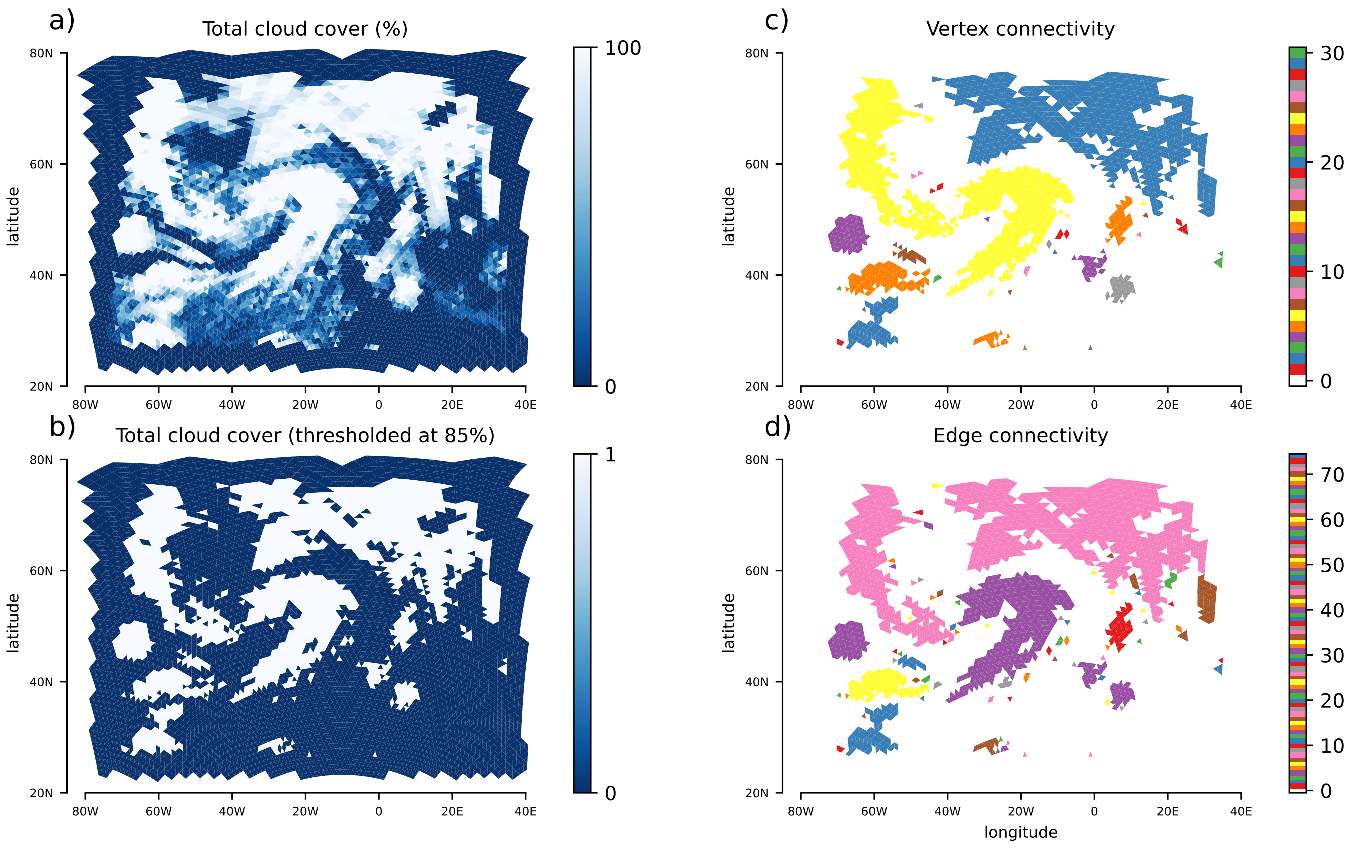

Figure 11. Application for total cloud cover from an ICON simulation with a limited-area grid over the North Atlantic. The triangular nature of the grid is visible. (a) Total cloud cover in per cent, with 0 corresponding to cloud-free and 100 to completely overcast conditions. (b) Total cloud cover thresholded at 85%, with values above 85% set to 1 and values below set to 0. (c) Connected components for vertex connectivity, with components being plotted in different colors. (d) Connected components for edge connectivity.

serial implementation that restricts use to a single core. Ultimately, they reflect that as climate scientists and pure mathematicians our expertise in software development and computational science is finite.

We use simulations with horizontal resolutions ranging from 80 km to 10 km. Their grid specifics are included in Tab. 1. Because we are interested in the computational performance, what matters here is not the grid resolution itself but the number of grid cells. The latter increases by roughly a factor of four for each grid refinement. The start cells and search radii depend on the grid and we find them following the approach outlined in Sect. 3.5. The benchmarks are run on a single core of a dedicated compute node of the Mistral supercomputer of Deutsches Klimarechenzentrum in Hamburg, Germany. A Mistral compute node has 2x12-core Intel Xeon E5-2680 v3 processors (Haswell) with a base frequency of 2.5 GHz and 64 GB main memory.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

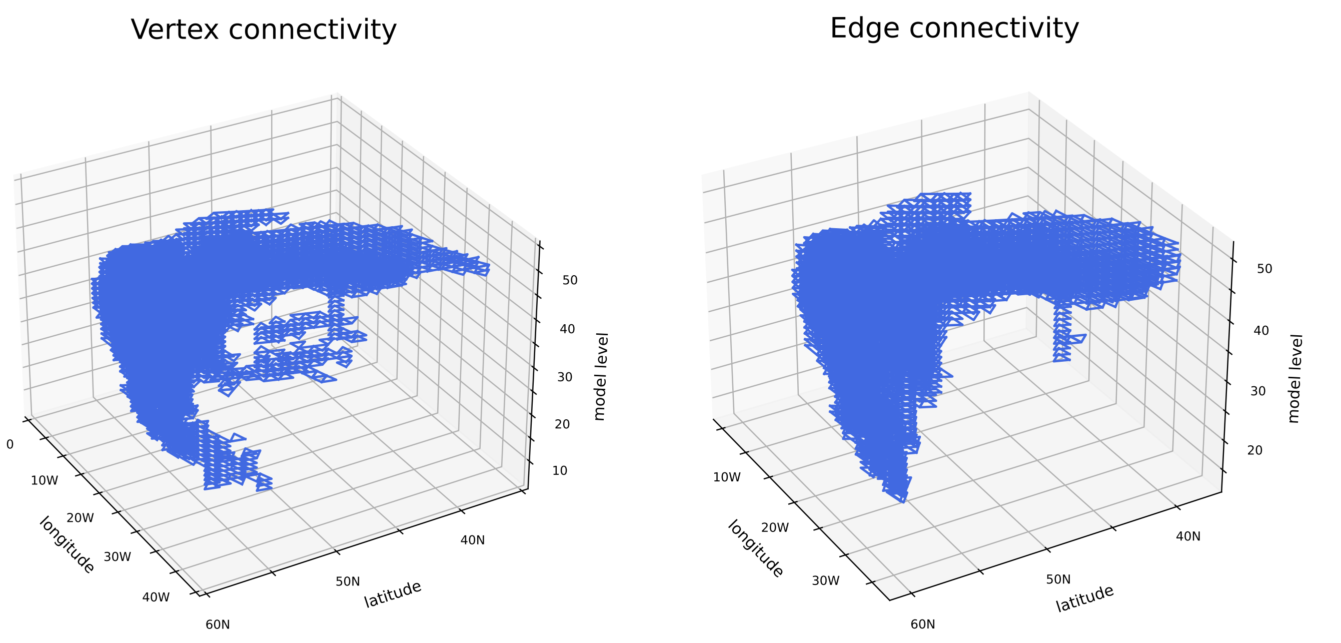

Figure 12. Application to vertically-varying cloud cover from the ICON simulation also used in Fig. 11. The plot shows the connected component corresponding to the commashaped cloud band near 20 W and 50 N for vertex connectivity (left) and edge connectivity (right). The model levels are counted upward from the Earth surface so that level 10 is near the surface and level 55 in the upper troposphere.

Horizontal resolution in km	80	40	20	10
Triangular grid				
Cells	7,920	31,728	127,052	508,988
Vertices	4,089	16,121	64,042	255,528
Edges	12,008	47,848	191,093	764,515
Cubulation				
Start cell	5,570	18,494	69,220	264,617
Search radius	104	210	423	851
Size of cubulation	105^3	211^3	400^3	802^3

Table 1. Size of the triangular grids used for benchmarking, as well as characteristics of the associated cubulations.

We measure the time needed for Steps 2, 3 and 4 described in Sect. 3. The computational cost for Step 1 is virtually zero and not considered. As in Sect. 3.5 we use total cloud cover for single-level data, and cloud fraction on 75 model levels for multi-level data.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

The time required to compute the cubulation (Step 2) increases strongly with the number of grid cells (Tab. 2). For the coarse 80 km grid with 7,920 cells, the cubulation is computed within a few seconds; for the 10 km grid with 508,988 cells this step takes 3 hours. We find this result encouraging as it shows that even our rather naive implementation can handle grids that contain up to 500,000 cells. To put that number into perspective, global simulations with ICON in climate mode are run using grids with 20,480 cells (R2B4 resolution; Giorgetta et al., 2018), and global simulations with ICON in weather-prediction mode for research purposes typically use grids with 327,680 cells (R2B6 resolution; Selz, 2019; Baumgart et al., 2019). Such grids are accessible already with our current implementation, in particular since the cubulation needs to be computed only once. Nevertheless, there is a clear need for improvement if one aims to handle larger grids, including those used in global storm-resolving models (Satoh et al., 2019).

Tab. 2 also includes the times required for reading and preparing the simulation data (Step 3), and for computing connected components (Step 4). In real-world applications of TriCCo, both steps are done together. Here, their times are separated to help identify performance bottlenecks. The times are obtained from 10 repetitions of analyzing 48 output time steps and are given as the average time required for a single time step. An exception is the 10 km grid for multi-level data, for which the time is obtained from a single analysis of 10 time steps due to the immense computational expense and the 8-hour wall-clock limit for a batch job on Mistral's compute partition. All times are given per single time step.

The time for computing the connected components dominates the time for reading and preparing the data. For single-level data, computing the connected components requires only a few seconds even for the 10 km grid and in fact is (much) less than 1 s for the smaller-sized grids. This shows that the current implementation is feasible for single-level data. For multi-level data, however, the picture is mixed. The time stays within a few seconds for the 80 km grid but increases drastically as the grid size is increased. For the 20 and 10 km grids analyzing a single time step takes 1 and 5 minutes, respectively. This limits, and for large datasets with many time steps in fact may inhibit the application for grids of this size.

Besides speed, another matter of concern is the amount of required main memory. In the current implementation, the entire data on the cubic grid needs to be hold in memory. The size of this data increases with the size of the cubulation, which is included in the last line of Tab. 1. For example, the cubulation of the 10 km grid consists of $802^3 = 516 \cdot 10^6$ cells, meaning that the cubulated version of single model-level data requires approximately 500 MByte of memory if one assumes the data is stored as 1-Byte integers. For multi-level data on, e.g., 75 levels, the requirement increases to 36 GByte. TriCCo's thirst for memory thus can become immense, and while it might be satisfied on high-performance computers, it poses a problem for the general applicability of TriCCo.

The need to reduce the amount of required memory is also evident from a consideration of information density. On the triangular grid, each cell contains information and the information density is maximum. When the data is moved

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021

Horizontal resolution in km	80	40	20	10
Step 2: Compute cubulation	12 s	75 s	12 m	3 h
Single-level data				
Step 3: Read and prepare data	0.05 s	0.1 s	0.3 s	1 s
Step 4: Vertex connectivity	0.02 s	0.1 s	0.5 s	3 s
Step 4: Edge connectivity	0.02 s	0.1 s	0.5 s	3 s
Multi-level data (75 levels)				
Step 3: Read and prepare data	0.5 s	2 s	10 s	40 s
Step 4: Vertex connectivity	2 s	12 s	60 s	330 s
Step 4: Edge connectivity	3 s	20 s	100 s	520 s

Table 2. Time required for different aspects of TriCCo's python implementation for different sizes of the triangular grid.

onto the cubic grid, the vast majority of cells in fact do not correspond to a cell on the triangular grid and contain no information. I.e., the information density is very low. A striking example is the 10 km grid, for which out of the 516 million cells of the cubic grid only 508,988 correspond to a cell on the triangular grid, that is less than 0.1%. Put differently, the data moved to the cubic grid data is a very sparse matrix whose entries for the overwhelming part are trivial 0s.

5 Conclusions

In this work we have developed a new method for identifying connected components for data on triangular grids. We have provided a python implementation of the method named TriCCo and have illustrated its use. In addition we have benchmarked its computational performance.

The principle of the method is to map the triangular grid to a structured cubic grid with self-evident neighbor relationships. This allows us to identify connected components on the triangular grid by identifying them on the cubic grid, and to rely on existing software to achieve the latter. We consider this a key strength of our method. Another key strength is that the cubulation, i.e., the mapping between the triangular and cubic grids, needs to be computed only once, which we expect to be beneficial for analysing many data fields for the same grid.

TriCCo performs reasonably fast for grids up to 100,000 triangles. As such, we find TriCCo to be ready for analyzing model output from simulations with grid sizes typical for current global multi-year climate simulations and shorter numerical weather prediction simulations. However, TriCCo in its current form is too slow for larger grids. Another issue is that when the data is mapped onto the cubic grid, the generated 3-dimensional matrix that represents the data on the cubic grid is very sparse and has a very low information density. This leads to an excessive memory usage and, together with the low speed, makes it impractical to analyse grids with 500,000 triangles and more.

TriCCo is made available open source. We welcome contributions from data and computational scientists to study if and how TriCCo can be improved. At the same time, we welcome climate and atmospheric scientists as well as more broadly colleagues from other geoscientific disciplines to use TriCCo for their research.

Code availability. The python implementation of TriCCo is available at https://gitlab.phaidra.org/climate/tricco and can be installed from pypi. The gitlab repository contains example scripts and ICON example data that illustrate the application of TriCCo and reproduce Figs. 11 and 12. Version 1.0.0 described and used in this paper is long-term archived at zenodo with doi:10.5281/zenodo.5774313.

Author contributions. AV and PS initiated, conceptualized and administered the project. NR and NK developed an initial python implementation of TriCCo that was then further developed and curated by AV. PS developed the mathematical aspects with support by NR and NK, AV led the application aspects with support from NR and NK. All authors wrote, edited and reviewed the manuscript.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. This research was supported by a YIN Award grant from the Young Investigator Network of the Karlsruhe Institute of Technology and by seeding funding from the Centre MathSEE: Mathematics in Sciences, Engineering, and Economics of Karlsruhe Institute of Technology. AV acknowledges supported by the German Ministry of Education and Research (BMBF) and FONA: Research for Sustainable Development (www.fona.de) under Grant Agreement 01LK1509A. The TriCCo package was developed and tested on the Mistral supercomputer of the German Climate Computing Center (DKRZ) in Hamburg, Germany.

We are extremely thankful to the communities of developers and maintainers of the open source python packages numpy, xarray, connected-components-3d, networkx, matplotlib and potly that are all used in the TriCCo package. We also thank the Phaidra service of University of Vienna for hosting the gitlab repository.

References

Baumgart, M., Ghinassi, P., Wirth, V., Selz, T., Craig, G. C., and Riemer, M.: Quantitative View on the Processes Governing the Upscale Error Growth up to the Planetary Scale Using a Stochastic Convection Scheme, Mon. Wea. Rev., 147, 1713–1731, https://doi.org/10.1175/MWR-D-18-0292.1, 2019.

Bridson, M. R. and Haefliger, A.: Metric spaces of non-positive curvature, vol. 319 of *Grundlehren der Mathematischen Wissenschaften*, Springer-Verlag, Berlin, 1999.

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., and Taylor, K. E.: Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization, Geosci. Model Dev., 9, 1937–1958, https://doi.org/10.5194/gmd-9-1937-2016, 2016.

Giorgetta, M. A., Brokopf, R., Crueger, T., Esch, M., Fiedler, S., Helmert, J., Hohenegger, C., Kornblueh, L., Köhler, M., Manzini, E., Mauritsen, T., Nam, C., Raddatz, T., Rast, S., Reinert, D., Sakradzija, M., Schmidt, H., Schneck, R., Schnur, R., Silvers, L., Wan, H., Zängl, G., and Stevens, B.: ICON-A, the Atmosphere Component of the ICON Earth System Model: I. Model Description, Journal of Advances in Modeling Earth Systems, 10, 1613–1637, https://doi.org/10.1029/2017MS001242, 2018.

Jakub, F. and Mayer, B.: The role of 1-D and 3-D radiative heating in the organization of shallow cumulus convection and the formation of cloud streets, Atmospheric Chemistry and Physics, 17, 13 317–13 327, https://doi.org/10.5194/acp-17-13317-2017, 2017.

Licón-Saláiz, J., Ansorge, C., Shao, Y., and Kunoth, A.: The Structure of the Convective Boundary Layer as Deduced from Topological Invariants, Boundary-Layer Meteorol., 176, 1–12, https://doi.org/https://doi.org/10.1007/s10546-020-00517-w, 2020.

Muszynski, G., Kashinath, K., Kurlin, V., Wehner, M., and Prabhat: Topological data analysis and machine learning for recognizing atmospheric river patterns in large climate datasets, Geosci. Model Dev., 12, 613–628, https://doi.org/10.5194/gmd-12-613-2019, 2019.

Neggers, R. A. J., Jonker, H. J. J., and Siebesma, A. P.: Size Statistics of Cumulus Cloud Populations in Large-Eddy Simulations, J. Atmos. Sci., 60, 1060–1074, https://doi.org/10.1175/1520-0469(2003)60<1060:SSOCCP>2.0.CO;2, 2003.

Rempel, M., Senf, F., and Deneke, H.: Object-Based Metrics for Forecast Verification of Convective Development with Geostationary Satellite Data, Mon. Wea. Rev., 145, 3161–3178, https://doi.org/10.1175/MWR-D-16-0480.1, 2017.

Rieck, M., Hohenegger, C., and van Heerwaarden, C. C.: The Influence of Land Surface Heterogeneities on Cloud Size Development, Mon. Wea. Rev., 142, 3830–3846, https://doi.org/10.1175/MWR-D-13-00354.1, 2014.

Sageev, M.: Ends of Group Pairs and Non-Positively Curved Cube Complexes, Proc. Lond. Math. Soc., s3-71, 585–617, https://doi.org/https://doi.org/10.1112/plms/s3-71.3.585, 1995.

Satoh, M., Stevens, B., Judt, F., Khairoutdinov, M., Lin, S.-J., Putman, W. M., and Düben, P.: Global Cloud-Resolving Models, Current Climate Change Reports, 5, 172–184, https://doi.org/10.1007/s40641-019-00131-0, 2019.

Schemann, V. and Ebell, K.: Simulation of mixed-phase clouds with the ICON large-eddy model in the complex Arctic environment around Ny-Alesund, Atmos. Chem. Phys., 20, 475–485, https://doi.org/10.5194/acp-20-475-2020, 2020.

Schwer, P.: Lecture notes on CAT(0) cubical complexes, AMS Open Math Notes, OMN:201907.110800, 2019.

https://doi.org/10.5194/gmd-2021-349
Preprint. Discussion started: 22 December 2021
© Author(s) 2021. CC BY 4.0 License.

Schäfer, S. A. K., Hogan, R. J., Klinger, C., Chiu, J. C., and Mayer, B.: Representing 3-D cloud radiation effects in two-stream schemes: 1. Longwave considerations and effective cloud edge length, J. Geophys. Res. Atmos., 121, 8567–8582, https://doi.org/https://doi.org/10.1002/2016JD024876, 2016.

Selz, T.: Estimating the Intrinsic Limit of Predictability Using a Stochastic Convection Scheme, J. Atmos.Sci., 76, 757–765, https://doi.org/10.1175/JAS-D-17-0373.1, 2019.

Senf, F., Voigt, A., Clerbaux, N., Hünerbein, A., and Deneke, H.: Increasing Resolution and Resolving Convection Improve the Simulation of Cloud-Radiative Effects Over the North Atlantic, Journal of Geophysical Research: Atmospheres, 125, e2020JD032 667, https://doi.org/10.1029/2020JD032667, 2020.

Stevens, B., Acquistapace, C., Hansen, A., Heinze, R., Klinger, C., Klocke, D., Rybka, H., Schubotz, W., Windmiller, J., Adamidis, P., Arka, I., Barlakas, V., Biercamp, J., Brueck, M., Brune, S., Buehler, S. A., Burkhardt, U., Cioni, G., Costa-Suros, M., Crewell, S., Crüger, T., Deneke, H., Friedrichs, P., Henken, C. C., Hohenegger, C., Jacob, M., Jakub, F., Kalthoff, N., Köhler, M., Laar, T. W. v., Li, P., Löhnert, U., Macke, A., Madenach, N., Mayer, B., Nam, C., Naumann, A. K., Peters, K., Poll, S., Quaas, J., Röber, N., Rochetin, N., Scheck, L., Schemann, V., Schnitt, S., Seifert, A., Senf, F., Shapkalijevski, M., Simmer, C., Singh, S., Sourdeval, O., Spickermann, D., Strandgren, J., Tessiot, O., Vercauteren, N., Vial, J., Voigt, A., and Zängl, G.: The Added Value of Large-Eddy and Storm-Resolving Models for Simulating Clouds and Precipitation, Journal of the Meteorological Society of Japan, 98, 395–435, https://doi.org/10.2151/jmsj.2020-021, 2020.

Voigt, A.: TriCCo v1.0.0 - a cubulation-based method for computing connected components on triangular grids, https://doi.org/10.5281/zenodo.5774313, 2021.

Wedi, N. P., Polichtchouk, I., Dueben, P., Anantharaj, V. G., Bauer, P., Boussetta, S., Browne, P., Deconinck, W., Gaudin, W., Hadade, I., Hatfield, S., Iffrig, O., Lopez, P., Maciel, P., Mueller, A., Saarinen, S., Sandu, I., Quintino, T., and Vitart, F.: A Baseline for Global Weather and Climate Simulations at 1 km Resolution, Journal of Advances in Modeling Earth Systems, 12, e2020MS002 192, https://doi.org/10.1029/2020MS002192, 2020.

Zängl, G., Reinert, D., Ripodas, P., and Baldauf, M.: The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core, Q. J. Roy. Meteor Soc., 141, 563–579, https://doi.org/10.1002/qj.2378, 2015.