4-5-2021

Computational Difficulty and Invariants of the Snake Cube Puzzle

Adrian Negrea
Andrews University, adriann@andrews.edu

COMPUTATIONAL DIFFICULTY AND INVARIANTS OF THE SNAKE CUBE PUZZLE

HONORS THESIS

Adrian Negrea
April 1, 2021

HONS 497

Adviser: Anthony Bosman, PhD

Signature: _____

Department of Mathematics

ABSTRACT

The snake cube is a popular puzzle that has been analyzed for its computational difficulty and shown to be NP-complete. Conceiving of the puzzle as a Hamiltonian path in an $n \times n \times n$ graph, we offer a novel mathematical analysis by considering invariants of the puzzle. This allows us to determine necessary conditions for a particular snake cube to be solvable and eliminate a large class of possible puzzles as unsolvable. In particular, we establish upper and lower bounds on the possible number of straight components in solvable snake cube puzzles.

Figure 1: The snake cube puzzle in its solved and original configurations. [1]

1 Introduction

Snake cube puzzles are physical puzzles comprised of wooden or plastic units attached with an elastic cord. In its most popular version, the snake consists of 27 cubes; the goal is to fold the cube into a $3 \times 3 \times 3$ cube (Figure 1). Variations exist, including a 64 cube snake that one folds into a $4 \times 4 \times 4$ cube. Note the configuration of the snake is carefully chosen so that a solution is possible, but it is easy to construct snake configurations that cannot be folded into a cube. This puzzle lends itself well to mathematical analysis using tools from modern graph theory. In particular, a Hamiltonian path is path that winds along the vertices of a graph via their connecting edges so that every vertex is visited exactly once; for example, see Figure 2. As such, a solved snake cube represents a Hamiltonian path in a $3 \times 3 \times 3$ cube graph (or, more generally, an $n \times n \times n$ cube graph).

A natural question arises: Given a snake, how can one determine if it is possible to fold it into a cube? Or, equivalently, does that snake respect a Hamiltonian graph in a cube. For example, when looking at the two snakes of Figure 2, it is not immediately apparent whether they are solvable or not. However, it turns out that only the snake on the right has a solution, while it is impossible to turn the left snake into a cube.

Figure 2: The snake on the left cannot be folded into a cube and thus has no solution. At the same time, the snake on the right can be folded into a cube, and thus it has a solution.

Abel et. al. [1] demonstrated that it is computationally hard (NP-complete) to decide whether a given snake configuration puzzle can be folded into an $n \times n \times n$ cube. This computational difficulty indicates the need for other approaches to determining the solvability of a snake puzzle. One understudied possibility is to identify snake puzzle invariants which are properties of the puzzle the remain the same under folding the snake into a cube. For a trivial example of such an invariant, note that regardless of how the puzzle is transformed, the number of cubes remains invariant, indicating that it is necessary for the snake puzzle to have n^3 cubes in order to fold into an $n \times n \times n$ cube. The goal of the research program is to identify more sophisticated invariants to help decide when a snake cube is solvable. The ultimate ambition

would be to develop a set of invariants that are both necessary and sufficient for solvability, completely characterizing the snake cubes that can be solved.

No such analysis has been attempted, though it is a natural next step to build on the existing analysis of the puzzle. Invariants are a well-used tool in graph theory, knot theory, and other areas of topology in settings where it is too computationally difficult to find solutions by brute force.

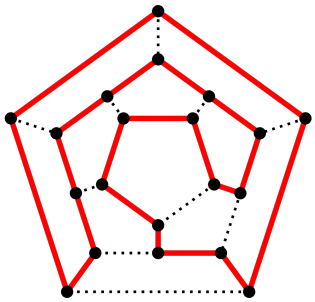

Figure 3: A Hamiltonian path: https://simple.wikipedia.org/wiki/Hamiltonian_path

2 Computational Difficulty

2.1 Turing machine

When solving a computational problem, we are often concerned with whether or not the problem can be computed efficiently with a Turning machine. The Turing machine is an abstract mathematical model typically imagined as a series of tapes subdivided into cells, with each cell holding one symbol from the machine's alphabet. Each tape has a head, a placeholder for the location on the tape that the Turing machine is looking at. One such tape must be the input tape, from which information can only be read, while information can be read or written to the rest of the tapes. This includes the output tape, to which the final answer is ultimately written, before the Turing machine halts. The remaining tapes are used by the machine for computation. Each Turing machine also has a certain finite alphabet, and all of the symbols on the tape must be within it. Along with a finite set of valid characters, a Turing machine also requires a finite set of possible states that it can be in. At every moment, a Turing machine must have a certain state, which in turn determines the next step of the computation. The reason Turing machines have proven to be so useful in breaking down problems has been their applicability to computers, since any character in any finite alphabet can be rewritten as a series of 1s and 0s. As a result, Turing machines can help identify those problems which can be computed by computers in a finite amount of steps.

2.2 P

Problems can be broken down into sets by how long their computation would take. A first such grouping is made up of those problems which require polynomial time to solve. The set of all such problems are then grouped together in a set called **P**. What this means in terms of a Turing machine is that, as the size of the input tape changes, the time to compute the problem is on the order of n^c, where $c > 1$ is a constant and n is the input tape. This may not seem particularly efficient, since c could be a large exponent; however, such problems prove to be much more efficient than their exponential counterparts, which can only be solved in exponential time or worse. A problem can be solved in exponential time if the time required is represented by c^n. Note that for sufficiently large n, c^n exceeds n^c. For input tapes of small size, there is often little difference between polynomial time and exponential time, but for large inputs (i.e. large n), non-polynomial time problems become computationally infeasible.

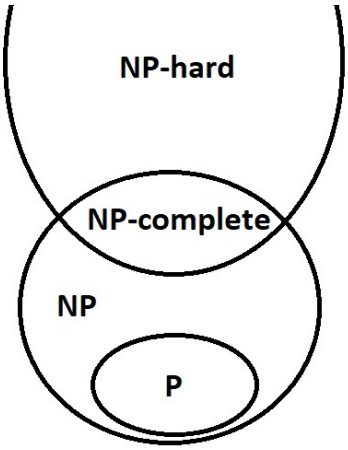

Figure 4: **P** is a subset of **NP**, while **NP**-complete is the intersection of **NP** and **NP**-hard.

2.3 NP and NP-hard

In addition to inquiring about the time needed to solve a problem, we let **NP** denote problems whose solutions may be verified in polynomial time. Of course, any problem of **P** is also a problem of **NP**, for if a solution can be found in polynomial, then we immediately have that it can be verified in polynomial time. A major open problem in computer science is to determine if **P=NP** or not. It certainly seems that there should be many problems that fall outside of **P** but that belong to **NP**, that is, a solution cannot be determined in polynomial time, but given a possible solution is proposed, it can be verified in polynomial time. This situation is analogous to solving a jigsaw puzzle. Given many pieces, it may take an incredibly long to determine the correct way to put them together, but given a possible solution, one can almost immediately tell if the assembled arrangement is correct or not.

There are several well-studied problems that are known to belong to **NP**. For example, the 3-partition problem states that a set of $3n$ positive integers can be partitioned into n triplets that all have the same sum. For example, for the set $\{20, 23, 25, 49, 45, 27, 40, 22, 19\}$, the numbers can be partitioned into 3 triplets which all add up to 90. They are $\{20, 25, 45\}$, $\{23, 27, 40\}$, and $\{49, 22, 19\}$. While it is trivial to verify the solution (just add up the numbers in each triplet), determining if such a partition is possible for any set of $3n$ positive integers is quite hard. Another famous **NP** problem is the traveling salesman problem that asks, given a set of points, what is the shortest path to visit every point and return at the starting point. A solution to this problem would allow for the most efficient paths to be found in a number of applications, such as finding the optimal delivery route.

Given a problem, such as a the traveling salesman problem, we can analyze its difficulty by reducing it to another well-studied problem, such as the 3-partition problem. That is, if we can show that any solution to the traveling salesman problem in fact gives a way of solving the 3-partition problem, then we can conclude that the traveling salesman problem must be at least as computationally difficult as the 3-partition problem.

If a problem in **NP** can be reduced to another, the second problem is then **NP**-hard and at least as hard as any other problem in **NP**. Hence, yet another set can be formed from the problems which are at least as hard as any other problem in **NP**. We call this set **NP**-hard. We define **NP**-complete to be the set of problems that are both **NP** and **NP**-hard. Both the 3-partition problem and the traveling salesman problem are known to be **NP**-complete.

The snake cube problem is **NP**; although it appears difficult to determine if a snake folds into a cube, given a possible solution, one can immediately verify if the folded puzzle is, in fact, in the shape of a cube. Abel et. al [1] demonstrated that the snake cube puzzle can be reduced to the 3-partition problem, thus showing the snake cube puzzle to be **NP**-hard. It therefore follows that the snake cube puzzle is **NP**-complete.

3 Invariants

3.1 Piece Types

Given a snake cube puzzle, we can classify each piece as being of one of three types:

- one of the two end pieces,
- a straight piece, that is, a piece where the elastic cord exists the face opposite to the face where it entered,
- a curved piece, that is, a piece where the elastic cord exits at a face adjacent to the face where it entered.

There are limits on how many of each type of piece can occur and how many can occur in a row. For instance, one cannot have a cube with no curved pieces, as such a cube cannot be folded. Moreover, in a $3 \times 3 \times 3$ cube, one cannot have more than 1 straight piece in a row, else one will have a row too long to fit inside the cube.

Already, this eliminates several possible puzzle configurations as having a possible solution. We are interested in determining stronger conditions on the possible total number and arrangement of straight pieces in a solvable snake cube puzzle.

3.2 Upper Bound

Proposition 3.1 *On a face of an $n \times n \times n$ cube, either each row has 2 non-straight pieces or each column has 2 non-straight pieces. Hence, there are at least $2n$ non-straight pieces on each face.*

Suppose some row does not have 2 non-straight pieces. We will argue then that every column has 2 non-straight pieces.

First consider the case that the row has no non-straight pieces. Then, in order for all of the pieces to be straight, they must all be vertical straights, since any string of horizontal straights must terminate in a non-straight by the time it reaches both the left and right edges of the cube. But since similarly each string of vertical straights must terminate in a non-straight before reaching the top or bottom edge, or before transitioning into a non-vertical straight (either horizontal or interior straights), we have that there must be 2 non-straights in every column of the face.

Now consider the case that the row has exactly one non-straight piece. Denote its position $p_{i,j}$ where $1 \leq i, j \leq n$ denote the row and column of the piece in the face, respectively. Note if $j = 1$ or $j = n$, then the non-straight piece is on an edge and since corners $p_{1,j}$ and $p_{n,j}$ cannot be straight, there are at least two non-straight pieces in column j. Moreover, since $p_{i,j}$ is the only non-straight in row i we have that all other pieces of row i must be vertical straights and thus, as in the case above, also have two non-straights in each columns.

Corollary 3.2 *Let $s_{max}(n)$ denote the maximum number of straight pieces in the $n \times n \times n$ snake cube. Then*

$$s_{max}(n) \leq n^3 - 8n + 8.$$

From Proposition 3.1 there at least $2n$ non-straight pieces per face on the six faces of a cube, giving $12n$ faces. However, the 8 corner pieces are non-straights that each are contained on three faces and hence were triple counted. Moreover, up to $n - 2$ non-corner edges pieces were double counted on up to two opposite edges per face. Since there is a maximum of 4 non-adjacent edges on a cube, this gives up to $4(n-2)$ pieces that were double counted. Therefore, the number of non-straights on the face of an $n \times n \times n$ snake cube is bounded below by

$$\underbrace{6(2n)}_{\text{total}} - \underbrace{8(2)}_{\text{corners}} - \underbrace{4(n-2)}_{\text{edges}} = 8n - 8.$$

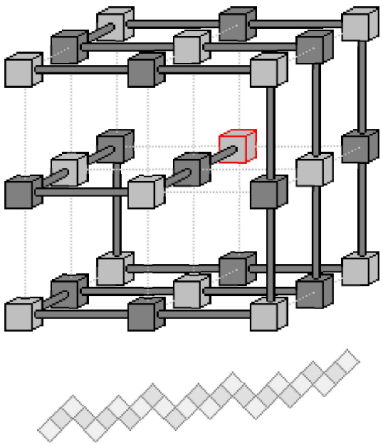

Figure 5: A solved snake puzzle with lots of straight pieces.

3.3 Lower Bound

We achieve a lower bound by exhibiting a solvable snake puzzle that has a large number of straights. For example, consider the puzzle of Figure 5. There are 3 horizontal straights on the top level, 2 horizontal and 3 vertical straights in the middle layer, and 3 horizontal straights on the bottom layer. Thus, this snake puzzle exhibits a total of 11 straight pieces and therefore $s_{max}(3) \geq 11$. However, we have already shown $s_{max}(3) \leq 11$ and hence we can conclude:

Proposition 3.3 $s_{max}(3) = 11$

Note Proposition 3.3 agrees with the brute force analysis of [3] which found that there are 11,487 solvable snake configurations when $n = 3$, two of which have 11 straight pieces and none of which have more than 11 straights.

We can generalize the pattern of Figure 5 to any $n \geq 1$. In particular, begin the Hamiltonian path at the top, left, front corner and draw a path along the front of the top face, down the right face, and along the bottom, then move over a row and reverse the pattern, and repeat. This will fill the top, right, and bottom faces. Then, move up (or down) a layer and fill the remainder of that middle layer with a zig-zag path. Continue doing this for all middle layers, filling the cube.

Let us count the number of straights in this generalized pattern. Observe that on the top layer, there are n rows that each have $n-2$ straight pieces (as neither end of the row is straight). The same is true for the bottom layer. There are $n-2$ middle layers and for each middle layer the rightmost column is comprises n vertical straight pieces; moreover, from the zig-zag pattern we also have $n-2$ horizontal straight pieces in each of the remaining $n-1$ columns. That is, the total number of straight pieces is given by:

$$\underbrace{n(n-2)}_{\text{top layer}} + \underbrace{(n-2)[(n-1)(n-2)+n]}_{\text{middle } (n-2) \text{ layers}} + \underbrace{n(n-2)}_{\text{bottom layer}} = n^3 - 2n^2 + 2n - 4$$

As in the $n = 3$ case above, this gives a lower bound to $s_{max}(n)$ for any $n \geq 1$.

4 Results

Putting together our lower and upper bounds, we have:

Theorem 4.1 *Given a snake puzzle that can be folded into an $n \times n \times n$ cube, the maximal number of straight pieces $s_{max}(n)$ is bounded above and below by:*

$$n^3 - 2n^2 + 2n - 4 \leq s_{max}(n) \leq n^3 - 8n + 8.$$

Calculating these bounds for several values of n gives the following table:

	$n^3 - 2n^2 + 2n - 4$	$\leq s_{max} \leq$	$n^3 - 8n + 8$	n^3
$n = 3$	11	$\leq s_{max} \leq$	11	27
$n = 4$	36	$\leq s_{max} \leq$	40	64
$n = 5$	81	$\leq s_{max} \leq$	93	125
$n = 6$	152	$\leq s_{max} \leq$	176	216

As we noted in Proposition 3.3, the lower and upper bounds agree for $n = 3$ letting us precisely determine s_{max}. In the other cases, we have a range of possible values for s_{max}. The size of the range is the size of the gap between the lower and upper bound which is the quadratic $2n^2 - 10n + 12$. Although this quadratic grows with n, it remains significantly smaller than n^3 for large n.

4.1 Computational Improvement

Let us consider the computational improvement achieved by our upper bound. In the case $n = 3$, the snake puzzle has 27 pieces, two of which are fixed as ends, but the other 25 can be either twisted or straight, so there are 2^{25} possible snake cube configurations. However, a sequence and its reverse produce the same snake, therefore we need to divide the number in half, giving $2^{24} = 16,777,216$ possible configurations.

However, we know that all snakes puzzles with 12 or more straight pieces are unsolvable. We can count the number of such puzzles by

$$\frac{1}{2}\left[\binom{25}{12} + \binom{25}{13} + \cdots + \binom{25}{25}\right] = 10,988,758.$$

Hence, we can immediately identify these 65.5% of puzzles as having no solution. While this fraction decreases rapidly for larger values of n, we still have that our upper bound rules out a large class of puzzles as unsolvable.

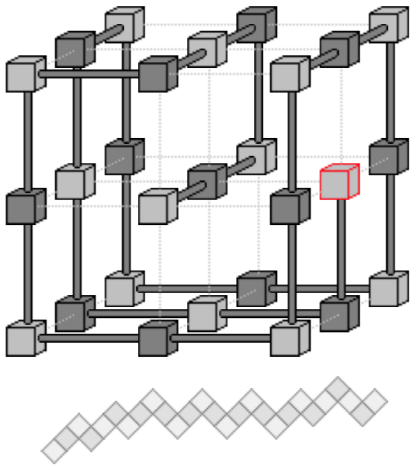

Figure 6: An alternative snake puzzle with maximum number of straights ($n = 3$).

4.2 Better Bounds?

How might our bounds be further improved? Recall the lower bound ($n^3 - 2n^2 + 2n - 4$) was established by giving an example of puzzle that has that has the maximal number of straights for a $3 \times 3 \times 3$ cube, then generalizing its pattern to arbitrary n. Our generalization was natural, but one might attempt an alternative generalization to obtain a different lower bound. Alternatively, one could begin with the other $3 \times 3 \times 3$ snake cube puzzle with 11 straight pieces in Figure 6 and attempt to generalize it.

Establishing the upper bound was more challenging. We achieved it by carefully partitioning the cube into a number of regions and then analyzing the necessary behavior of the the snake in and between each region. One could attempt alternative partitions of the cube, but none we attempted gave a tighter bound.

4.3 Future Work

It is also natural to consider bounds on $s_{min}(n)$, defined to be the minimal number of straight components in a solvable puzzle of length n^3. For instance, we might wonder if $s_{min}(n) = 0$, that is, if there is a snake cube puzzle with *no straight components* that folds into a cube. This is exactly the problem considered for odd n in [2]; one might generalize their work to achieve bounds on $s_{min}(n)$ more generally.

References

[1] Z. Abel, et. al. Finding a Hamiltonian Path in a Cube with Specified Turns is Hard In *Journal of Information Processing*, pages 368–377, July 2013

[2] A. Itai, C. H. Papadimitriou, and J. l. Szwarcfiter Hamilton Paths in Grid Graphs In *Siam J. Comput.* Vol. 11, No. 4, November 1982

[3] J. Scherpuis, Snake Cubes, In *Jaap's Puzzle Page*. https://www.jaapsch.net/puzzles/snakecube.htm Date Accessed: August 2020

[4] C. Zamfirescu and T. Zamfiresc Hamiltonian Properties of Grid Graphs In *SIAM Journal on Discrete Mathematics* Vol. 5, pages 564–570, 1992