The Eternity Puzzle: A Linear Algebraic Solution

Marcus Garvie, John Burkardt

06 June 2021

1 Introduction

In 1999, Christopher Monckton introduced a geometric dissection puzzle known as "Eternity: the puzzle worth a million", for which a million pound reward was offered for the first correct solution. To the surprise of the proposer, such a solution was reported in 2000 by Alex Selby and Oliver Riordan, and the prize was duly awarded. The solution of the puzzle involved exploiting a hypotheses that there were actually many solutions, and that a hybrid approach could identify favorable partial initial placements, after which an exhaustive search algorithm could check whether the remaining pieces could all be positioned.

The Eternity puzzle is a member of a large class of exact cover problems, which can be thought of as consisting of a fixed *grid*, composed of a number of adjacent polygonal elements, and a number of *tiles*, each composed of a smaller number of elements. A solution of the puzzle can be thought of as rule for laying the tiles down onto the grid in such a way that each grid element is covered exactly once by a tile element, and each tile element is matched to exactly one grid element.

Selby and Riordan used a two-phase algorithm, which first set up a "favorable state" of a certain size in which the most awkward pieces are positioned and the remaining uncovered region has a favorable shape, followed by an exhaustive search of all possible placements of the remaining "easy-to-place" tiles.

It is our purpose to return to the Eternity puzzle, and consider a reformulation of the problem as an underdetermined system of linear algebraic equations, for which there are efficient, reliable methods that produce all solutions in an automatic procedure.

To carry out this reformulation requires associating an equation with each element of the grid, and a variable with each legal orientation of each tile. A legal orientation of a tile is any combination of translation, reflection, and rotation, which matches each element of the tile with exactly one element of the grid.

2 The Geometry of the Eternity Grid

The grid for the Eternity puzzle appears as Figure 1. On a first glance at the grid, a regular pattern of equilateral triangles appears. A closer look reveals an underlying finer grid of 30-60-90 triangles, which make geometric calculations more complicated.

The Eternity puzzle has several notable characteristics:

- The grid uses a *fine grid* of 2,508 elements, which are 30-60-90 triangles; these elements are sometimes called *drafters*, since they share the shape of the drafter's triangle;
- The grid includes a *medium grid* of equilateral triangles each formed from 6 of the elements; however, some triangles are only partially formed because they intersect the boundary of the grid;
- The grid includes a coarse *hex grid* of hexagons, each formed from 12 of the elements. If we wish to locate any specific element in an Eternity-type grid, we can do so by specifying a particular hexagon,

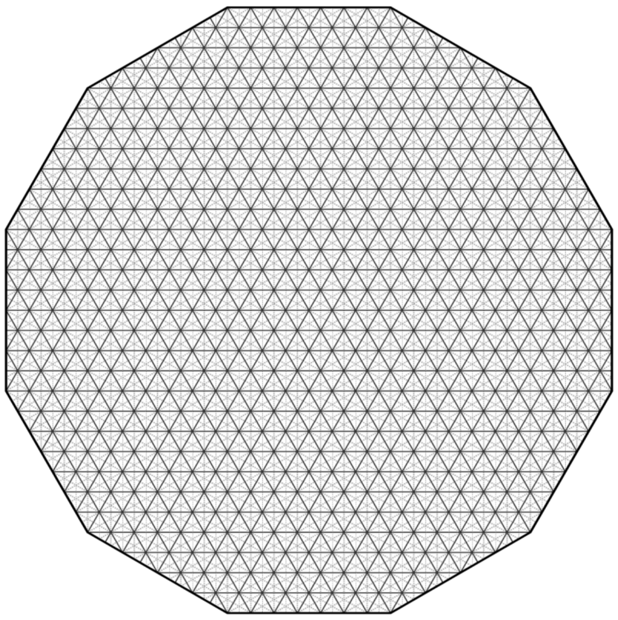


Figure 1: The Eternity Grid: The fine grid of 30-60-90 triangles is outlined in light gray. The medium grid of equilateral triangles is in darker gray. Many hexagons are easily visible, but they overlap. The coarse grid is constructed by selecting staggered rows of them. It is clear that some triangles and hexagons along the border are only partially formed.

and then the rotation degrees, or clock-position, of the element. This idea offers us a clue into how to construct a coordinate system.

- The grid also reveals small rectangular structures involving 6 elements, and squares of 12 elements can be formed. However, it will be to our advantage to consider a coarse *rectangular grid* whose basic cell is a vertical stack of two squares, comprising 24 elements. This cell can tile the plane to any width and height desired. By analyzing a single rectangular cell, we will later be able to automatically number the nodes and elements of any finite Eternity object, and identify the nodes that are the vertices of each element.
- The Eternity grid is in the shape of a slightly irregular dodecagon, a 12-sided polygonal figure. The irregularity arises because the sides alternate in length between 7 units and $8\frac{\sqrt{3}}{2} \approx 6.9292$ units, assuming that the equilateral triangles in the medium grid have a side length of 1 unit.
- There are 209 tiles; each tile is formed from 36 elements; each tile is unique; no tile has any symmetry; a tile that is flipped over has a distinct shape. Because they are formed from drafters, the tiles are sometimes called *polydrafters.*

It will be necessary to impose a measurement scale on the grid. To do this, we make the convenient assumption that the equilateral triangles in the grid have a side length of one unit. From this, we can deduce measurements for the various interesting features of the grid:

- Each 30-60-90 element has sides of length $\frac{\sqrt{3}}{6}$, $\frac{1}{2}$, and $\frac{\sqrt{3}}{3}$, opposite the angles of 30°, 60°, and 90° respectively
- Each equilateral triangle has sides of length 1, and a vertical height of $\frac{\sqrt{3}}{2}$.

- Each rectangle (a vertical stack of two squares) has horizontal width 1 (one equilateral triangle width) and a vertical height of $\sqrt{3}$ (two equilateral triangle heights).
- Each hexagon has horizontal width 1 (or from the middle of any side to the middle of the opposite side) and a vertical height of $\frac{2\sqrt{3}}{3}$ (or from any vertex to the opposite vertex).

Because of the use of 30-60-90 triangles as the elements of the fine grid, it is a difficult task to construct a coordinate system that is efficient in locating elements, identifying their neighbors, defining a tile that is composed of a specific arrangement of elements, and generating the legal transformations of a tile.

It is the purpose of this note to record the decisions made in choosing such a coordinate system, and to describe how the algorithms for various computational task rely on this coordinate system.

3 Characteristics of the 30-60-90 Triangles

Many geometric dissection puzzles rely on an underlying grid of identical cells, which are typically squares or triangles, as in puzzles involving polyominoes or polyhexes. In the Eternity puzzle, the fundamental cell is, unusually, a 30-60-90 triangle. Geometric questions about the Eternity puzzle must take account of the properties of these elements.

- Each element has a *position.* Once we have chosen a coordinate system for the grid, we can specify the position of any element by, for example, giving the coordinates of its vertices. If we choose such a system, we will always order the vertices to appear in the same 30-60-90 ordering as their corresponding angles. Looking at the grid, we can see that a given element reappears in the grid in the exact same orientation many times. A simple *translation* operation could have been used to create such a pattern. However, we can also see that the grid is too complicated to be constructed solely by translations of one element.
- Each element has a *rotational angle.* In a grid, focus on one of the hexagons containing a full set of 12 elements. Pairs of elements are neighbors along rays at $0°, 30°, ..., 330°$. We assign a label of "1" to the element between the 3 o'clock and 2 o'clock positions, then by applying counterclockwise multiples of a $60°$ rotation to this element, we can generate 5 more elements, which we can label "2" through "6". Similarly, if we label "-1" the element between the 4 o'clock and 4 o'clock positions, we can using $60°$ rotation in a clockwise sense to generate the remaining lements, which we will label "-2" through "-6". The labels we have defined uniquely identify each of the elements in any given hexagon, and tell us how that element relates to the element labeled "1".
- Each element has a *parity*; the parity is essentially the sign of the label. For any element with a positive label, traversing the nodes in the order $30°, 60°, 90°$, results in a counterclockwise motion, corresponding to a positive sense. Correspondingly, elements with a negative label are traversed in a clockwise motion, and have a negative sense. This implies that transforming one element to another of the same parity requires only rotations and translations, but transforming to an element of opposite parity requires a *reflection* operation as well. The label of a reflected element is the negative of the original element. Thus, an element labeled "-4" has a corresponding reflection labeled "4".
- The label assigned to an element will be referred to as its *type.* The pattern of types can be viewed in Figure 2.

Now suppose that we had a trivial tiling problem, in which the tiles we were given were simply a sufficient number of identical elements, so that all we had to do was place them on the grid. We can think of a placement as a mapping from a reference tile to a physical tile in the grid. On uniform grid of squares, it is possible to carry out such a task using just translations; on uniform triangular grids, translations and rotations suffice. But for the Eternity grid, we will require a combination of translation, rotation, and reflection in order to

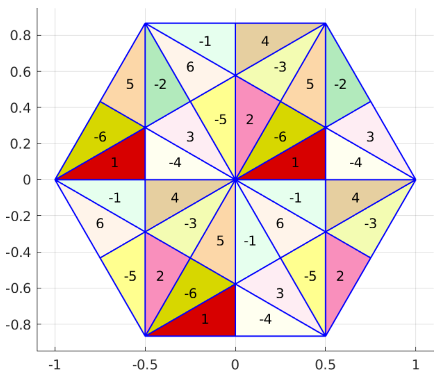


Figure 2: This hexagonal grid contains a central hexagon of 12 elements, which includes all 12 element types, distinguished by label and color, which are also displayed on similar elements in the grid.

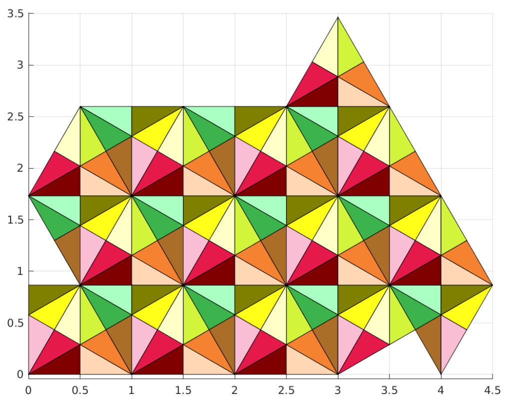


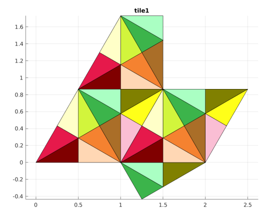
Figure 3: The Trinity Grid. There are 144 copies of the elemental 30-60-90 triangle. Twelve colors are used to distinguish the 12 possible orientations of the element. The equilateral triangles of the medium grid have a side length of 1 unit.

describe how any tile must be oriented in order to match some corresponding collection of elements on the grid. If there is actually a solution to the problem, then we should be able to describe that solution in terms of the successive application of these transformations to each of the tiles we have been given.

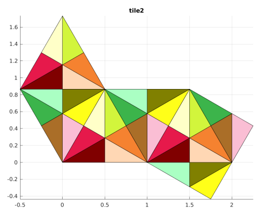
4 The Trinity Puzzle

A beginner will find it difficult to see beyond sheer size of the Eternity grid and the number of tiles employed. It will be very helpful to defer the consideration of the full puzzle for now, and instead to work with a considerably simpler puzzle, discussed by Alex Selby in one of his presentations. He did not name the puzzle, and so we have taken the liberty of referring to it as the "Trinity puzzle", since the name vaguely rhymes with "eternity", and because Alex Selby comes from Cambridge University, which has a Trinity College, and because we might see the shape of a mythical Trinity cathedral suggested by the grid, as in Figure 3.

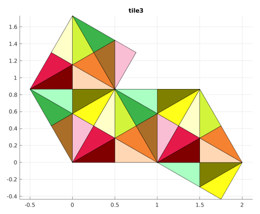
The Trinity grid is to be covered using the four tiles shown in Figure 4, which must be appropriately reflected, rotated and translated. Thes tiles exhibit some features worth mentioning, because they will also be characteristic of the Eternity tiles:



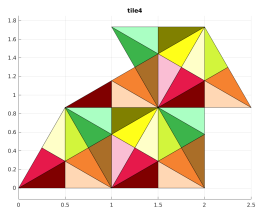
(a)



(b)



(c)



(d)

Figure 4: The Trinity Tiles. There are 4 tiles, each involving 36 of the elemental 30-60-90 triangle. Element colors are for illustration, and are not part of the matching process.

- Each tile is formed from exactly 36 of the 30-60-90 elements;
- Each tile can be formed by combining exactly 12 halves of the equilateral triangles; that is, if an element is part of a tile, than so are 2 or 5 other contiguous elements that lie in a common equilateral triangle; This fact somewhat limits the possibility of any tile having a bizarre shape.
- Notice, moreover, that any shape formed by combining three contiguous elements in an equilateral triangle is also a 30-60-90 triangle. This fact means that the simple term *”30-60-90 triangle”*, and even *“polydrafter”* can be ambiguous. Thus we prefer to say *element* when referring to the smallest triangles in the grid, and *half-triangles* to the shapes formed by three contiguous elements of an equilateral triangle.
- Each tile can be assigned a binary parity, which can be illustrated by a sort of checkboard red/black coloring. This parity is simply the difference between the number of red and black elements. A tile can be regarded as a combination of halves and wholes of equilateral triangles, and any constituent whole triangle will have a parity of zero. Hence, it is only necessary to consider the unpaired half-triangles to determine the binary parity of a tile.
- It is possible to consider higher order parities of a tile. Notice that 12 colors can be used to color the elements of any Eternity object. Instead, imagine using 6 colors, but allowing each color to be “positive” or “negative”. Thinking counterclockwise, we color 3, 1, 11, 9, 7, and 5 o’clock with (positive) red, green, blue, cyan, magenta, and yellow (RGBCMY). Then we color 4, 6, 8, 10, 12, 2. The R parity of the tile is the sum of the positive red and negative red tiles. Similarly summing the other colors, we get a parity vector of six elements. Such a parity vector can be used in various arguments about matchings and coverings.

A solution of the puzzle is suggested by Figure 5. The tiles had to be rotated and translated, and one tile also had to be reflected (“turned over”) in order to cover the region. While this problem could be solved by hand, it is the goal of the programmer to devise an algorithm that will automatically determine whether a given puzzle has no, one, or many solutions, and to describe or plot them. This, in turn, requires a symbolic representation of the region and the tiles, transformation algorithms that turn, flip, and shift the tiles, and an efficient procedure for hunting down solutions.

5 The Boundary Word of an Eternity Object

By an *Eternity object* we mean any shape constructed by selecting some of the elements of the Eternity grid, or (in general) of the Eternity grid extended to infinity. From now on, we will impose the additional assumptions that the elements of an Eternity object are connected, and that the object will not involve any internal holes.

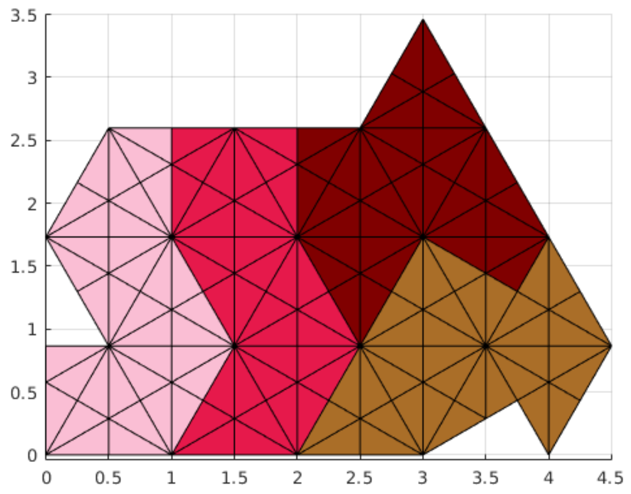


Figure 5: How 4 tiles cover the Trinity region, (i,j) coordinates

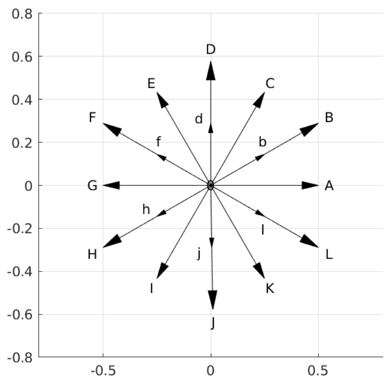


Figure 6: The Boundary Word "Compass". Each symbol indicates a direction and stepsize.

Thus, an Eternity object will always a polygon, and moreover one whose boundary can be constructed by tracing a sequence of boundary lines of the non-interior elements. These boundary lines have only 12 possible directions, and two possible lengths. This means that we can describe the shape (though not the position) of an Eternity object by listing the step directions and step lengths of a path around the boundary. Alternatively, if we are using the (i,j) coordinates system, we can immediately write down the horizontal and vertical components of the step as integers. If we adopt an appropriate set of symbols for each possible step, we can describe the shape by a symbol string known as a *boundary word*. Boundary words originated in combinatorial geometry. For our purposes, a boundary word provides a compact and useful representation of our Eternity objects.

Table 1 lists the symbols used for Eternity object boundary words. Each alphabetic character specifies a direction. In cases where two stepsizes are possible, a lower-case version of the letter is used to indicate that a half-step is taken. A sort of boundary word "compass" is displayed in Figure 6, which graphically suggests the direction and length of the steps corresponding to each symbol.

As an example, let us consider the Trinity grid, whose boundary word is **AAAAAABBbKCCEEEEEEI-IGGGGIIKKGjJ**. Figure 7 shows how each segment of the boundary of the grid corresponds to a successive

Symbol	Angle	(x,y) Stepsize	(i,j) Step
A	0°	1/2	(2, 0)
B	30°	$\sqrt{3}/3$	(2, 2)
b	30°	$\sqrt{3}/6$	(1, 1)
C	60°	1/2	(1, 3)
D	90°	$\sqrt{3}/3$	(0. 4)
d	90°	$\sqrt{3}/6$	(0, 2)
E	120°	1/2	(-1, 3)
F	150°	$\sqrt{3}/3$	(-2, 2)
f	150°	$\sqrt{3}/6$	(-1, 1)
G	180°	1/2	(-2, 0)
H	210°	$\sqrt{3}/3$	(-2,-2)
h	210°	$\sqrt{3}/6$	(-1,-1)
I	240°	1/2	(-1,-3)
J	270°	$\sqrt{3}/3$	(0,-4)
j	270°	$\sqrt{3}/6$	(0,-2)
K	300°	1/2	(1,-3)
L	330°	$\sqrt{3}/3$	(2,-2)
l	330°	$\sqrt{3}/6$	(1, 1)

Table 1: Symbol meanings for Eternity object boundary words.

letter in the boundary word.

The boundary word only encodes the outline of the polygonal Eternity object. However, using that outline, and some information about the position of the initial point, it is possible to fill in the entire diagram, that is, to "flesh out" the skeleton of the Trinity grid with the location of all the internal elements and their connectivities. This is a procedure which we will discuss in a later section.

6 Analysis of the Rectangular Cell

We have mentioned that the Eternity grid can be extended to an arbitrarily large portion of the plane by translation of a fundamental rectangular cell comprising 24 of the 30-60-90 elements. In turn, this means that any Eternity object can be thought of as being a subset of such a grid. Therefore, given any object, if we can properly match it to a portion of a grid made of rectangular cells, we can use our understanding of the rectangular grid to understand the object. Moreover, if we simply have the boundary word for the object, and we can overlay that boundary on the rectangular grid, then we can instantly fill in the object with the locations of nodes and elements. The power of this idea will only gradually become clear.

To begin with, let us consider a single rectangular cell, as shown in Figure 8. We will describe this as having height $h = 2$ and width $w = 1$, where these measurements refer to the number of squares. In general, an $h \times w$ rectangular grid will be h squares high and w squares wide. To settle a parity choice, we will always assume that the square in the lower left corner of a rectangular grid is identical with the bottom square in the single rectangular cell.

The 24 elements of the cell can be numbered somewhat arbitrarily. The 19 nodes, however, will be more carefully numbered, starting at the lower left corner, proceeding upwards, and then shifting to the right one column at a time. This labeling process gives us a sort of map of the basic rectangular cell, formed from two squares of opposing parity. A rectangular grid is formed by repeating the squares in an alternating pattern, vertically and horizontally.

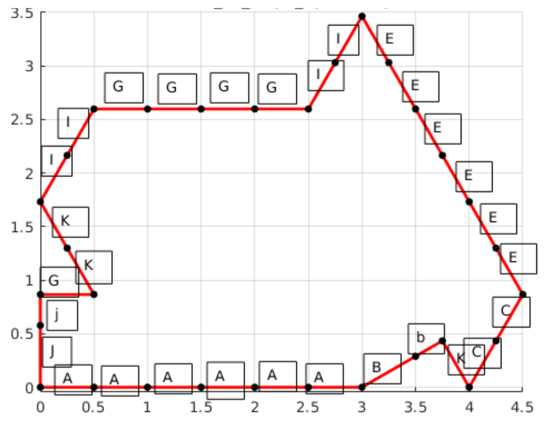


Figure 7: The Trinity Grid Constructed From Its Boundary Word, (i,j) coordinates.

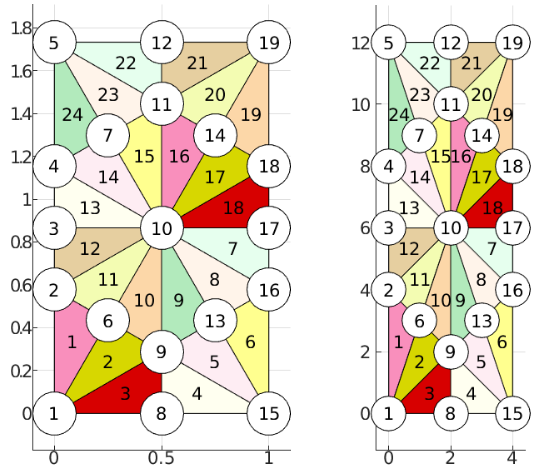


Figure 8: The 1 × 2 rectangular cell, using (x,y) and (i,j) coordinates.

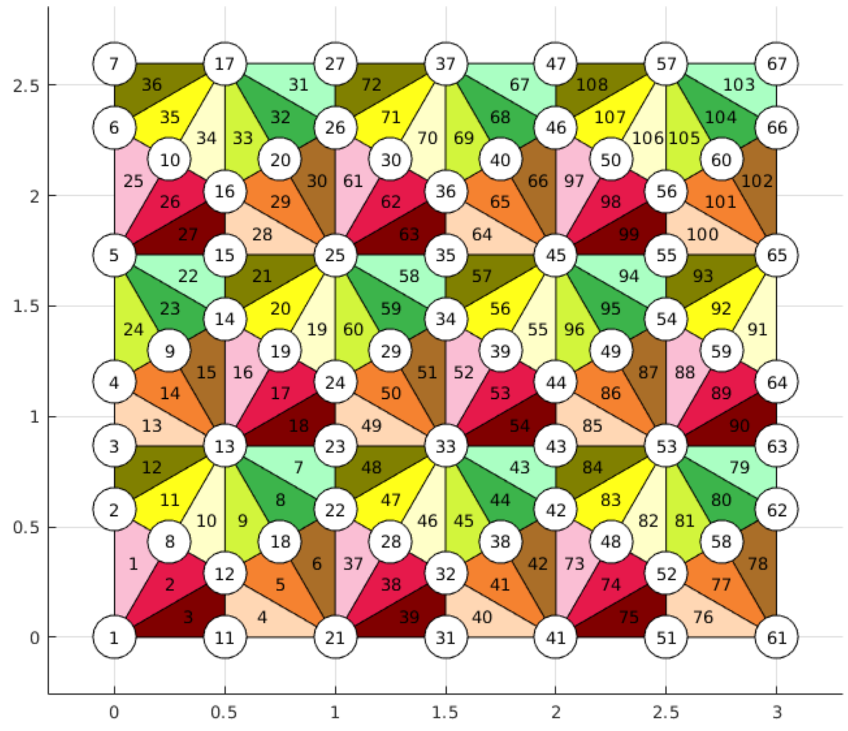


Figure 9: A 3x3 rectangle grid, (x,y) coordinates.

Suppose that we want to extend the grid by stacking squares vertically. We will always ensure that the stacking is done in such a way that the squares alternate in parity, forming a sequence of cells, with possibly one final unpaired square. Given this scheme, can we number the nodes and elements?

We can begin with the element numbering. If our stack involves a single square, or a pair of squares, then we can simply copy the element numbers from the map. If there are more squares, then process them in pairs, adding 24 to the element labels in the previous pair. Proceed in this way until all elements are numbered. If we have h squares in the vertical stack, we will have labeled all $12 * h$ elements.

Suppose that we have an $h \times w$ grid of squares. After we have numbered the elements in the first column of squares, we move to the neighboring column on the right. We repeat the numbering process, starting from the base value of $12 * h$. Continuing in this way, we will finally process the squares in column w, uniquely numbering all $12 * h * w$ elements.

A similar process can be applied to the nodes. However, for the nodes, we don't start at the first cell by simply accepting the numbers used in the map. Instead. we consecutively number all the nodes with the lowest x coordinate, then move to the next column of nodes, and so forth. Even viewing the single rectangular cell, it is clear that the number and position of the nodes in each node column varies. Moreover, when counting nodes, there is a sort of edge effect that has to be handled. For an $h \times w$ array of squares, we need to start with an initial "investment" of $2 * h + 2 * w + 1$ nodes along the extreme left and bottom boundaries. Thereafter, each additional square adds 6 nodes. Hence an $h \times w$ rectangular grid will contain $6 * h * w + 2 * h + 2 * w + 1$ nodes.

7 Filling in an Eternity Object from its Boundary Word

It will be vital to rely on the boundary word as our description of the Eternity objects we deal with. Even when we reflect, rotate, or translate them, we will want to do so by applying the appropriate operations to the boundary word. But the boundary word only gives us information to trace out the boundary of the region, and we will at times need to know the internal structure of the item as well, that is, the number, location, and adjacency of the nodes and elements. A reliable and efficient way to do this involves comparing the object to a template, which essentially lays a copy of the object on top of a rectangular grid whose structure is easy to understand.

There is some delicacy required in setting up this background template grid. The lower left corner of the rectangular grid is always a square of a particular parity. It may be necessary that the object to be matched be shifted to the right or upwards a few steps, in order to get the proper matching. This can be done automatically, if we are given, in addition to the boundar word, the location and orientation of the first node of the first triangle of the object. From this we can specify the necessary height, width, and base point to be used in constructing the rectangular grid.

Now our rectangular grid will have the property that the boundary of the Eternity object exactly lies along its grid lines. This guarantees in turn that the nodes and elements of the object are exactly a subset of those belonging to the grid. Now we can think of the Eternity object as "selecting" the nodes and elements of the rectangular grid that it surrounds. If we could visualize the grid and the object, we could simply read off the information we want. Since we can't see the data, we must determine an automatic procedure for identifying the selected nodes and elements, so that we can construct the lists necessary to reference them later.

Because the rectangular grid covers the object and properly aligns with it, every node in the object is also a node in the rectangular grid. We understand how to index and locate every rectangular grid node. Therefore, for ever node in the rectangular grid, we pose the question of whether it is contained in the object as well. We can answer this question using a well known algorithm known as *point-in-polygon* [5]. Because some nodes will be on the exact boundary of the object, it is advisable to use the (i,j) coordinate system to avoid roundoff issues. The result of this process is a list of object nodes.

We now perform a similar analysis to determine which elements (30-60-90 triangles) of the rectangular grid are included in the object. To determine if an element is inside the object, we compute its centroid and again call the point-in-polygon algorithm. When we add an item to the object triangle list, we automatically translate the labels of its nodes from the rectangle grid to the just-created object node list. The object elements are each given a new label in the order in which they are "discovered".

Thus, by using the rectangular grid as a template, we are able to reliably determine the number and arrangement of the nodes and elements in any Eternity object.

8 The (x, y) Coordinate System

We are working with 30-60-90 triangles, whose angles and opposing sides are correspondingly labeled (α, β, γ) and (A, B, C). Therefore, as physical objects, we know that, given the length of side C as $||C||$, we have

$$||A|| = \frac{1}{2}\,||C||$$
$$||B|| = \frac{\sqrt{3}}{2}\,||C||$$

If we insist that the equilateral triangles associated with the problem are to have sides of length 1, then we must take $||B|| = \frac{1}{2}$ and so we have

$$||A|| = \frac{\sqrt{3}}{6}$$
$$||B|| = \frac{1}{2}$$
$$||C|| = \frac{\sqrt{3}}{3}$$

Once we have chosen our measurement scale, we can assign physical (x, y) coordinates to the nodes in the grid, to the vertices of a tile, to the endpoints of an edge. We can construct proper plots of the grid or any tile to scale. When we use this measurement scheme, we say we are working with the (x, y) coordinate system.

9 The (i, j) Coordinate System

The (x, y) coordinate system is physically meaningful, but can be computationally awkward. For instance, given the (x,y) coordinates of a point, we might like to determine if it corresponds to a node, or lies on the edge between two nodes, or lies within the interior of a particular tile. These determinations require us to divide the vertical coordinate by an irrational number, and hence can result in very small, but nonzero, variations from the exact numeric result. Unfortunately, a very small error is enough to produce the wrong answer to our questions.

Looking at the basic rectangle, we can see that a natural horizontal scale might use $||B|| = \frac{1}{2}$ while a natural vertical measurement might be $||A|| = \frac{\sqrt{3}}{6} \approx 0.2887$. In fact, in order to capture all the detail of the grid, we will need to use horizontal and vertical scales that are half this size. Given the structure of the Eternity grid, or simply looking at the basic rectangle, we see that, starting from the origin, all nodes are spaced an integer multiple of $||B||/2$ horizontally, and $||A||/2$ vertically. (The converse is not true, however. Not all such multiples correspond to a legal node in the grid. This is one testament to the underlying complexity of the grid.)

Thus, we can use the (x, y) system implicitly, but convert to a simpler system, based on integer values, which we term the (i, j) coordinate system. In this system, a coordinate value of (a, b) means to go a horizonal steps left, and b vertical steps upwards, using the (i, j) stepsizes. The simplicity of the (i, j) system is illustrated by Table 2, which lists the (i, j) coordinates of the nodes in the 1×2 rectangular cell.

Because the vertical and horizontal scales differ in our (i, j) coordinate system, distances and angles are slightly skewed, but the general shape and structure of Eternity objects is reasonably preserved. Because the coordinates are integers, it is easy to detect groups of nodes that lie on a straight line, to identify a node in a plot given its coordinates, or conversely to determine the exact coordinates of a point from its plot. At any time, we can convert an (i, j) coordinate to its corresponding (x, y) value using the formula:

$$x = \frac{1}{4}\,i$$
$$y = \frac{\sqrt{3}}{12}\,j$$

10 Representing the Tiling Process

Imagine that we are about to carry out the successful tiling of a region. Our tiling will be a sequence of moves. In each move, we select a tile, and, knowing where it is to go, we then we may turn it, possibly flip it

Node	(i,j) Coordinates
1	(0, 0)
2	(0, 4)
3	(0, 6)
4	(0, 8)
5	(0,12)
6	(1, 3)
7	(1, 9)
8	(2, 0)
9	(2, 2)
10	(2, 6)
11	(2,10)
12	(2,12)
13	(3, 3)
14	(3, 9)
15	(4, 0)
16	(4, 4)
17	(4, 6)
18	(4, 8)
19	(4,12)

Table 2: (i, j) coordinates of nodes in the basis rectangle.

over, and then having got the orientation right, translate the tile to its position on the grid. For the Trinity problem, there are 4 tiles, and so there are exactly 4 correct moves to make. As it turns out, however, there are 142 possible moves, involving a tile, a rotation, a reflection, and a translation. The process of solving the puzzle involves representing every one of these moves, and then selecting the correct 4.

So far, we know how to describe a tile in a default position, using its boundary word W and base point P. Now, in order to generate all the possible tiling moves, we need to be able to represent the transformations involved in any tiling move. It turns out that a given move can be represented by starting with the W and P of the selected tile, and then, just as we rotate, reflect, and translate the tile, applying corresponding mathematical transformations to W and P.

By chance, the Eternity and Trinity tiles have no rotational or reflection symmetries. This means that when we start with a given tile, we have 6 possible rotations and 2 reflections that can be applied, meaning that every tile has exactly 12 distinct orientations. We distinguish orientations from configurations, which include the final step of choosing a translation, for which the number of choices varies.

We now consider the details of these transformations.

11 Rotations

When we are given a tile, we are free to rotate it before placing it on the grid. Of the 12 possible orientations of a 30-60-90 triangle, six of them are immediately achievable by rotating the tile through some multiple of 60 degrees. The other six involve both a rotation and a reflection. We consider applying rotations of 0, 60, 120, 180, 240 and 300 degrees to the data in W and P.

The rotation will be done by rotating about the point whose coordinates are P. Therefore, the base point P will not be changed by this transformation. The tile simply spins around that fixed point.

Notice that the boundary word uses an alphabet that specifies direction. Thus, the letter A indicates movement to the east, along the 0 degree direction. But suppose apply a 60 degree rotatation to a tile whose boundary word originally contained an A. The portion of the boundary corresponding to this a, which original headed east, should now move at a 60 degree angle, corresponding to the letter C in the boundary word compass. And in fact, just as A should be replaced by C, B and b should be replaced by D and e, C replaced by E, and all subsequent letters simply incremented by two positions, with wrap-around at the end. Similarly, a 120 degree rotation moves the boundary word letters by 4 positions, and so on.

As an example, Eternity tile #1, for which P=(0,0), has its boundary word transformed by rotation as follows:

rotation	boundary word
0^o	`AAAAAAEEFfIFfIII`
60^o	`CCCCCCGGHhKHhKKK`
120^o	`EEEEEEIIJjAJjAAA`
180^o	`GGGGGGKKLlCLlCCC`
240^o	`IIIIIIAABbEBbEEE`
300^o	`KKKKKKCCDdGDdGGG`

12 Reflections

In order to achieve all possible orientations of a tile, it may be necessary to follow the rotation by a reflection. We choose to reflect about the horizontal axis. This means that the type of each triangular element in the tile reverses its sign, $+1 \leftrightarrow -1$ and so on up to type 6.

Reflection across the horizonal axis can be applied to the boundary word by a simple alphabetical substitution, as occurred for rotations. The substitution table is as follows:

tile	boundary word alphabet
old	`ABbCDdEFfGHhIJjKLl`
new	`GfFEdDCbBAlLKjJIhH`

Thus, Eternity tile #1 might be rotatated by 120^o and then reflected, we would compute the following sequence of boundary words:

rotation	boundary word
original	`AAAAAAEEFfIFfIII`
rotated	`EEEEEEIIJjAJjAAA`
reflected	`CCCCCCKKjJGjJGGG`

13 Translations

Once we have chosen a tile, a rotation and a reflection, it remains to choose a translation. The type of element #1 in the tile changes during rotation and reflection, but the type is updated as part of these transformations. We also know the type of every element in the grid. A translation can only move the tile in such a way that element #1 is placed on top of a grid element of the same type. So we begin by listing all grid elements of the same type as element #1.

The 30 degree node of tile element #1 is the point P. For grid element of index K, whose type matches that of tile element #1, we can determine Q, the (i,j) coordinates of its 30 degree node. Now the proposed translation, which will lay the tile into the grid in such a way that tile element 1 covers grid element K, is simply P' = Q - P.

However, we now have to confirm that all the other elements of the tile also remain inside the grid. To do this for an element, we determine the (i,j) coordinates of its vertices, take the average of, the vertices, which corresponds to the centroid of the element, and then do a "point in polygon" query that reports whether the centroid is contained in the grid. If we get 36 positive answers, then the entire translated tile remains in the grid, and this rotated, reflected, and translated tile is a configuration that must be considered in the final tiling computation.

14 Generating All Tile Configurations

Our solution process begins by choosing one of the tiles (4 in Trinity, or 209 in Eternity), one of the six rotations, one of the two possible reflection options. It is now necessary to consider every possible translation as well. It might seem that, if the grid has 144 triangles, then we need to consider, for K from 1 to 144, the translation that shifts the tile so that triangle #1 lies on grid triangle #K. However, only a much smaller set of translations is possible because:

- the translation is only possible if triangle #1 of the tile is translated to a grid triangle with the same type. There are 12 possible types, and we have tabulated them for each grid triangle.
- the translated tile must lie entirely within the grid. This requirement is checked by computing the centroid of each triangle in the tile, and verifying that it is contained within the grid.

We refer to translations of a tile which satisfy these two conditions as a **conforming translation**.

By considering each tile, each rotation, each reflection and each conforming translation, we have indexed the complete set of tile configurations that could be involved in a tiling of our target region. We think of these as our **variables**. Our task will be to assign to these variables the values 0 or 1, so that we can specify a particular set of configurations that tile our region. We will do this by generating a linear system.

15 The Linear System for Tiling

To express the tiling problem as a linear system of equations, we need equations describing the interaction of variables. The tile configurations are our variables, each one represented by a list of the grid elements that it would cover. The equations then express the complementary condition, from the element's point of view, that some configuration (and only one!) must cover that element. Now we can start to see the structure of a matrix A, such that entry $A(i, j)$ is 1 if element i is covered by configuration j. The right hand side vector b will have the value $b(i) = 1$, indicating that the covering must occur, and must occur just once.

Our linear system is not complete; we need to state that each tile must be used exactly once. We know which tile was used to generate each configuration. Therefore, we must append an additional constraint equation corresponding to each tile. The constraint equation for the K-th tile must have a 1 in every column that corresponds to a configuration involving that tile, and a 1 on the right hand side, indicating that exactly one configuration of tile K was used in the solution.

We now have constructed a linear system of the form $A\,x = b$, that represents our tiling problem. Before we try to solve it, we must pose an additional condition: the only acceptable values in the solution vector x must be 0 and 1, with a 1 indicating that the corresponding configuration was actually used in the solution.

There is no reason to think that our system is square, and in general it will not be, Moreover, for cases like the Eternity puzzle, the thousands of equations will be vastly outnumbered by the millions of variables. This means we should be prepared to solve an underdetermined linear system. We cannot determine in advance whether the system has zero, one, or several solution. However, since the linear system precisely represents the tiling problem, we can assert that there are exactly as many solutions of the linear system as there are distinct solutions to the tiling problem.

Because of its size and its restriction to binary solution values, this linear system is not suitable for treatment by a standard linear algebra package like LAPACK. It can, however, be regarded as a special kind of linear programming problem, for which there are many robust, flexible, and powerful packages available, including CPLEX, GUROBI, and SCIP. Thus, after forming the linear system, we proceed with our solution strategy by appealing to an external linear programming solver that can handle large systems, and that can honor our requirement for a binary solution.

16 Interacting with Linear Programming Software

Most linear programming software can process a common file structure known as the LP format, which describes the linear system to be treated, an objective function, and some subsidiary constraints and other information such as a problem title.

Only the nonzero entries of the matrix are stored. In the Trinity matrix, here are 144+4 rows and 142 columns. However, each column contains only 36+1 nonzeros, out of the 148 entries. In the full Eternity matrix, again there will only be 37 nonzeros in each column, but now this represents a reduction from 2,508+209 entries in each column. The number of nonzeros in a row of the matrix is somewhat irregular, depending on the grid and the shape of the tiles. In some cases it could be as few as a single nonzero entry. For the Trinity puzzle, the first two equations look like this:

```
 x76 = 1
 x33 + x76 + x97 + x138 = 1
```

After the equations describing the covering requirements, there follow equations specifying the constraint that each tile must be used exactly once. These constraint equations are very similar in form to the covering equations. For the Trinity puzzle, the first such equation looks like

```
x1 + x2 + x3 + x4 + ... + x35 + x36 = 1
```

because the first 36 configurations involve tile #1.

In the LP format, variables whose values are restricted to 0 and 1 values can be labeled as "binary". For our tiling problems, all variables will have this designation. For the Trinity puzzle, this section of the file looks like this:

```
Binary
  x1 x2 x3 x4 x5 x6 x7 x8 ... x141 x142
```

Once the LP file has been created, it is necessary to execute one of the linear programming applications. For instance, if the LP file is named *trinity.lp*, then the CPLEX program could be invoked with the following commands, to find all solutions and write them to the file *trinity.sol*:

```
set mip pool absgap 0.0
set mip pool intensity 4
set mip limits populate 10
set mip pool capacity 10
set output writelevel 1
read trinity.lp
populate
write trinity.sol all
quit
```

Once the solution file has been created, a simple MATLAB postprocessing procedure can read the solution information as an array, which it can then analyze and plot.

17 Recovering the Tiling Configurations from the LP Solution

A solution vector returned by the linear programming software is simply a sequence of 1's and 0's. The 1's identify the tile configurations used in the given solution. Given only that component #K of the solution vector is a 1, how do we recover the corresponding configuration, that is, the tile that was used, its rotation, reflection and translation values?

When we constructed the linear system $Ax = b$, we selected a tile, transformed it, and if it remained within the region, we assigned it the index value #K, and then ... discarded the configuration information. This makes sense, because there is such a large number of configurations, only a few of which will actually end up being used in a solution. But once we know that configuration #K is used, we want to recover that original information.

The way we have chosen to deal with this need is, once we know the configurations that were selected, to simply go through the configuration generation process again. Each time that we generate a configuration, we momentarily have the associated tile index and transformations, which will be discarded when we move to the next configuration. But when we generate a configuration whose index #K corresponds to a solution component, we can generate the transformed tile, including the coordinates of its component triangles, and write this to a file, or add an image of this selected tile configuration to a plot of the solution.

18 Highlights of the Investigation

This study of the Eternity puzzle followed work on the tiling problem for polyominoes. The two tiling problems are analogous, and so we were able to employ much of the modeling and methods developed previously. However, as we worked on the Eternity puzzle, it became clear that its peculiar features required the discovery of new approaches and the construction of appropriate techniques to handle its complexity, including:

- The creation of the miniature "Trinity" tiling puzzle, to facilitate testing;
- The analysis of the grid and tiles into the nonisotropic 30-60-90 triangular elements;
- The construction of an underlying Cartesian grid;
- The use of an (i, j) integer coordinate to simplify work and guarantee exact results;
- The identification of a basic rectangle, of 19 nodes, that could be used as a background for any Eternity type grid;
- The use of the basic rectangle to allow for the consistent numbering of nodes and elements in any Eternity grid or tile;
- The extension of the boundary word idea from 4 letters to 12, including a varying step size;
- The discovery of how to apply rotations and reflections to a boundary word;
- The use of the "point in polygon" test to check whether a transformed tile remains in the region;
- The recovery of the details of a given configuration, given only its index from the LP solution to the tiling problem, by repeating the configuration construction process;

All of these concepts had to be thought through, then implemented in software, and tested. The goal of a tiling solution seemed to be just around the next corner, but making the turn, we found yet another corner. It was, therefore, an immensely satisfying moment to completely solve the Trinity problem. Although so small and unchallenging, we believe there the exact same procedures will solve the Eternity problem, barring issues of computer time and memory.

19 Conclusion

This investigation is inspired by the Eternity puzzle. However, it became clear that in order to test and debug the software being developed, a much smaller problem was necessary. This was the origin of the Trinity puzzle, which was an example used in a talk by Alex Selby. The grid involved 144 triangles, and the 4 tiles each had the usual 36 triangles. A solution to the puzzle was provided, but it was not stated whether there were more solutions.

The boundary words for the region and the four tiles were generated. The linear system was generated with 144+4 rows and 142 columns. CPLEX was used to analyze the system, and it returned a single solution vector x. This solution vector was then postprocessed to create a plot of the tiling, and a list of the transformations used on each of the four tiles.

The operations of generating the linear system (MATLAB), solving it (CPLEX), and postprocessing it (MATLAB) each took less than a minute.

Theoretically, the treatment of the Eternity puzzle is no different from that for Trinity. However, it should be clear that that problem is enormously more challenging. Generating the 209 boundary words for the tiles took a significant amount of patient sketching and programming. Just generating the matrix A is expected to be a substantial task. It will have 2,508+209 rows. The number of variables will be much larger, since each tile has 6*12 orientations, and it is likely that each orientation will be translatable to most of the approximately 2,508/12 elements of the same orientation, creating on the order of half a million configurations.

We cannot extrapolate a necessary processing time for the Eternity problem based on the trivial size of Trinity. If the current algorithms do not seem able to handle the problem, it may be necessary to search for other techniques to generate the configurations. Once we move to the LP solution stage, we anticipate running the problem through a version of CPLEX on a parallel computer system with extensive memory resources. If CPLEX cannot solve our problem, or cannot solve it in a reasonable time, then we may have to conclude that our approach, while mathematically correct, is currently computationally unfeasible.

This discussion has laid out the models for viewing the Eternity puzzle, and the techniques designed to find solutions to the tiling problem. The example case Trinity has been successfully handled in this way. The results for the Eternity problem will be discussed in a later presentation.

A Computer Software

During this initial work, a number of MATLAB files have been created. All these files are available in a set of related web directories which are publicly available. Each web directory has an index page which briefly describes its contents. In general, the directories come in pairs, with a source code directory and a test directory. The test directory is intended to demonstrate features of the source code.

The directories are located under `http://people.sc.fsu.edu/~jburkardt/m_src` and the directory page repeats the name of the directory with an html extension. Thus, the directory page for eternity is at `http://people.sc.fsu.edu/~jburkardt/m_src/eternity/eternity.html`

Name	Comments
eternity	General utilities for the Eternity and Trinity puzzles
eternity_test	tests
eternity_tiles	Defines the 209 Eternity tiles
eternity_tiles_test	tests
trinity	Trinity puzzle files
trinity_test	tests
trinity_solution	Postprocessing the Trinity solution from CPLEX
trinity_solution_test	tests

References

[1] Marcus Garvie, John Burkardt, “A new mathematical model for tiling finite regions of the plane with polyominoes”, *Contributions to Discrete Mathematics*, Volume 15, Number 2, July 2020.

[2] Solomon Golomb, *Polyominoes: Puzzles, Patterns, Problems, and Packings*, Princeton University Press, 1996.

[3] “Who wants to ruin a millionaire?”, The Guardian, 20 October 2000.

[4] Ed Pegg, “Polyform Patterns”, in *Tribute to a Mathemagician*, Barry Cipra, Erik Demaine, Martin Demaine, editors, pages 119-125, A K Peters, 2005.

[5] Moshe Shimrat, “Algorithm 112: Position of Point Relative to Polygon”, *Communications of the ACM*, Volume 5, Number 8, August 1962, page 434.

[6] Mark Wainwright, “Prize specimens”, *Plus Magazine*, 01 January 2001.