Graph Coloring with an Origami Context

Megan Ku
Data Structures and Algorithms

May 5, 2020

Introduction

Modular origami is fascinating; with it, we can create complex geometric solids out of simple interlocking paper units. Using multiple units opens up the possibility of creating visually pleasing models with color diversity. In this project, I sought to find a solution to the coloring problem that would use 3 colors, such that each color was distributed equally on a graph that represented my modular origami creations. I would be able to prove the correctness of my algorithms through the assembly of origami modular that I folded.

Background

Origami is the art of paper folding. Modular origami, in particular, relies on the interlocking of several individual pieces of origami, known as units, without using any adhesive. The inherent geometric properties in origami allow for the creation of complex models that can be abstracted as polyhedra. In particular, I will be using the Bascetta star unit [2] as my base unit. This unit is super easy to fold, locks exceptionally well, and can be used to create a wide range of polyhedra. Origami Bascetta stars are commonly assembled using three different colors of paper, arranging them so that no two units of the same color are adjacent. This sounds easier than it is, and the difficulty is that typically a folder is unsure if their current arrangement of the units will yield a feasible solution until they get to the very end and are only left with same-colored units. Finding a solution to this problem would provide folders with a visual roadmap and mitigate the frustration that comes with assembling modular origami in this color fashion.

1 Graph Representations of Modular Origami

To solve this coloring problem, we first need to represent the origami as 2D graphs. As mentioned before, we can represent modular origami as known polyhedra. The thirty-piece Bascetta star, for example, can be represented as an icosahedron, a polyhedron that has equilateral triangular faces.

We can also represent an icosahedron as a planar graph, which is a graph whose edges only intersect at the nodes of the graph. The faces of the graph are the regions enclosed by vertices and edges. Now we have a 2D representation.

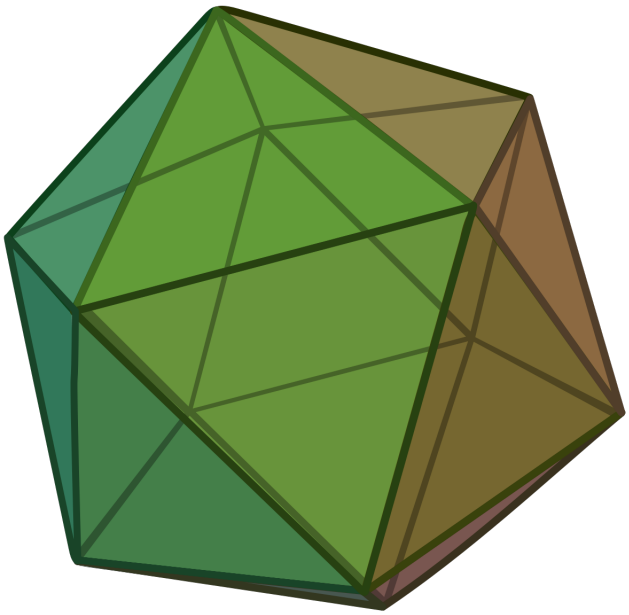

Figure 1: An icosahedron. Image from Wikipedia.

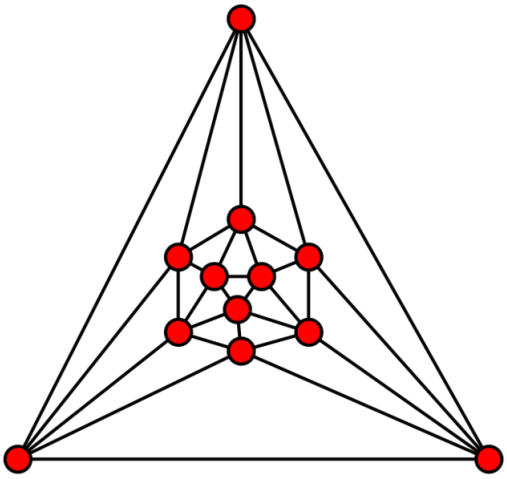

Figure 2: Planar graph of an icosahedron. Image from Wikimedia Commons.

The icosahedron doesn't reflect the entire geometry of a Bascetta star, though. The model is spiked, as in each triangular face has triangular faces on it. So we can create the 'spiked' icosahedron by dividing each face. It's important to note that in the planar graph, the white space around the graph counts as a face.

Now the graph accounts for all edges (convex and concave) and all nodes (peaks and sinks) of the modular origami. The thing is, for this problem, we want to color the faces of the model, not the vertices. So we can take the dual of the graph, which is where every node of the graph becomes a face and vice versa. [1]

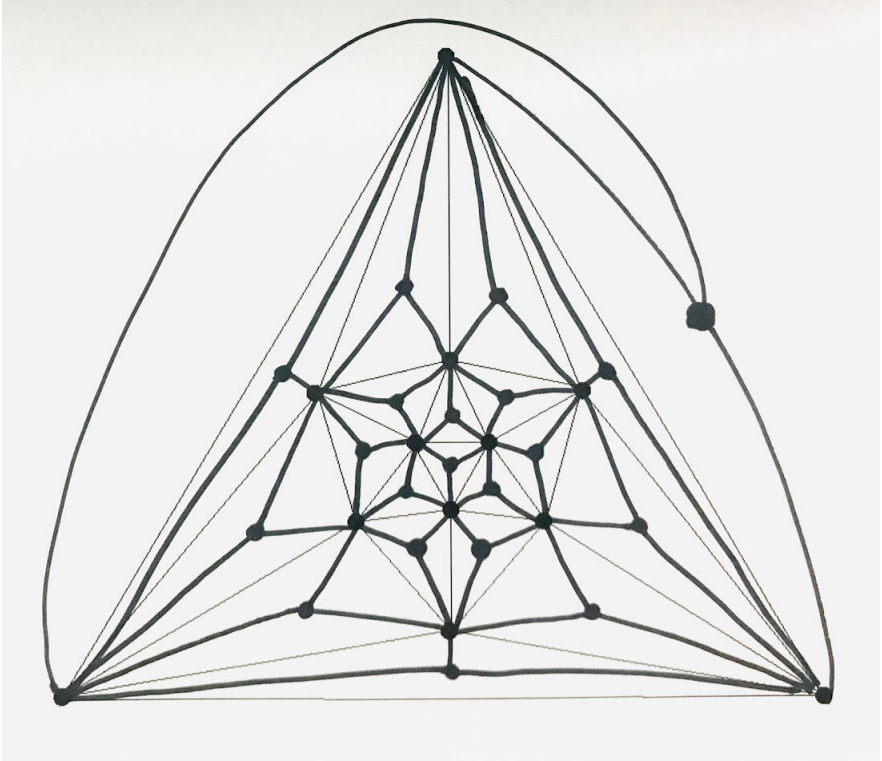

Figure 3: Planar graph of a spiked icosahedron.

The final step is a direct consequence of the unit we are using to construct our modular origami. A single Basecetta start unit comprises two triangles that make two different spikes on the model. So the final step is to reduce the number of edges by "merging" node-edge-node linkages into single nodes. This final graph is the graph that can be used in the coloring problem.

Algorithms and Implementation

I implemented both a greedy and local search algorithm to try to solve this NP-hard problem. This problem is quite similar to the original graph coloring problem, with the added constraint that the number of times each color is used should be the same.

Greedy Heuristic Algorithm

The greedy heuristic algorithm uses breadth-first search to traverse through the nodes. At each node, it colors the node that is currently the least used that isn't already used by its neighbors and then enqueues all unvisited neighbors to be subsequently colored. This algorithm is indeed greedy; it chooses the locally optimal color at each node, only looking at its immediate neighbors and the total color distribution. This greedy method usually ends up creating color maps that have conflicts (ie. a node is surrounded by all three colors). The

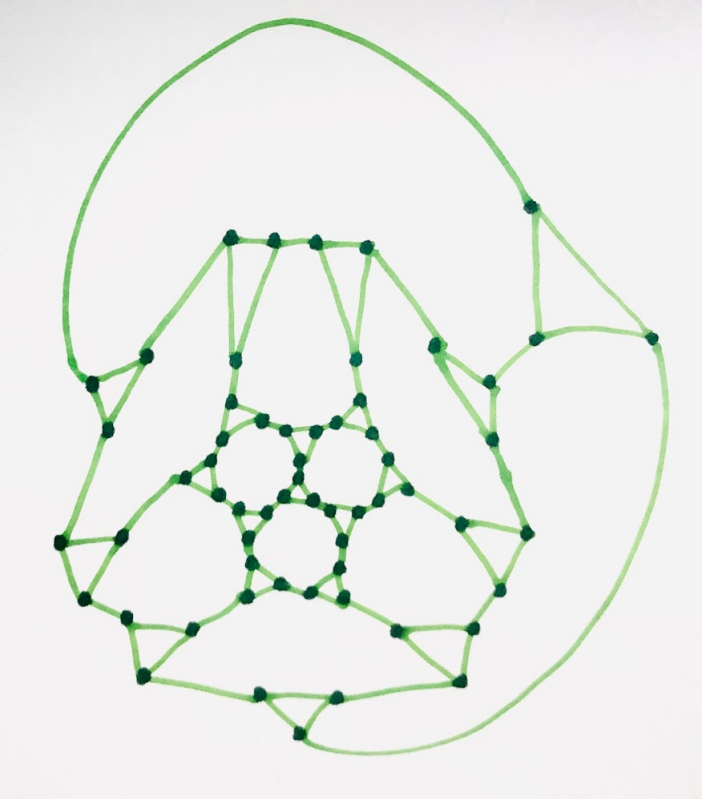

Figure 4: Planar graph of the dual of the spiked icosahedron.

algorithm accounts for this by defaulting to the first color and creating a list of nodes that are invalidly placed. A solution can be derived from this greedy implementation, but it is highly dependent on which node is the starting node.

Local Search Algorithm

The local search algorithm takes a solution provided by the greedy algorithm and improves it by attempting to reduce the number of "violation" nodes. For each node in the generated list, the color of that node is changed to one of the other two colors – whichever color is less present among its immediate neighbors. Once the color is changed, if there are nodes that are now violating the same color rule, those nodes are added to the violation list. This process repeats until an upper limit of iterations is reached or an optimal solution is found.

Overall Implementation

I combined the greedy heuristic and local search algorithms to potentially produce the most optimal result. Because I realized that a solution can be found with the greedy heuristic algorithm, I run the greedy algorithm several times to decrease the starting number of incorrect nodes. I then run the local search algorithm on the best solution found by the iterations of the greedy algorithm.

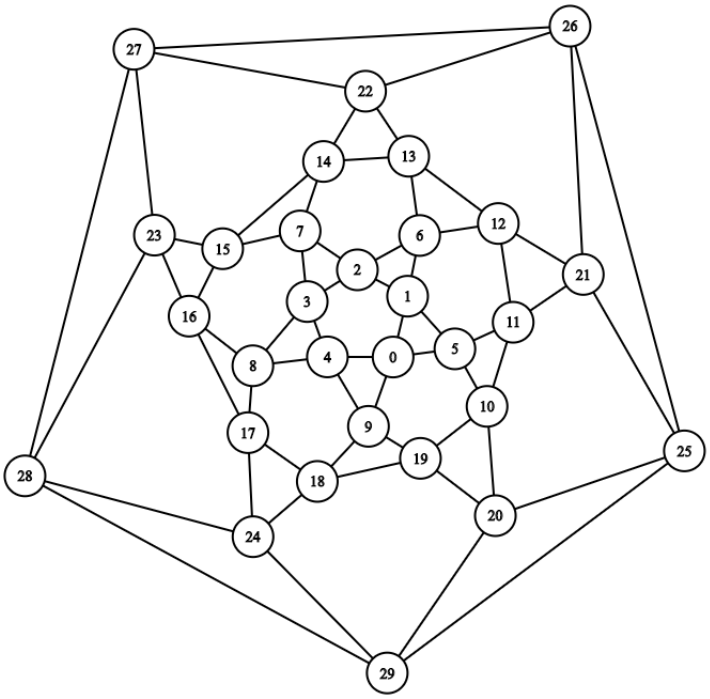

Figure 5: The final graph representation of the 30-piece Bascetta star.

2 Results and Analysis

	Runtime (s)	Optimality Gap
Greedy Heuristic	5.531e-5	0.100
Local Search	2.289e-4	0.075

The following runtimes are related to the 30-piece Bascetta Star. Overall, the local search algorithm did improve the greedy solution, and the runtime for the local search algorithm is, as expected, longer than the greedy heuristic algorithm due to the number of potential iterations the algorithm can run while trying to find the perfect solution.

There were several points where I was able to find an optimal solution just by using the greedy algorithm. Here is output from one run with an optimal solution.

```
0:‘blue’, 1:‘green’, 2:‘blue’, 3:‘yellow’, 4:‘green’, 5:‘yellow’,
6:‘yellow’, 7:‘green’, 8:‘blue’, 9:‘yellow’, 10:‘blue’, 11:‘green’,
12:‘blue’, 13:‘green’, 14:‘blue’, 15:‘yellow’,16:‘green’, 17:‘yellow’,
18:‘blue’, 19:‘green’, 20:‘yellow’, 21:‘yellow’, 22:‘yellow’, 23:‘blue’,
24:‘green’, 25:‘green’, 26:‘blue’, 27:‘green’, 28:‘yellow’, 29:‘blue’
```

I took the above solution and implemented it on both the planar graph and the origami modular to confirm that indeed, the solution is correct.

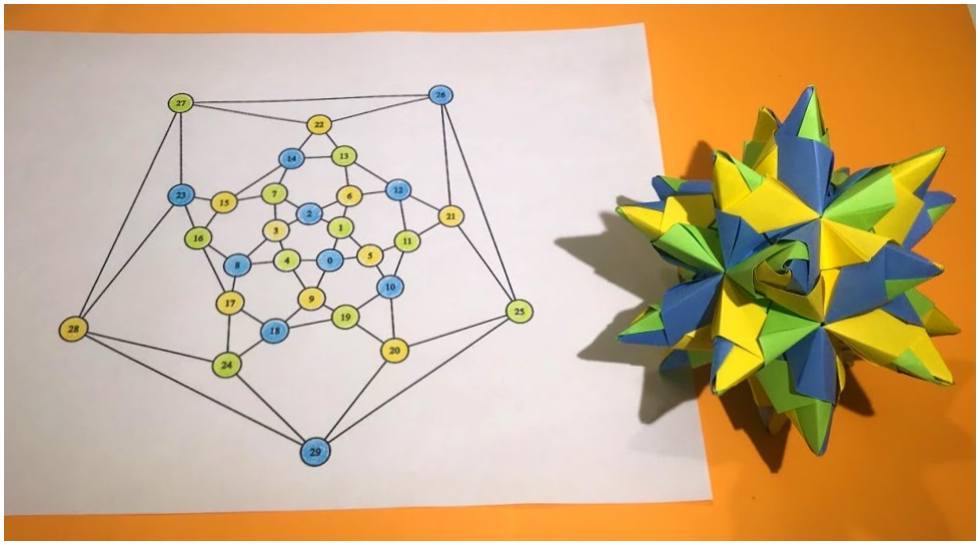

Figure 6: A valid solution to the coloring problem.

Reflection

One thing to note was that an ideal solution could be obtained just using the greedy algorithm at the right starting node, so there was a probability that the local search algorithm did not need to be used at all. Also, the local search algorithm needs improvement; more often than not, the search algorithm would fall into an infinite loop while trying to reduce the number of violations. This code will need to be improved upon to increase the chances of finding an ideal solution. Due to the amount of work it would take to create and implement a 120 piece Bascetta star, I decided to scale back and focus on the 30 piece module, something that I'm very familiar with. In the future, I plan to implement the larger graph and analyze its performance, as well as improve the optimality of my current algorithm. I've realized that origami almost begs to be put hand-in-hand with algorithms and optimization.

I found this project as a fun way to bring two realms of knowledge together. For a while, I had avoided the mathematics and theory around origami, afraid that it was too daunting for me. But this project seemed extremely approachable given my existing knowledge in both domains. Origami has been studied as a means to do something better than it has been done before, whether it be designing airbags or solar arrays. I wait for the day where origami can do something that could never have been done before.

3 Code

The code can be found on my GitHub page here.

References

[1] Thomas Hull. *Unit Origami as Graph Theory*. URL: http://origametry.net/papers/uogt.pdf.

[2] *Paolo Bascetta About Page*. URL: https://paolobascetta.format.com/contact.