On the problem of providing unique identifiers for areas with any shape on Discrete Global Grid Systems

Rubén Béjar[1], Miguel Á. Latre, Francisco J. Lopez-Pellicer,
Javier Nogueras-Iso, F. Javier Zarazaga-Soria
Aragon Institute of Engineering Research / Universidad
Zaragoza
[1]rbejar@unizar.es

Abstract

This short paper explores the problem of assigning a unique identifier to any area, i.e. arbitrary size and shape, defined on a Discrete Global Grid System (DGGS). The problem is framed in terms of the components of any DGGS, multi-resolution cell tessellations with unique identifiers for each cell, what takes us to areas defined as sets of cells. Two solutions to create unique identifiers for sets of cells, and thus for any area on a DGGS, are experimentally compared to measure their performance, the shorter the better, in different cases. The issue of constraining identifiers to a fixed maximum length is also considered. Some calculations are provided in order to give an idea of the magnitude of these two problems and the scope of the proposed solutions.
Keywords: DGGS; Geocoding; Addressing.

1 Introduction

The problem of assigning names or identifiers to places on the Earth has been addressed with different approaches.

Geographical names and street addresses are a solution for some places, but they are neither systematic nor unique and, in many cases, their positions and shapes are only approximate.

Coordinate systems, e.g. geodetic coordinates, can be used to identify points on the surface of the Earth with a pair of Real numbers. Using Real numbers as identifiers, i.e. finite length strings, requires rounding them. Besides this, most Real numbers can't be represented exactly in the standard format for floating point Real numbers in computers and are automatically approximated (IEEE, 2008), although this can be addressed with alternative encodings that can be as precise as needed.

Global grids provide an alternative by partitioning the Earth into cells. Discrete Global Grid Systems (DGGSs), multi-resolution grids with non-overlapping, equal-area, uniquely identified cells, provide a solution to identify places on Earth by pointing out, for instance, the smallest cell, up to a maximum resolution, which includes those places.

We are often interested in identifying places with complex shapes. On a DGGS, this means identifying a set of cells, not only one. This paper explores options to give unique identifiers to sets of cells, and thus to areas with any shape.

2 Related work

A Discrete Global Grid System (DGGS) is a spatial reference system based on a hierarchical multi-resolution grid of cells which form equal area tessellations of the surface of the Earth, without any overlapping, at every resolution (Purss, 2017). Besides this, a DGGS must define methods to address, i.e. identify, each cell, assign data to the cells and perform algebraic operations on them and their data. In this paper we use the term DGGS as defined by this OGC abstract specification.

The OGC specification does not require a given system to create the cell addresses as long as they fulfill the requirements. In (Amiri, et al., 2015) they describe three general strategies: hierarchy-based, space-filling curves and axes-based indexing

With a hierarchy-based index you give an identifier to the cells of a given resolution, and then you use those identifiers as prefixes for the cells at the next resolution. The rHEALpix DGGS uses this (Gibb, 2016). In rHEALPix, N, S, O, P, Q and R are the six identifiers for the cells of resolution 0. If you subdivide N into 4 cells (nside = 2) at resolution 1, their identifiers will be N0, N1, N2 and N3. If you subdivide N0 into 4 cells at resolution 2, their identifiers will be N00, N01, N02 and N03. And so on.

Space-filling curves are 1D curves that, when created recursively, end up covering a 2D space. The identifiers are typically constructed with digits from a suitable numeric base which depends on the refinement (i.e. on how you subdivide the cells). For each additional resolution you add a new digit. For example in (Bartholdi & Goldsman, 1999) they define an indexing method based on the Sierpinski space-filling curve for triangular cells and base 2 digits, with valid cell ids such us 0 (1 subdivision), 011 (3 subdivisions) or 00000 (5 subdivisions).

In axes-based indexing you define m axes (e.g. m = 2 for a 2D DGGS) and enumerate the cells along those axes. Refinements are solved by adding an additional number for the resolution. In the cube-based example in (Amiri, et al.,

2015, Figure 10), a cell at resolution 1 can have an identifier such as [3,(0,0)]1 (third face of the cube, lower-left cell at resolution 1), which would be subdivided into 4 cells at resolution 2 with identifiers [3,(0,0)]2, [3,(0,1)]2, [3,(1,0)]2 and [3,(1,1)]2.

Alphanumeric geocoding systems provide unique, human friendly identifiers for locations which are either points or cells of the same shape, usually squares or triangles, with a limit on the resolution or accuracy. They are often based on Discrete Global Grids, but not always. A well-known example is what3words (Jones, 2015), which assigns three fixed words to each square cell of 3m × 3m on the Earth and is being proposed as an alternative addressing system (Howgego, 2017). It has some extension proposals, such as (Jiang & Stefanakis, 2018) which would add variable resolution and support for a vertical dimension by using up to five words, and even some satiric alternatives, such as what3fucks* which uses swear words and triangular cells. There are other systems, such as 3geonames†, which encodes points, not cells, with an accuracy of up to 1m, and GeoKeys‡ which encode 1m × 1m squares with 4, 4-letter words.

3 Creating unique identifiers for areas on a DGGS

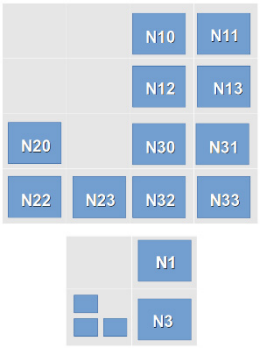
A DGGS must provide a way to address each of its cells with a unique identifier (a **Cell Unique Identifier, CUID**). Any area on a DGGS is formed by a set of cells, possibly mixing cells of different resolutions.

There are many possible areas in any DGGS. For instance, in a DGGS that has 6 cells at resolution 0, and refines each one into 9 at the next resolution level, there will be 6×9^R cells at resolution R, e.g. 486 at R=2. The number of areas that can be defined by using only cells of a given resolution is equal to the number of non-empty sets of cells that can be chosen at that resolution, which is $2^{(\#\text{ of cells at resolution})}-1$. Following the example, we have $2^{486}-1$ possible areas with cells of resolution 2, which is a number with 148 digits.

In practice, we may not need to define areas with a very large number of cells thanks to the hierarchical nature of the DGGSs. As an example of this, Figure 1 shows that 11 cells at resolution R (upper square) are just 2 cells at resolution R-1 (lower square) and 3 cells at resolution R.

The example in Figure 1 also shows that areas on a DGGS may be defined with different sets of cells. For instance the cell with CUID N1 and the cells with CUIDs N10, N11, N12 and N13 cover the same area. Let's define **an optimal CUID set** for an area on a DGGS as the set of the CUIDs of the smallest set of cells that cover exactly that area. Given the requirements of a DGGS (non-overlapping cells which are a tessellation at every resolution, and exactly one way to refine cells) it follows that there is exactly one optimal CUID set for any area on a given DGGS.

Figure 1: The same area with cells of the same resolution, and with cells of two different resolutions



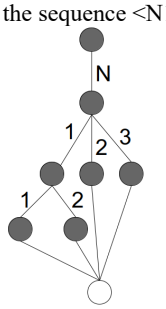
Once we have an optimal CUID set, for example {N11, N12, N2, N3}, we need to encode it as a sequence of symbols that can be used as a unique identifier. Let's call this an **Area Unique Identifier (AUID)**.

For this, we need to order the elements of the set with any total order relation, for instance the alphabetical order, and to encode that ordered set as a sequence. The simplest option is a concatenation of the ordered CUIDs: N11N12N2N3. It may be necessary for some DGGSs to use an additional symbol to separate the CUIDs so it is possible to parse the resulting AUID and extract the original CUIDs.

As we will need to store these AUIDs and, possibly, to send them over the Internet, we prefer them as short as possible. The multi-resolution, hierarchical structure of cells in a DGGS suggests a tree structure as a natural choice to encode them in an efficient way.

Tries, also known as prefix trees, are ordered trees which can be used to store data associated to certain keys (Knuth, 1998, p. 192). These keys are usually strings of characters, and a common use of tries is just storing those strings. The structure of a trie allows for a compact representation of a number of strings. For example, the sequence <N11, N12, N2, N3> can be expressed as a trie as shown in Figure 2.

Figure 2: A trie for the sequence <N11, N12, N2, N3>



To use a trie as an AUID, we need to linearize it as a sequence. The Balanced Parentheses (BP) strategy is simple and generates short strings (Munro & Raman, 2001). The BP strategy allows to encode the structure of any tree, but we need to differentiate trees with the same structure that correspond to different CUID sets. We thus need to encode the CUIDs along with the parentheses. For instance, the trie in Figure 2 can be expressed as a BP string as (R(N(1(1($))(2($)))(2($))(3($)))), using R to denote the root

* http://www.what3fucks.com/
† https://3geonames.org/
‡ https://geokey.xyz/

node, and $ to denote the special end node (the white one in the Figure).

Including the CUIDs in the BP string provides us with an opportunity to make that string shorter. As every branch is labeled with one symbol, including R and $, we do not need the opening parentheses because their role in the BP string can be fulfilled by those symbols. This short BP string for the trie in the Figure would be RN11$))2$)))2$))3$)))).

3.1 Experimental results

The BP strings shown in the previous section are not shorter than the concatenation of the CUIDs. However the example already shows two important characteristics:

1. Parentheses in BP strings are just two symbols, or just one if using the short version, repeated many times. Any lossless compression algorithm will make a significant reduction in their length because of this.
2. The trie benefits from some repetitions in the original CUIDs. For instance there are only two 1s in the BP string, but there are three in the concatenation of CUIDs. The longer the CUIDS, the more relevant this saving will become.

Those two characteristics suggest that with sufficiently large CUIDs, the compressed BP strings may be shorter than the compressed concatenations[§].

To test this hypothesis, we have run a number of simulations, see Figure 3, generating different random sets of CUIDs and creating the AUIDs for them as simple concatenations (blue lines with squares), tries expressed as BP strings (red lines with stars) and tries expressed as BP strings without the opening parentheses (green lines with circles). In the three cases we have applied the same lossless compression algorithm to the resulting AUIDs.

The code used to run these experiments and to produce the Figure 3 is available at https://github.com/IAAA-Lab/dggs-auids as a Jupyter Notebook[**] with code in Python 3.6.

In each plot in Figure 3, the number of CUIDs used to generate the AUIDs is in the horizontal axis, and the size of the AUIDs, in bytes, is in the vertical axis. Each plot is labeled with the range of resolutions where the random CUIDs have been produced, if the set is an optimal CUID set or not, and the indexing strategy used (axis-based following the pattern described in Section 2, or rHEALPix style, i.e. prefix-based, with nside = 2 or with nside = 3). For instance, "Res [6,9]. Non-optimal set, axis CUIDs", means CUIDs of resolutions 6, 7, 8 and 9, with axis-based indexing and the set is not optimal (there may be repeated CUIDs, and it is possible that there are more cells than needed).

For this initial work, we have chosen two different indexing strategies, and ranges of resolutions which allow us to generate thousands of CUIDs. We have also focused our efforts on non-optimal sets, as the optimal ones are far slower to generate with the current version of the testing scripts (that is the reason why they have been tested with less CUIDs). Of course, the three different ways to create the AUIDs have always been compared among them under the same conditions.

The results show that the trie as a compressed BP string without the opening parentheses, i.e. 'short BP string', is generally shorter than the concatenation and compression of the CUIDs, although the differences are not very large. In the best cases, compressed short BP strings are around 25% shorter than the compressed concatenations. These differences usually tend to grow as the size of the CUID sets increases. However this result is not universal, and there are a few cases where the compressed concatenations of the CUIDs are slightly, around 2%, shorter than the compressed BP strings.

3.2 Shortening the AUIDs

The number of the AUIDs that can be generated is huge for any DGGS, so they may need to be very long. Following the previous example, to identify 2^{486}-1 possible areas at resolution 2, at least 486 bits, some 60 bytes, are required. To identify all the areas at resolution 10 (cells of around 100 m ×100 m in that example) we would need to use identifiers of 20,920,706,406 bits (6×9^{10}), some 2.6 Gigabytes. Producing **short AUIDs**, with a fixed maximum length, may be necessary for instance to use them safely as part of URLs.

We can associate short identifiers to AUIDs in any arbitrary way, for instance using correlative numbers, but a better solution is using a hash function, because this guarantees that anyone can generate the same short version for the same AUID. With, for example, 256 bit hashes we have 2^{256} possible unique identifiers. Although there are many more possible AUIDs in any DGGS, 2^{256} is approximately 1.16×10^{77}, which is still huge.

It is impossible to recover the CUIDs from the hash of an AUID. In order to maintain that correspondence, we need to store the AUIDs with their hashes, and to provide a lookup mechanism to recover the AUID for any given hash. So even if we wanted to use only the hashes, making the original AUIDs as short as possible is still a relevant issue to save storage space.

4 Conclusion

In this short paper we have started to explore the problem of providing identifiers to places on Earth when we need them for arbitrary areas defined on a DGGS.

We have shown that the problem consists of choosing the smallest set of cells that covers our area of interest, up to a maximum resolution, and producing a sequence of symbols that encodes the DGGS identifiers of those cells.

We have also shown that a shorter, fixed length, version can be produced with a hash function. This can be used where there are strict limitations on size, as long as we provide a lookup mechanism to recover the original CUIDs from a hash.

There are many proposed DGGSs, with different cell shapes, refinement strategies and indexing schemes. Our initial experiments have included two different indexing schemes, axis-based and prefix-based, and a refinement

[§] Compression will be in general a good idea as we intend to have identifiers which are as short as possible.

[**] https://jupyter.org/

strategy where each cell at resolution R+1 is fully contained in a single cell at resolution R. For those, we have shown that encoding the cell identifiers in a trie and then linearizing it as a compressed BP string typically produces shorter identifiers than the simplest approach, which is a compressed concatenation.

Identifiers such as those explored in this paper are mainly intended to facilitate the automatic storage, search and retrieval of datasets georeferenced on DGGSs. Nevertheless, systematic ways to produce human-friendly names for them could be designed. The main issue would be that there are too many possible different areas. But for limited resolutions or certain subsets, this could be solved.

As future work, there are several paths to explore:

- More indexing schemes and refinement strategies.
- Areas based on real features instead of randomly generated.
- New strategies to produce shorter AUIDs.
- Mechanisms to produce human-friendly strings for the AUIDs.

Acknowledgments

This work has been partially supported by the Aragon Government (project T59_17R) and the Spanish Government (project TIN2017-88002-R).

References

Amiri, A. M., Samavati, F. & Peterson, P., 2015. Categorization and Conversions for Indexing Methods of Discrete Global Grid Systems. *ISPRS International Journal of Geo-Information,* Volume 4, pp. 320-336.

Bartholdi, J. & Goldsman, P., 1999. Continuous Indexing of Hierarchical Subdivisions of the Globe. *International Journal of Geographical Information Science,* Volume 15, pp. 489-522.

Gibb, R., 2016. *The rHEALPix Discrete Global Grid System.* s.l., IOP Conference Series: Earth and Environmental Science, vol 34, p. 012012.

Howgego, J., 2017. Where in the world?. *New Scientist,* Volume 235, pp. 30-32.

IEEE, 2008. *IEEE Standard for Floating-Point Arithmetic,* s.l.: The Institute of Electrical and Electronics Engineers, Inc..

Jiang, W. & Stefanakis, E., 2018. What3Words Geocoding Extensions. *Journal of Geovisualization and Spatial Analysis,* Feb, Volume 2, p. 7.

Jones, G. R., 2015. Human Friendly Coordinates. *GeoInformatics,* Volume 18, pp. 10-12.

Knuth, D. E., 1998. *The Art of Computer Programming: Volume 3: Sorting and Searching.* 2 ed. s.l.:Addison-Wesley Professional.

Munro, J. I. & Raman, V., 2001. Succinct Representation of Balanced Parentheses and Static Trees. *{SIAM} J. Comput.,* Volume 31, pp. 762-776.

Purss, M., ed., 2017. *The OpenGIS Abstract Specification - Topic 21: Discrete Global Grid Systems Abstract Specification).* s.l.:Open Geospatial Consortium.

Figure 3: Sizes of the AUIDs generated with three different strategies for a number of random sets of CUIDs