The Collatz Graph as Flow-Diagram, The Copenhagen Graph and the different Algorithms for generating The Collatz Odd Series

Rolf Bruun, Sourangshu Ghosh
Indian Institute of Technology Kharagpur, September 2021

Abstract
By defining the Adjacency-Matrix for the Collatz Graph it is shown that it is possible, from the Adjacency-Matrix, to generate a “picture” which defines the Flow-Diagram for the Collatz Graph. In a sense the Flow-Diagram is the Collatz Graph.

While defining a “Double 2D Matrix” containing all odd numbers it is shown, that the 3N+1-Problem is in fact a problem concerning Groups of values not only individual Values. This implicates, that it is possible to predict the number of Operations needed to reach the Origo/Root 1 for a Group of values, if the number of Operations is known for a single Value in the Group.

After the formal analysis of the “Double 2D Matrix” a group of five Algorithms is investigated.

Based on the fact that two different Algorithms (with different rules) can generate identical Odd-Series and the fact that two of the Algorithms are able to generate identical “Branching-codes” in both directions UP and DOWN in a (binary) Tree, it is concluded that Collatz Conjecture is True.

1 Introduction
The present work is based on previous work by one of the authors (RB). Most of the previous work is included, but the present work contains more analyses of the key-concepts. 3D and 4D versions of Matrix Alpha are introduced as is The Copenhagen Tree (TCT) together with a group of five Algorithms. Two of the Algorithms are able to generate identical Odd-Series based on two different sets of rules. Two other Algorithms are able to generate identical “Branching-codes” where for one Algorithm it is in direction UP and the other in the direction DOWN in TCT.

When considering The Collatz Graph (TCG), where each positive integer is a Node and Links are established according to The Rule in Collatz Conjecture, it is a problem that TCG is infinite, as almost all “laws” for graphs are valid only for finite graphs, but it is actually possible to construct a well-defined Adjacency-Matrix for the Collatz Graph. This follows from the rule in the Conjecture.

1.1 The Rule in Collatz Conjecture (RCC)
Choose any positive integer $N_0$

If $N_n \equiv 1 \pmod{2}$ then $N_{n+1} = 3N_n + 1$ Type Odd Operation (TOD)

If $N_n \equiv 0 \pmod{2}$ then $N_{n+1} = N_n/2$ Type Even Operation (TEO)

Repeat the rule with new $N_{n+1}$

The Conjecture states that at some point after a final amount of repetitions, $n_{end}$, then $N = 1$.

The Complete Collatz Series (CCS) is generated when listing the values after each repetition.

$N_0 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4 \rightarrow N_5 \rightarrow N_6 \rightarrow N_7 \rightarrow N_8 \rightarrow \ldots \rightarrow N_n \rightarrow \ldots \rightarrow 1_{n_{end}}$

$n$ is the Total Number of Operations (TNT) in the Complete Collatz Series.

The Collatz Odd Series (COS) is seen when listing the odd values in CCS.

$N^0 \rightarrow N^1 \rightarrow N^2 \rightarrow N^3 \rightarrow N^4 \rightarrow N^5 \rightarrow N^6 \rightarrow N^7 \rightarrow N^8 \rightarrow \ldots \rightarrow N^p \rightarrow \ldots \rightarrow 1_{p_{end}}$

$p$ is the Number of Odd Operations (NOD) in the Complete Collatz Series.

Remark 1. As $N_{n+1} \equiv 1 \pmod{3}$ for TOD only $N^0$ can possibly have 3 as a divisor.

1.1.1 Conclusion 1
All $N^0 \rightarrow N^1 \rightarrow N^2 \rightarrow N^3 \rightarrow N^4 \rightarrow N^5 \rightarrow N^6 \rightarrow \ldots$ are $N \equiv 1 \pmod{6}$ or $N \equiv 5 \pmod{6}$ for $n > 0$. 

1
2 The Collatz Graph Flow-Diagram (CFD)

2.1 The Adjacency Matrix for The Collatz Graph (TCC)

RCC implicates, that all odd \( N = (2M - 1) \) are adjacent to \( 3(2M - 1) + 1 = 6M - 2 = 2^1(3M - 1) \) and that all even \( N = 2^m + 1 \) \( (2M - 1) \), \( m \geq 0 \) are adjacent to \( 2^m(2M - 1) \).

Detail: All \( 2^1(3M - 1) \) are adjacent to \( 2^0(3M - 1) \)

From this follows, that for all \( M > 0 \) it is true that:

\[ \ldots \xrightarrow{adj} 2^1(2M - 1) \xrightarrow{adj} 2^0(2M - 1) \xrightarrow{adj} 2^1(3M - 1) \xrightarrow{adj} 2^0(3M - 1) \xrightarrow{adj} \ldots \]

The trick is, that for all \( M \) it is possible to skip the \( \xrightarrow{adj} 2^0(2M - 1) \xrightarrow{adj} 2^1(3M - 1) \xrightarrow{adj} \) step (in a Flow-sense) which leaves the “adjacency-link” \( \ldots \xrightarrow{adj} 2^1(2M - 1) \xrightarrow{Link} 2^0(3M - 1) \xrightarrow{adj} \ldots \) and it is this “Flow-rule” that is used to generate the following “picture” from the Adjacency-Matrix.

Please notice, that \( 2^1(2M - 1) > 2^0(3M - 1) > 2^0(2M - 1) \) for all \( M > 1 \) which is important (see 3.10).

2.2 Generating the Flow-Diagram

Appendix A contains a step by step procedure for generating The Pattern from the Adjacency-Matrix.
Please notice, that the procedure is “non-destructive”, in the sense that no new rules are imposed on the Adjacency-Matrix i.e. The Pattern is “an image” or “a picture” of existing dynamics.

2.2.1 Navigating in The Pattern
You enter The Pattern in the Diagonal-line (marked with 8-symbols).

If you enter at an even value (that does not have 3 as a factor) you follow The Pattern vertically up until you reach the Double-line (marked with 2-symbols) and follow The Pattern from there.

If the even value has 3 as a factor, you follow the grey lines below the Diagonal left/up until an odd row.

If you enter at an odd value you go horizontally to the (3N + 1)-line (marked with 1-symbols) and then you go vertically down to the Double-line and follow The Pattern from there.

Once inside The Pattern you move to different positions on the Double-line depending on if the row is even or odd. If you are at a “2” in an even row you follow The Pattern horizontally left to the “8” and up to the next “2”. If you are at a “2” in an odd row you follow The Pattern horizontally right to the “1” and down to the next “2”.

The Pattern has direction and this direction is (everywhere) towards The Origo/Root 1 (which is to be proven).

Furthermore The Pattern actually has a metric in the form of the number of coloured squares/positions travelled from entering The Pattern to the current position.

We will get back to The Pattern in “Discussion of The Collatz Graph Flow-Diagram and the Matrices” (3.9).

3 Defining the “Double 2D Matrix” and the implications
3.1 Matrix Alpha

Figure 3: “The Double 2D Matrix” a.k.a. Matrix Alpha containing α-values

Values on the Left side turns to (6X-1) and on the Right side Values turns to (6X-5) i.e. next value in the COS.

Please notice, that each value in the two help-columns (6X-1) and (6X-5) is also an α-value L(X,Yc) or R(X,Yc).

3.1.1 Definition 1. Definition of Matrix Alpha
Matrix Alpha is defined as:

Put all the N ≡ 3 (mod 4) values in ascending order in same Column (but different Rows) in a “Left-Table”, then the values in the Column are defined by the Row-value X as L(X,1) := (4X-1)

Similar; put all the N ≡ 1 (mod 8) values in ascending order in same Column (but different Rows) in a “Right-Table”, then the values in the Column are defined by the Row-value X as R(X,1) := (8X-7)

All other values L(X,YL+1) and R(X,YR+1) are defined as 4*L/R(X,Y) + 1

Matrix Alpha is conveniently displaying both L(X,YL) and R(X,YR) as a “Double-2D-Table”.

4
3.2 Construction of Matrix Alpha

The idea behind Matrix Alpha is based on the fact that for all odd \( N \) the value \((4N+1)\) transforms to the same value as \( N \) when using RCC, e.g.

\[
\begin{align*}
3\times(4\times 3 + 1) + 1 &= 3 \times 10 = 30 = 2(15) \\
4\times(3) + 1 &= 13, 3\times(13) + 1 &= 40 = 8(5), 4\times(13) + 1 &= 53, 3\times(53) + 1 &= 160 = 32(5), 4\times(53) + 1 &= 213, 3\times(213) + 1 &= 640 = 128(5), 4\times(213) + 1 &= 853, 3\times(853) + 1 &= 2560 = 512(5)
\end{align*}
\]

TOD for odd \( N \) : \((2^M - 1)\) → \(3(2^M - 1) + 1 = (6^M - 2) = 2(3M - 1)\)

TOD for \((4N+1)\) : \(4(2^M - 1) + 1\) → \(3(4(2^M - 1) + 1) + 1 = (2^4M - 8) = 8(3^M - 1)\)

For all \( M \in \mathbb{N} \) it is true that

\[
\begin{align*}
(4N+1) \text{ turns to the same value as } (\text{odd}) \ N \ \text{so the inverse is also true; all } (N - 1)/4 \text{ becomes the same value as } N \ \text{to a certain limit. The value } (N - 1)/4 \text{ has to stay odd and it is observed that:} \\
N = (4V - 1) \Rightarrow (N - 1)/4 = (V - \frac{1}{2}) \text{ and} \\
N = (8V - 7) \Rightarrow (N - 1)/4 = 2(V - 1),
\end{align*}
\]

indicating that \(\{3, 7, 11, 15, 19, \ldots\} \equiv 3 \pmod{4}\) and \(\{1, 9, 17, 25, 33, \ldots\} \equiv 1 \pmod{8}\) are some type of Start-values for the \((4N + 1)\)-values following from \(N\) in these two non-overlapping infinite Groups.

As \( N \equiv 3 \pmod{4} \) covers all the \( N \equiv 3 \pmod{8} \) and \( N \equiv 7 \pmod{8} \), and we already have the Group \( N \equiv 1 \pmod{8} \) covered, all remaining values must be \( N \equiv 5 \pmod{8} \) i.e. all the remaining values are of type \((8V - 3)\).

So it is in fact possible to split the odd numbers in two separate (non-overlapping) Groups:

\[
\begin{align*}
\ldots(4i(4i(2i(2iM - 1i) + 1i) + 1i) + 1i) + 1i) \xrightarrow{\text{Link}} (6X - Q), X \in \mathbb{N}, Q \in \{1, 5\}
\end{align*}
\]

This gives us the opportunity to arrange all odd numbers in the “Double 2D Matrix” called Matrix Alpha.

The mere fact that it is possible to generate Matrix Alpha from the above observation makes it interesting.

It obviously covers all odd numbers i.e. \((4X-1), (8X-7)\) and all the \((8X-3) = (4V+1)\) for \(V\) odd.

All \((6X-1)\) and \((6X-5)\) are odd hence of type \((4X-1), (8X-7)\) or \((8X-3)\) and as all odd values \((2M-1)\) has a unique “position” on the left side of the Links it seems intuitively, that there is a fixed direction.

It is “a collection of one-way streets all leading to Rome.” All \((2M-1)\) must reach The Origo 1.

3.2.1 Conclusion 2

All \((4N+1)\) becomes the same value as odd \( N \) when using the RCC.

We observe that the Groups of odd values corresponds to Rows \(L(X,Y_L)\) or \(R(X,Y_R)\) in Matrix Alpha for \(X_n\) constant. This is no coincidence as this is how Matrix Alpha is defined.

Remark 2. All \(\alpha\)-values in the same Row becomes the same value (next in the COS) when using the RCC.

3.2.2 Conclusion 3

The individual Rows in Matrix Alpha are made from the (infinitely large) Groups/Sets of \(\alpha\)-values that share identical COS from \(N_1\) (i.e. the next \((6-Q)\)-value in the COS) and DOWN in the COS towards the Root 1.

Remark 3. The authors believe that Conclusion 3 in itself (with careful analyses) would be enough to prove The Collatz Conjecture, as any other Loop than the known \(1 \rightarrow 4 \rightarrow 2 \rightarrow 1\) can be shown impossible from this fact.

3.2.3 Conclusion 4

For Matrix Alpha all \((3^\ast L(X_n,Y_c) + 1) + 1\) are two TEO from \((3^\ast L(X_n,Y_c) + 1)\) and similar for \(R(X_n,Y_c).\)
3.3 Some more details about Matrix Alpha

It should be noticed that an alternative definition for the functions \( L(X,Y) \) and \( R(X,Y) \) is:

\[
L(X,Y_L) = \frac{(6X - 1)2^{2Y_L} - 1}{3} \tag{1}
\]

\[
R(X,Y_R) = \frac{(6X - 5)2^{2Y_R} - 1}{3} \tag{2}
\]

This alternative definition gives the \( \alpha \)-values directly from the coordinates \((X,Y)_{L/R}\).

This explains why all \( L(X,Y_L) \) becomes \((6X-1)\) and all \( R(X,Y_R) \) becomes \((6X-5)\) using RCC as:

\[
(6X - 1) = 3 \times L(X,Y_L) + 1 \tag{3}
\]

\[
(6X - 5) = 3 \times R(X,Y_R) + 1 \tag{4}
\]

**Remark 4.** This proves that all \( L(X,Y_L) \) are one TOD and \((2Y_L+1)\) TEO from \((6X-1)\) while all \( R(X,Y_R) \) are one TOD and \((2Y_R)\) TEO from \((6X-5)\) (the next value) in the COS.

3.3.1 Conclusion 5

All \( L(X,Y_L) \) are TNT = \( 2Y_L \) Operations from the \((6X-1)\)-value for the Row (next value in the COS) and all \( R(X,Y_R) \) are TNT = \( 2Y_R + 1 \) Operations from the \((6X-5)\)-value for the Row.

A Rooted Tree is defined later so the term *Branch* in the following is defined as:

“A specific Row \( L(X_c,Y) \) or \( R(X_c,Y) \) in Matrix Alpha - values arranged in ascending order.”

3.4 The 3D version of Matrix Alpha a.k.a. The 3D-Graph

In the following it is shown that *The Collatz Graph* exists as an “embedding” in the 3D version of Matrix Alpha.

The model is a “Pearls on Strings” type, where each positive integer is a *Node/Pearl* and *Links* between Nodes are the Strings. All the Nodes are associated with a unique *Position* in a 3D-grid-lattice.

The Links are established according to the Rule in Collatz Conjecture:

All Odd Nodes \((2M - 1) \equiv 1 \pmod{2}\) are Linked to the (Even) Nodes \((6M - 2) \equiv 4 \pmod{6}\) (and vice versa)

All Even Nodes \(2N \equiv 0 \pmod{2}\) are Linked to the Node with half the value \(N\) (and vice versa)

With the above definition the *Direction* in TCG is DOWN, but the important point here is that the Nodes are *Adjacent* in TCG. The “Pairs” of Nodes are *Linked* and all integers are included in the above definition of *Links* in TCG.

The Nodes are associated with Lattice-points in a normal \((x,y,z)\) coordinate system.

To be able to include all integers in the same coordinate system we place the *Left* part of the 3D-Matrix in one octant and the *Right* part in an other octant. As the Right part contains The Origo 1 = \(R(1,1)\) we choose that the octant with signs (+, +, +) is used for the Right part. We choose that the octant with signs (+, +, -) is used for the Left part.

The *Value* for the Nodes in the 3D-Matrix is *Defined* as:

\[
L(x,y,z) := \frac{(6x - 1)2^{2y} - 1}{3} = L(X,Y)2^{z} \text{ for } x > 0, y < 0, z \geq 0 \tag{5}
\]

\[
R(x,y,z) := \frac{(6x - 5)2^{2y} - 1}{3} = R(X,Y)2^{z} \text{ for } x > 0, y > 0, z \geq 0 \tag{6}
\]
In each half Left/Right in the 3D-Matrix we find Values (Nodes), Rows, Columns, Pillars, Layers and Slices:

A Value in the 3D-Matrix is defined by \( L_3(x,y,z) \) or \( R_3(x,y,z) \) for \( \{x, y, z\} \) constant

A Row in the 3D-Matrix is defined by the values for growing \( y \) and \( \{x, z\} \) constant

A Columns in the 3D-Matrix is defined by the values for growing \( x \) and \( \{y, z\} \) constant

A Pillar in the 3D-Matrix is defined by the values for growing \( z \) and \( \{x, y\} \) constant

A Layer in the 3D-Matrix is defined by the values for \( z \) constant and \( \{x, y\} \) growing

A Slice in the 3D-Matrix is defined by the values for \( x \) constant and \( \{y, z\} \) growing

Please notice that we find Odd Values in the Layer for \( z = 0 \). The Even Values are in the Layers for \( z > 0 \).

Please notice that “Pillars” are the collections of values \((2M - 1)2^z\) for \( M \) constant and \( z \) growing from zero.

Please notice that a TEO is \((2M - 1)2^z \rightarrow (2M - 1)2^{z-1}\), for \( z > 0 \)

Please notice that a TOD is \((2M - 1)2^z \rightarrow (6M - 2)\), for \( z = 0 \)

Important point: Please notice that all the Odd Values in a unique Row for \( z = 0 \) are Linked to the unique Pillar, where \((2M - 1) = (6x-Q), Q \in \{1,5\}\). The \((6x-Q)\)-Values are found in the help-columns in Matrix Alpha.

**Remark 6:** The individual Rows for \( z = 0 \) contains all the “Saplings” growing from a specific Pillar.

We establish all the Type TEO Links/Strings (EST) by connecting all nodes in individual Pillars. Now we have a complete collection of Pillars where all \((2M - 1)2^z \rightarrow (2M - 1)2^{z-2}\) and no Pillar is yet connected to another Pillar.

We establish all the Type TOD Links/Strings (OST) by connecting all \((2M - 1)\) to \((6M - 2)\). Now all possible Links in TCG are established and all Pillars, \((2M - 1)2^z\), for \( M > 1 \), are connected to other Pillars.

Now we have TCG embedded in the 3D-Graph.

Please notice that it is possible to find the Distance between adjacent Nodes in the 3D-Graph.

The Lenght of the EST and OST are found from the coordinates using the known Distance-formula:

\[
\text{Distance (Adjacent Nodes)} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2},
\]

(7)

For all Type TEO Strings this give the distance 1 between \((x, y, z + 1)\) and \((x, y, z)\) while for Type TOD Strings we use \((x_{Odd}, y_{Odd}, z_{Odd})\) and \((x_{Even}, y_{Even}, z_{Even})\) in the Distance-formula (where we know that \(z_{Odd} = 0\)).

Please notice that Pillars \((2M - 1)2^z\) where \((2M - 1) \equiv 3 \pmod{6}\) do not contain any even Nodes with Value \( \equiv 4 \pmod{6}\) as all the even values in these Pillars have values \( \equiv 0 \pmod{6}\).

Please notice that Pillars \((2M - 1)2^z\) where \((2M - 1) \equiv 5 \pmod{6}\) do contain even Nodes with Value \( \equiv 4 \pmod{6}\). Every second Even Node is of Type \( \equiv 4 \pmod{6}\), starting with \( z = 1 \), i.e. in the odd Layers, and the rest of the Even Nodes are Type \( \equiv 2 \pmod{6}\).

\[
(6x_C - 1) = \frac{3 \times L_3(x_C-w,z)}{2^{(w-z)}} + 1 \quad \text{for} \quad x > 0, y < 0, z \geq 0
\]

(8)

Please notice that Pillars \((2M - 1)2^z\) where \((2M - 1) \equiv 1 \pmod{6}\) do contain even Nodes with Value \( \equiv 4 \pmod{6}\). Every second Even Node is of Type \( \equiv 4 \pmod{6}\), starting with \( z = 2 \), i.e. in the even Layers, and the rest of the Even Nodes are Type \( \equiv 2 \pmod{6}\).

\[
(6x_C - 5) = \frac{3 \times R_3(x_C-w,z)}{2^{(w-z)}} + 1 \quad \text{for} \quad x > 0, y > 0, z \geq 0
\]

(9)

**Remark 6:** In the 3D-Graph the Values \( \equiv 4 \pmod{6}\) are found in all Layers for \( z > 0 \).

Important point: Please notice that all Values in a Slice in the 3D-Graph share identical NOD / \( p_{End} \).

Note: In the following it is recommended to use the mental picture of six different Types of Pearls:

The Odd Pearls all have the same Shape (Tetrahedron) but are distinguished by three different color-codes,

- the Pearls with a Value \( \equiv 1 \pmod{6}\) are Yellow
- the Pearls with a Value \( \equiv 3 \pmod{6}\) are Grey
- the Pearls with a Value \( \equiv 5 \pmod{6}\) are Blue
The Even Pearls all have the same Shape (Sphere) but are distinguished by three different Sizes, the Pearls with a Value $\equiv 0 \pmod{6}$ are Small, the Pearls with a Value $\equiv 2 \pmod{6}$ are Normal, the Pearls with a Value $\equiv 4 \pmod{6}$ are Large. Note. The even Pearls can have additional color-code corresponding to the odd Pearl for the Pillar.

### 3.5 The 4D version of Matrix Alpha a.k.a. The 4D-Graph

When we introduce Time, $t$, to the 3D-Graph we get a 4D-Graph where it is now possible to consider Flow relative to the Initial Position (IP) for $t = 0$. The Nodes are no longer fixed to the IP but are transported to the Next Position in TCG (a new set of coordinates $(x, y, z)$) when $t$ increases by one unit.

The IP is unique; it is the only situation where all the Odd Pearls are located in the Layer for $z = 0$

The First Position (FP) for $t = 1$ is also unique; it is the only situation where no Odd Pearls are located in the Layer for $z = 0$

In the Second Position (SP) for $t = 2$ Odd Pearls are found in the Layer for $z = 0$ for the IP that contained Blue odd Pearls.

In the Third Position (TP) for $t = 3$ Odd Pearls are found in the Layer for $z = 0$ for the IP that contained Yellow odd Pearls.

One third of the Odd Positions, the ones with a Value $\equiv 3 \pmod{6}$ will always contain only one Node i.e. the Node with the Initial Position $(x, y, t)$.

Please notice that we meet the paradoxes from Hilberts Hotel when we observe situations for $t > 0$:

In the IP (and only in the IP) all Positions $(x, y, z)$ in the 4D-Graph contains exactly one Pearl/Node.

In the FP all the Positions where the Initial Value (IVA) $(6M - 2) \equiv 4 \pmod{6}$ now contains two Nodes i.e. the Node with the Value $2(6M - 2)$ and the Node with the Value $(2M - 1)$.

All other Positions $(x, y, z)$ in the FP contains one Pearl/Node i.e. the Node with the IP $(x, y, z + 1)$.

To avoid the effect of the known Loop 1-4-2-1 a Drain is defined for the 4D-Graph:

When a Pearl/Node is at the Position $(x, y, z) = (1, 1, 0)$ (IP for Node 1) it will not go to $(x, y, z) = (1, 1, 2)$ (IP for Node 4) but will go to the Position $(x, y, z) = (1, 1, -1)$ i.e. it leaves The Domain for the IP-Graph.

We now define “The Codomain” as the “Pillar” $(x, y, z) = (1, 1, -z)$ filling up from $z = 0$ for $t$ growing.

For the IP at $t = 0$ all Positions in the Codomain are empty (except for $(1, 1, 0)$ containing the Origo-Node 1)

We now have the opportunity to define two different scenarios: The TNT-Scenario and The Distance-Scenario.

1. The TNT-Scenario assumes that all Strings/Links that arrive in the Codomain via $(1, 1, 0)$ take on the fixed Length of one (1) Unit.

   In this case the Positions $(1, 1, -z)$ will be populated by the finite Group of Values that have $n_{End} = (t - z)$

2. The Distance-Scenario assumes that all Strings/Links have the original Length i.e. the Type TOD Strings are one (1) Unit, but the Type TOD have the Length defined by the Distance-formula (Equation 7).

   In this case the Origo-Node 1 is a Total Distance from any individual Node in the Codomain corresponding exactly to the Distance from $(1, 1, 0)$ to the Values Initial Position in the 4D-Graph i.e. it is the same Distance in the Codomain and in the Domain for the 4D-Graph.

Note: The authors believe that no two Nodes are the same distance from $(1, 1, 0)$ in the 4D-Graph.

If we assume that the 4D-Graph is a “Real-world” model we have the opportunity (as a Thought-experiment) to assume that both the Pearls and the Strings are “real”. We can then “pull” the Origo-Pearl from Position $(1, 1, 0)$ to Position $(1, 1, -1)$ and assume that all Pearls that “feel the pull” moves one Position in the 4D-Graph.

In this case all Nodes Adjacency-Linked to the Origo-Node will move one Position in the 4D-Graph.

If all Odd Nodes leaves Layer $z = 0$ in the Thought-experiment (and we see the FP for all Pearls) then all Nodes in the 4D-Graph are Adjacency-Linked to the Origo-Node (and Collatz Conjecture is True).

The following concerns the Groups of (Sapling) Values in Rows $(x, y, 0)$ that are Adjacency-Linked to Groups of Values $\equiv 4 \pmod{6}$ in a specific Pillar in the 3D-Graph.
3.6 Matrix Beta

Figure 4: Matrix Beta containing $\beta$-values i.e. the $X_{L/R}$ to find Children in TCT

The values in cells / positions $(X,Y)_{L/R}$ in Matrix Beta points to the Row in Matrix Alpha where values that share identical COS are found (Children in TCT). The values in the Row are “adjacency-linked” to the $(6X-Q)$-value for the Row i.e. the $\alpha$-value with the same coordinates $(X,Y)$ for the cell / position in Matrix Alpha.

Please notice the color-codes; Blue refers to the Left part and Yellow to the Right part of Matrix Alpha.

3.6.1 Construction of Matrix Beta

![Table I. Mini-Matrix showing $L/R(X,Y) \equiv K \pmod{6}$, $K \in \{1,3,5\}$ for $(X,Y) \equiv k \pmod{3}$, $k \in \{0,1,2\}$](image)

The dynamics emphasized in the Mini-Matrix makes it possible to generate Matrix Beta from Matrix Alpha.

All values $L/R(X,Y) \equiv 5 \pmod{6}$ are adjacency-linked to all $\alpha_Y$ in Row $L/\left(\frac{L/R(X,Y)+1}{6}, Y_L\right)$

All values $L/R(X,Y) \equiv 1 \pmod{6}$ are adjacency-linked to all $\alpha_Y$ in Row $R/\left(\frac{L/R(X,Y)+5}{6}, Y_R\right)$

Definition of Beta-values

$\beta_L := \frac{L/R(X,Y)+1}{6}$ for $L/R(X,Y) \equiv 5 \pmod{6}$

$\beta_R := \frac{L/R(X,Y)+5}{6}$ for $L/R(X,Y) \equiv 1 \pmod{6}$

Please notice that Matrix Beta gives a complete overview for all adjacency-links between all possible pairs of odd numbers. The number of Operations TNT to reach $(6X - Q)$ is known for all $(2M - 1) = L/R(X,Y)$.

3.7 Matrix Gamma-zero

Figure 6: Matrix Gamma-zero containing TNT to reach the $(6X-Q)$-value for the Row

Matrix Gamma-zero reflects Conclusion 4 and 5. All Rows on the Left-side contain identical values, as does all Rows on the Right-side in this Matrix.

When number of Operations, $n_{End}$, to reach 1 for $(6X_{L/R} - Q)$ is known, then it is known for the entire Row $X_{L/R}$

Matrix Gamma is generated by adding $n_{End}$ for $(6X - Q)$ in the help-columns to the (known) Gamma-zero values.
3.8 Matrix Gamma

Figure 7: Matrix Gamma containing $\gamma$-values i.e. $n_{\text{mod}}$ for all the $\alpha$-values in Matrix Alpha

In the help-columns TOD is for the entire Row $X_{L/R}$ and TNT is for the unique $(6X_{L/R} - Q)$-value for the Row $X_{L/R}$

In the illustration for Matrix Gamma is only included values that can be verified “moving away” from The Origo.

3.9 Discussion of The Collatz Graph Flow-Diagram and the Matrices

When (manually) constructing The Pattern each odd number $(2M - 1)$ was paired with the “blueprint-layer”, $M$, with the convention that lower $M$ is above higher $M$ (i.e. visible) when two “beams” cross. But this “layering” is sort of arbitrary and does not hold much information.

The Adjacency-Matrix is independent of the “layering” so we are free to use the “layering” in any way that can give us information and there is one specific way that is particularly important.

We can use the “blueprint-layers” to keep track on the number, NOD, of odd operations needed to reach The Origo 1. In this case the black “backbone” $2^q(2(1) - 1)$ is in Layer 0 and all values $\{1,5,21,85,341,...\}$ in Layer 1.

This “happens” to be the Row $R(1,Y)$ in The Double 2D Matrix so now we know TOD and TEO for all values in this Row; $\text{TOD} = 1 \land \text{TEO} = 2^Y \Rightarrow \text{TNT} = \text{TOD} + \text{TEO} = (1+2^Y)$ where TNT is total number of Operations.

$R(1,2) = 5 \equiv 5 \pmod{6}$ so all values in Row $L(1,Y)$ are linked to 5 and are two TOD from 1 (Layer 2)

$R(1,4) = 85 \equiv 1 \pmod{6}$ so all values in Row $R(15,Y)$ are linked to 85 and are two TOD from 1 (Layer 2)

Let us take one more step to clarify:

$L(1,2) = 13 \equiv 1 \pmod{6}$ so all values in Row $R(3,Y)$ are linked to 13 and are 3 TOD from 1 (Layer 3) etc.

It is possible to program an “Intelligent Autonomous Self-replicating Operator” (IASO)

The IASO is able to update the entire Matrix Gamma (starting in Row $R(1,Y)$) using simple “laws”.

The authors have used the mental image of a “bug” magically appearing at the origin for a Row.

The first assignment is to update the entire Row with correct $\gamma$-values.

The second assignment is to “walk” the unique Row and make clones when Matrix Beta contains a value. These clones magically appear in Row $L/R(\beta,Y)$ etc.

Remember that the Collatz Graph Flow-Diagram exists, as it is defined from the Adjacency-Matrix.

If the Conjecture is False then the method of constructing the pattern from the “backbone” keeping track of NOD (the “generation” of the “bug”) will not yield a complete pattern.

But how can this be possible? The authors can see no possible way that two different methods of generating the same pattern can give different results.

As far as the authors can see, constructing The (de facto existing) Pattern in the “picture” each odd number at a time from 1 and up (The Pattern is then by definition “complete”) should yield the same pattern as constructing it from the “backbone” and “activate” values when the relevant “blue print layer” is reached and all odd numbers in that specific layer has a (known) finite NOD ($p_{\text{End}}$) and TNT ($n_{\text{End}}$).
3.10 Comments on the ratio of the fraction NOD divided by N

It appears that this ratio is strictly lower than 2. Remember from earlier that
\[ \ldots \xrightarrow{adj} 2^4(2M - 1) \xrightarrow{adj} 2^8(2M - 1) \xrightarrow{adj} 2^4(3M - 1) \xrightarrow{adj} 2^8(3M - 1) \xrightarrow{adj} \ldots \]
and
\[ 2^4(2M - 1) > 2^8(3M - 1) > 2^8(2M - 1) \text{ for all } M > 1. \]

As stated previously this is important. The following is an attempt to explain why this is important:

It is obvious that any TEO results in a smaller value (with identical odd part) but it follows from the above, that there are two values involved in all TOD where one is higher than the other:
\[ 2^4(2M_1 - 1) > 2^8(3M_1 - 1) > 2^8(2M_2 - 1), \quad q \geq 0. \]

The significant detail to notice here is, that for these two values the odd parts are different.

At least locally (globally would assume the Conjecture is true) we know, that after a TOD the relevant odd values (2\(\ldots\)1) changes layer as this is how \(\text{Origo 1}\) (M → L) is defined in both the “each odd number at a time” and the “start from the backbone and count TOD” approach, but what remains to be proven is, that this change of layer always brings us a unit closer to the \(\text{Origo 1}\).

And we are locally sure, that all (2\(M_1 - 1\) “locks” two values \(2^4(2M_1 - 1)\) and \(2^8(3M_1 - 1)\) thereby blocking “immediate future use”, but this also includes all the other even values \(2^8(2M_1 - 1)\) and \(2^8(2M_2 - 1)\).

If the value \(2^8(3M_1 - 1)\) is even or odd depends on the parity of \(M_1\) (see 7.2 “Detail about hypothetical divergent COS”). The point here is, that in any “local neighbourhood” of a given Start-value \(N\), there is only a limited supply of “very even” values, so the CCS will encounter values of type \((4V + 1)\) where \(M\) is odd. Depending on “how odd” (size of \(Y\) in \(L/R(X, Y)\)) the encounter can lower the series values by several orders of magnitude.

As more and more TOD are seen in CCS, more and more even values are “locally locked” and at some point the CCS “run out” of even values to use as each TOD blocks a \(2^8(2M_2 - 1)\).

So apparently the number of TOD, NOD, can never exceed double the start-value \(N\) (see Detail, Appendix B.3).

3.11 Important observation

The Conjecture has been confirmed \([1]\) for all \(N < 2^{68}\), but having Matrix Gamma (zero) in mind, this has some implications that are worth mentioning:

1. The Conjecture is confirmed for all \((6X-1) & (6X-5) < 2^{68}\)
2. \(2^{68} \approx 2.95 \times 10^{20} \approx 3 \times 10^{20}\) so the Conjecture is confirmed for the first \(5 \times 10^{19}\) Rows \(X_{L/R}\)
3. This implicates that all values in the Rows are confirmed for all Matrices

Please realize that this implicates, that in reality the Conjecture is confirmed not for just all \(N < 2^{68}\) but for all the (infinitely many in each Row for \(Y\) tending to infinity) \(\alpha\)-values in the first \(4 \times 10^{19}\) Rows.

3.12 An Example

It is not necessary to know the actual (base 10) value of \(N = R(12, 300000)\) to know that:
\[ R(12, 300000) = \frac{2^{2(300000)1}6(12) - 5}3 = \alpha_k \equiv 5 \text{ (mod 6)} \]
as
\[ X \equiv 0 \text{ (mod 3)} \land Y_{\nu} \equiv 1 \text{ (mod 3)} \]

Number of TNT to 1 \((n_{\text{TNT}})\) is:
\[ \text{TNT}(\alpha_k) = 1 + 2(300000) + \text{TNT}(67) = 6000003 + 27 = 6000030 \]

Number of TOD to 1 \((p_{\text{TOD}})\) is:
\[ \text{TOD}(\alpha_k) = 1 + \text{TOD}(67) = 1 + 8 = 9 \]
\(\alpha_k\) is adjacency-linked to the Row \(L(\beta_L, Y)\) where it is now confirmed that the entire Row is linked to \(\alpha_k\):
\[ \beta_L = \frac{\alpha_k + 1}6 \Rightarrow L(\beta_L, 1) = (4V - 1) \]

Please notice that we can be certain without actually performing the series with all TOD & TEO.

Please notice that the example holds for \(any\) number of zeros \((0)\) in \(N = R(12, 30\ldots01)\).
3.13 Comments and preliminary conclusions

Speaking as engineers, the authors are forced to conclude that the Collatz Conjecture cannot be false.

Speaking as hobby-mathematicians, the authors are forced to conclude that there is still work to be done!

The authors believe that it is possible to reach a valid proof in (at least) one of the following ways:

1. The authors expect that a formal analysis of the matrices Alpha, Beta, and Gamma will be able to put some mathematical basis to the naive observation: “All the values in the two help-columns \((6X - 1)\) and \((6X - 5)\) are also \(\alpha\)-values so all the values in Matrix Alpha must be part of the same pattern”.

2. Notice that we are locally sure, that two \(\alpha\)-values in the same row cannot be adjacency-linked. e.g. 5 and 21 are both adjacency-linked to 16 but we can not go from 5 to 21 (or vice-versa) applying any finite number of TOD & TEO (similar for all values \(\alpha > 1\) in the row).

If it can be shown that we can be globally sure for all rows it would prove the Conjecture.

3. Showing that the ratio NOD over N is strictly lower than 2 for all N will prove the Conjecture.

4. Analyzing the graph associated with the Copenhagen Tree can prove the Conjecture.

The argumentation in the following is based on the first and last of the proposed ways to a valid proof.

4 Analyses of the congruence of odd N modulus \(\{4,6,8,12,24\}\)

4.1 The Sub-Domain Nodes

To gain some insight in the behaviour of the “Dynamics” governing RCC a table is constructed to be able to find patterns for the different subgroups of congruence-classes.

For reference the individual cells in the 3x3-Table is given unique reference-numbers from one to nine.

| 4A - 1 | 6B - 1 | 6B - 5 | 6B - 3 |
|--------|--------|--------|--------|
| 8A - 7 | 4     | 5     | 6     |
| 8A - 3 | 7     | 8     | 9     |

Figure 8: Table II. Subgroups of odd values

The individual cells in Table II contains all the values that can be expressed with \((A, B) \in \mathbb{N}^2\).

4.2 The Criterion Table

Table III contains the specific criteria for the individual subgroups to exist.

| Criterion Table | 6B - 1 | 6B - 5 | 6B - 3 |
|-----------------|--------|--------|--------|
| 4A - 1          |解读|解读|解读|
| 8A - 7          |解读|解读|解读|
| 8A - 3          |解读|解读|解读|

Figure 9: Table III. Criteria for Subgroups

In Table III the Domain is marked for \(H(2M-1) = \text{integer} \left(\frac{3(2M-1)+1}{2}\right)\) for max possible \(q\).

The specific criteria corresponds to the subgroups \(N \equiv C \mod 24\) shown in Table IV.
4.3 The Subgroups modulus 24

In Table IV the Codomain is marked for $H(2M-1) = \text{integer } \frac{(3(2M-1)+1)}{2^q}$, for max possible $q$.

It is now possible to analyse how the individual subgroups behave in relation to RCC.

Table V contains the result after a Type 3N+1-Operation, a TOD, for each subgroup.

4.4 Table V. The result after one TOD

In Table V the high-lighted values in parenthesis are next value, $N^{p+1}$, in the COS.

4.5 The Six Node Graph (SNG)

The “Dynamics” revealed in Table V gives the opportunity to draw a finite Graph.

Please notice the two “Loops” $1 \iff 1$ and $5 \iff 5$ for the sub-group-nodes 1 and 5.

Any Simple Loop must be found in Group 1 or in Group 5.

A Simple Loop (SL) is when a value Maps to itself.
The known SL Loop-value, 1, is found in subgroup 5.

A SL is only possible for \(3(2M - 1) + 1 = (2M - 1)2^q\) if \(M = 1\) and \(q = 2\) i.e. for \((2M - 1) = 1\).

### 4.6 Domain and Codomain for RCC

It follows from Table V, that specific domain-subgroups have specific codomain-subgroups:

For Domain \(\{1,2,3\}\) the Codomain is \(\{1,4,7\}\)

For Domain \(\{4,5,6\}\) the Codomain is \(\{2,5,8\}\)

For Domain \(\{7,8,9\}\) the Codomain is \(\{1,2,4,5,7,8\}\) depending on parity of \(A\)

If we ignore the groups not in the Codomain i.e. \(\{3,6,9\}\) where values have 3 as a factor we have:

For Domain \(\{1,2\}\) the Codomain is \(\{1,4,7\}\)

For Domain \(\{4,5\}\) the Codomain is \(\{2,5,8\}\)

For Domain \(\{7,8\}\) the Codomain is \(\{1,2,4,5,7,8\}\) depending on parity of \(A\)

Remark 7. Any possible Out-Flow from all the Nodes \(\{1,2,4,5\}\) in Figure 12 is known.

The values in the sub-groups \(\{1,2,3\}\) and \(\{4,5,6\}\) are called “First Branch Values” (FBV).

### 5 The Copenhagen Tree

#### 5.1 Constructing The Copenhagen Tree

We can generate The Copenhagen Tree from the Origo 1 after defining Rules for Child-Nodes. We have to define different rules for the FBVs and “Growing Branch Values” (GBV) to be able to generate the Tree.

All alpha-values are odd and are associated with specific coordinates \((X,Y)_{L/R}\) in Matrix Alpha. We define the Start-value to be the Origo 1. This is a “First Branch Value”, \(R(1,1) \equiv 1 \mod 6\).

The next branch-values comes from multiplying by four and adding one i.e. the values in \(R(1,Y)\).

#### 5.2 Definition 2. Definition of TCT

The Copenhagen Tree (TCT) is defined as:

The Origo-Node 1 is the Root of the Tree. Direction away from the Root is defined as “UP” and direction towards the Root as “DOWN”.

The odd \(N\) is the value of the Parent-Node.

- If \(N = (2M - 1) \equiv 1 \mod 2\) then \(N\) has a Branch-Child-Node \(NB := (4N + 1)\)
- If \(N = (6X - 5) \equiv 1 \mod 6\) then \(N\) has a Right-Child-Node \(NR := (8X - 7)\)
- If \(N = (6X - 1) \equiv 5 \mod 6\) then \(N\) has a Left-Child-Node \(NL := (4X - 1)\)

The main point in the following argumentation is the fact that Definition 2, where the Graph for TCT is shown, corresponds exactly to Definition 1, where the concept for Matrix Alpha is shown.

All \((8X - 3)\) connects to \((2X - 1)\), all \((4X - 1)\) connects to \((6X - 1)\), all \((8X - 7)\) connects to \((6X - 5)\).

It is observed that all odd nodes has a Branch-Child-Node. Also the definition reveals that all \(NR = (8X - 7)\) are found in the Column \(R(X,1)\) in Matrix Alpha and all \(NL = (4X - 1)\) in \(L(X,1)\).

When the index B-Branch, R-Right or L-Left is added for each Step a unique Index-Series is generated which gives all the individual Nodes in The Tree a “Branching-code”.

When constructing TCT it is practical to use Matrix Beta, as the \(\beta\)-value in a position \((X,Y)_{L/R}\) indicates the Row to find Left-Children, if the \(\alpha\)-value in the position \(\equiv 5 \mod 6\), and Right-Children if the \(\alpha\)-value in the position \(\equiv 1 \mod 6\). If \(\alpha \equiv 3 \mod 6\) \(\iff \beta = \{\emptyset\}\).

As all Branch-Child-Nodes are in the same Row as the FBV-Node all the “Direct” Child-nodes are known.

The Copenhagen Tree (TCT) is obviously a (type of) Binary Tree (BIT) as each Parent-node can have one or two Children. A BIT can not contain any Loops and please notice, that Matrix Alpha do not contain the known SL as all individual odd values are in exactly one cell/Position in the Matrix.
5.3 Illustration of TCT with six complete levels

Figure 13: The Origo and first six complete levels of the Graph for TCT

5.4 Illustration of TCT with seven complete levels

| Level | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|
| R     | 1 | B | L | R | B | L | R | B | L | R | B | L  |
| L     | 1 | B | L | R | B | L | R | B | L | R | B | L  |

Please notice in Figure 14, that all the Branches generated corresponds to Rows in Matrix Alpha. When all the Saplings growing from the values $N \equiv Q \pmod{6}$, $Q \in \{1, 5\}$, connects to the cut Branches via the FBV-Nodes, then TCT contains a Route from the Origo to any and all Nodes.

Figure 14: The first seven levels in TCT with “Branching-code”
Notice that one Step in TCT corresponds to moving one cell/Position in Matrix Alpha.

The \( \beta \)-values are found in the left-most columns in Figure 14.

Matrix Alpha is in essence the (ordered) collection of unique “Saplings” cut from TCT.

5.5 Analyses of \( \alpha \)-values modulus 3

Consider an odd alpha-value \( T_C = (2M - 1) = L/R(X_C, Y_C) \)

If \( T_C = (6X_A - 5) \equiv 1 \) (mod 6) it has a Right-Child,

\[
TR := 8X_A - 7 = R(X_A, 1) \Rightarrow X_A = (T_C + 5)/6 = \beta_R
\]

If \( T_C = (6X_B - 1) \equiv 5 \) (mod 6) it has a Left-Child,

\[
TL := 4X_B - 1 = L(X_B, 1) \Rightarrow X_B = (T_C + 1)/6 = \beta_L
\]

If \( T_C \equiv 3 \) (mod 6) it has no “Side-child”

Now consider the Branch-Children

\[
T_C = L/R(X_C, Y_C)
\]

\[
\Rightarrow L/R(X_C, Y_C + 1) := 4T_C + 1
\]

\[
\Rightarrow L/R(X_C, Y_C + 2) := 16T_C + 5
\]

\[
\Rightarrow L/R(X_C, Y_C + 3) := 64T_C + 21
\]

Now consider \( T_C = 3C + k \equiv k \) (mod 3), \( k \in \{0, 1, 2\}, C \geq 0 \)

\[
4T_C + 1 = 4(3C + k) + 1 = 12C + 4k + 1
\]

\[
k = 0 : 12C + 4k + 1 = 12C + 1 = 6(2C) + 1 \equiv 1 \) (mod 6)
\]

\[
k = 1 : 12C + 4k + 1 = 12C + 5 = 6(2C) + 5 \equiv 5 \) (mod 6)
\]

\[
k = 2 : 12C + 4k + 1 = 12C + 9 = 6(2C + 1) + 3 \equiv 3 \) (mod 6)
\]

\[
16T_C + 5 = 16(3C + k) + 5 = 48C + 16k + 5
\]

\[
k = 0 : 48C + 16k + 5 = 48C + 5 = 6(8C) + 5 \equiv 5 \) (mod 6)
\]

\[
k = 1 : 48C + 16k + 5 = 48C + 16 + 5 = 6(8C + 3) + 3 \equiv 3 \) (mod 6)
\]

\[
k = 2 : 48C + 16k + 5 = 48C + 32 + 5 = 6(8C + 6) + 1 \equiv 1 \) (mod 6)
\]

\[
64T_C + 21 = 64(3C + k) + 21 = 192C + 64k + 21
\]

\[
k = 0 : 192C + 64k + 21 = 192C + 21 = 6(32C + 3) + 3 \equiv 3 \) (mod 6)
\]

\[
k = 1 : 192C + 64k + 21 = 192C + 64 + 21 = 6(32C + 14) + 1 \equiv 1 \) (mod 6)
\]

\[
k = 2 : 192C + 64k + 21 = 192C + 128 + 21 = 6(32C + 24) + 5 \equiv 5 \) (mod 6)
\]

5.5.1 Conclusion 6

We can be certain that for triples of Nodes in a Branch \( L/R(X_C,Y) \) in Matrix Alpha

\[
\Leftrightarrow L/R(X_C,Y_C) \Leftrightarrow L/R(X_C,Y_C+1) \Leftrightarrow L/R(X_C,Y_C+2) \Leftrightarrow
\]

One of the Nodes is congruent to 1, one to 3 and one to 5 (mod 6).

5.6 Analyses of the \( \beta \)-values modulus 3

Now consider patterns in the beta-values.

If \( L/R(X_C,Y_C) \equiv Q \) (mod 6), \( Q \in \{1,5\} \) then Matrix Beta has a value \( \beta_{L/R} = B_{L/R}(X_C,Y_C) \)

Note: Careful with the indices as \( L \) and \( R \) in \( \beta_{L/R} \) and \( B_{L/R} \) are different references.

\[
L/R(X_C,Y_C) \equiv 1 \) (mod 6) \( \Leftrightarrow \beta_R = (L/R(X_C,Y_C) + 5)/6 = X_A
\]

\[
L/R(X_C,Y_C) \equiv 5 \) (mod 6) \( \Leftrightarrow \beta_L = (L/R(X_C,Y_C) + 1)/6 = X_B
\]

We have from before the implication \( L/R(X_C,Y_C+3) = 64T_C+21 \), and we know that;

\[
L/R(X_C,Y_C) \equiv k_C \) (mod 6) \( \Leftrightarrow L/R(X_C,Y_C+3) \equiv k_C \) (mod 6)
So what is the pattern in the beta-values?

We have to analyze \( L/R(X, Y) \equiv 1 \pmod{6} \) and \( L/R(X, Y) \equiv 5 \pmod{6} \) in turn.

Let's begin with \( L/R(X, Y) \equiv 1 \pmod{6} \) which is of type \( T = (6C - 5) \Rightarrow \beta_0 = C \)

\[
L/R(X, Y + 3) = 64T + 21 = 64(6C - 1) + 21 = 64(6C - 64) + 21 = 6(64C - 49) - 5 \Rightarrow \beta_3 = (64C - 49)
\]

Now consider \( C = 3A + k \equiv k \pmod{3}, k \in \{0, 1, 2\}, A \geq 0 \)

\[
k = 0 : \beta_3 = (64C - 49) = 64(3A + k) - 49 = 192A + 0 - 49 = 192A - 49 \equiv 2 \pmod{3}
\]

\[
k = 1 : \beta_3 = (64C - 49) = 64(3A + k) - 49 = 64A + 64 - 49 = 64A + 15 \equiv 0 \pmod{3}
\]

\[
k = 2 : \beta_3 = (64C - 49) = 64(3A + k) - 49 = 192A + 128 - 49 = 192A + 79 \equiv 1 \pmod{3}
\]

We find a similar result for \( L/R(X, Y) \equiv 5 \pmod{6} \) of type \( T = (6C - 1) \Rightarrow \beta_0 = C \)

\[
L/R(X, Y + 3) = 64T + 21 = 64(6C - 1) + 21 = 64(6C - 64) + 21 = 384C - 43 = 6(64C - 7) - 1 \Rightarrow \beta_3 = (64C - 7)
\]

Now consider \( C = 3B + k \equiv k \pmod{3}, k \in \{0, 1, 2\}, B \geq 0 \)

\[
k = 0 : \beta_3 = (64C - 7) = 64(3B + k) - 7 = 192B + 0 - 7 = 192B - 7 \equiv 2 \pmod{3}
\]

\[
k = 1 : \beta_3 = (64C - 7) = 64(3B + k) - 7 = 192B + 64 - 7 = 192B + 57 \equiv 0 \pmod{3}
\]

\[
k = 2 : \beta_3 = (64C - 7) = 64(3B + k) - 7 = 192B + 128 - 7 = 192B + 121 \equiv 1 \pmod{3}
\]

5.6.1 Conclusion 7

We can be certain that for triples of L-Saplings or R-Saplings all \( k \in \{0, 1, 2\} \) are present.

6 The Algorithms

Appendix B contains five different Algorithms with examples coded in Thonny Python:

1. The Complete Collatz Rule Series (CCS)
2. The Collatz Odd Series (COS)
3. The Complete TCT Series (CTS)
4. The TCT Odd Series (TOS)
5. The TCT “UP” Algorithm (TUP)

The CCS-Algorithm uses the RCC to generate The Complete Collatz Series.

The COS-Algorithm is in essence the CCS-Algorithm with one line of code deactivated.

The COS has direction DOWN (i.e. towards The Origo/Root 1) and it is possible to define Algorithms for Matrix Alpha / TCT with direction DOWN i.e. Algorithms that terminate at The Root 1.

The CTS-Algorithm a.k.a. Algorithm Alpha generates the (only possible) Route in TCT / Matrix Alpha from any odd Start-value to The Root 1. It does so by moving from Node to adjacent Node (DOWN) in TCT.

The TOS-Algorithm is in essence the CTS-Algorithm with one line of code deactivated.

A very important point is the fact that the COS-Algorithm and the TOS-Algorithm generates absolutely identical Odd-Series, but are based on two different sets of rules.
The TUP-Algorithm generates the next three generations of Child-Nodes (in direction UP) for any odd Start-value. If it is changed to Run continuously (using the Node with the value 5 as the Start-value) it is in essence the IASO described previously. The TUP-Algorithm was used when generating Figure 14 and would be able to continue the pattern seen in the illustration indefinitely!

An other very important point is the fact that identical “Branching-codes” are generated with the CTS-Algorithm (DOWN) and the TUP-Algorithm (UP). This fact is the Corner-Stone for the argumentation in the present work.

The following contains some details about CTS as Algorithm Alpha is essential for the argumentation.

6.1 Algorithm Alpha

![Algorithm Alpha](image)

6.2 Definition 3. Definition of CTS / Algorithm Alpha

The Route in Matrix Alpha for any odd Start-value T:

Choose an odd Start-value T₀:

- If $T_i \equiv 5 \pmod{8}$ then $T_{i+1} = (T_i - 1)/4$ #The $T_i$-value is a Branch-Child
- If $T_i \equiv 1 \pmod{8}$ then $T_{i+1} = 6(T_i + 7)/8 - 5$ #The $T_i$-value is a Right-Child
- If $T_i \equiv 3 \pmod{4}$ then $T_{i+1} = 6(T_i + 1)/4 - 1$ #The $T_i$-value is a Left-Child

Repeat

This iterative algorithm generates the complete Route to The Root 1 from any odd start-value. It is designed / defined as the exact opposite of Definition 2.
7 Discussion

As mentioned in the high-lighted lines in the code-example, Algorithm Alpha is able to generate the complete COS as it is “included” in the Complete TCT-Series. All it takes is to deactivate one line of code from Algorithm Alpha thereby not printing the Branch-Child-Steps in Matrix Alpha.

By only printing the Left-Child-Steps and Right-Child-Steps a Series identical to COS is generated. This follows from the alternative definition of Matrix Alpha (Equation 5 and 6).

The two formulas are repeated here:

\[(6X-1) = \frac{3 \cdot L(X,Y_L) + 1}{2 \cdot Y_L} \quad \text{All } L(X,Y_L) \text{ are one TOD and } (2Y_L - 1) \text{ TEO from (6X-1)}\]

\[(6X-5) = \frac{3 \cdot R(X,Y_R) + 1}{2 \cdot Y_R} \quad \text{All } R(X,Y_R) \text{ are one TOD and } (2Y_R) \text{ TEO from (6X-5)}\]

If L/R(X,Y) is a Branch-Child (Y > 1) it will in the COS jump to the (6X-1) or (6X-5) value in one go, while in the Complete TCT-Series we see every Step in TCT from Node to adjacent Node. The COS/TOS is seen when we only register the (6X-1)- or (6X-5)-Nodes from the Complete TCT after a FBV is reached.

7.1 The Nine Node Graph (NNG)

Notice an important detail; in TCT all the odd nodes are included. The Codomain for TCT is identical to the Domain as the sub-groups \(\{3,6,9\}\) from Table II are now included. This is an important difference when comparing the Complete TCT-Series and the COS. It changes the Flow seen in SNG in important ways i.e. \(\{3,6,9\}\) now have “In-arrows” as seen only for the \(\{1,2,4,5,7,8\}\)-subgroup-Nodes in “The Six Node Graph”.

The Flow in direction DOWN in The Nine Node Graph depends on congruence of the odd value modulus 6 and 8.

The following table list the Domain and Codomain for the subgroup-Nodes from Table II.

| Domain  | Codomain |
|---------|----------|
| \(4X-1\) | 1, 2, 3, 1, 4, 7 |
| \(8X-7\) | 4, 5, 6, 2, 5, 8 |
| \(8X-3\) | 7, 2, 5, 8 |
| 8       | 3, 6, 9 |
| 9       | 1, 4, 7 |

Figure 17: Table VI. Domain and Codomain for the NNG

Notice that together the Branch-Children has the entire Domain as Codomain, while the FBVs together has the same Codomain as RCC / SNG.

Notice that we again can identify the “Loops”, 1 ⇐⇒ 1 and 5 ⇐⇒ 5

All other values must go to an other Domain-Node after a Step in TCT.

It seems there is no obstructions for the Flow in the Graph associated with Table VI.

7.1.1 Conclusion 8

When Stepping in TCT it is possible to reach Nodes from all sub-domains.
7.2 Detail about hypothetical divergent COS

\[ M_0 = (2D - 1)2^d \]

\[ T_0 = (2M_0 - 1) = (2D - 1)2^{d+1} - 1 \]

\[ 3T_0 + 1 = 3(2M_0 - 1) + 1 = 3((2D - 1)2^{d+1} - 1) + 1 \]

\[ = 3((2D - 1)2^{d+1} - 2) \]

\[ = 2(6(2D - 1)2^{d-1} - 1) \]

\[ T_1 = (3T_0 + 1)/2^1 = 3(2D - 1)2^d - 1 \]

\((2D-1)\) is odd and 3 is odd so whether \(T_1\) is odd or even depends on \(d > 0\) (odd) or \(d = 0\).

\[ 3(2D - 1) - 1 = 2(3D - 2), d = 0 \]

\[ T_1 = (2M_1 - 1) \Rightarrow M_1 = 3(2D - 1)2^{d-1} = 3M_0/2, d > 0 \]

So \(T_0\) can only “Loop around” in the sub-group \(\{1\}\) exactly \(d\) cycles.

The above detail implicates that all COS will contain an “even mix” of values \(N \equiv 1 \pmod{4}\) and values \(N \equiv 3 \pmod{4}\).

As long as \(d > 0\) the COS “Loops around” in \(\{1\}\) while the values gets higher for each \(p\), but at some point the \(d\) hits zero and then the value is \(N \equiv 1 \pmod{4}\).

7.2.1 Conclusion 9

No divergence can exist in the COS as no \(T\) has infinite \(d\).

7.3 Final remarks about computability

As it is possible to design Algorithm Alpha, it is possible on “A Perfect Computer” to Run Algorithm Alpha for any odd Start-value and see \(p_{End}\) and \(n_{End}\) when the Root is reached.

The problem is “computable”; Algorithm Alpha Stops at \(T = 1\) for any and all odd Start-T.

If the keen readers try to Run the code-example in Appendix B.3 they will observe, that a Child-code is written for each Step. This gives the identical “Branching-code” (reversed) in the direction DOWN as would be observed if the pattern in Figure 14 was continued (using the TUP-Algorithm in B.5) until the Test-value appear in a Level.

As the same “Branching-code”, i.e. Position in the Binary Tree TCT, is generated in direction DOWN as direction UP it proves, that indeed all odd positive values are part of TCT and hence of the Collatz Graph.

8 Final Conclusion

The Collatz Conjecture is True

References

[1] Barina, D. “Convergence verification of the Collatz problem.” J Supercomput 77, 2681–2688 (2021). https://doi.org/10.1007/s11227-020-03368-x
9 Appendix A. How to generate the Collatz Graph Flow-Diagram

9.1 The Adjacency Matrix

![Figure 18: Adjacency Matrix Raw (all blank squares \((N_1, N_2)\) contains an invisible zero)](image)

The RCC implicates that all Odd Nodes \((2M - 1)\) are linked to two other Nodes i.e. the Node \((6M - 2)\) (DOWN) and the Node \((2M - 1)^2\) (UP)

The Even Nodes \((2M - 1)^2\) with Values \(\equiv 0 \pmod{6}\) and Values \(\equiv 2 \pmod{6}\) are also linked to two Nodes i.e. the Node \((2M - 1)^{2z-1}\) (DOWN) and the Node \((2M - 1)^{2z+1}\) (UP)

The Even Nodes \((6M - 2)\) with Values \(\equiv 4 \pmod{6}\) are linked to three Nodes i.e. the Nodes \((3M - 1)\) (DOWN) and \((12M - 4)\) (UP) and the Odd Node \((2M - 1)\) (UP)

![Figure 19: Adjacency Matrix with Diagonal-Line (marked with “8”-symbols)](image)

![Figure 20: Adjacency Matrix with Diagonal-Line “Folded in Diagonal”](image)
9.2 The Odd and Even Bars

The Odd Bars

A Full TOD for \((2M - 1)\) actually goes from the “8” horizontally to the “1” and vertically down to the “8” below. The distance travelled is then \(2(2M + 1)\) i.e. double the “numerical distance”.

But as \((6M - 2) = 2(3M - 1)\) is even, in the next Step it has no choice other than a TEO resulting in \(2^q(3M - 1)\) so considering \(\text{Flow}\) (Where will the value go next?) it is possible to skip the TEO and go to \(3M - 1\).

In Fig. 21 this is indicated by the crossed out squares, so for any \((2M - 1)\) you go to \((3M - 1)\) on the Double-line.

The Even Bars

A Full TEO for \(2^q(2M - 1), q > 0\), actually goes from the “8” vertically up to the “2” and left to “8”.

The distance traveled is then \(2^q(2M - 1)\) i.e. double the “numerical distance”.

But if \(q = 1\) then \(q - 1 = 0\) and the next value in CCS odd, then the next step will be a TOD so considering \(\text{Flow}\), it is possible to skip the TOD and go from \(2^1(2M - 1)\) to \(2^1(3M - 1)\).

In Fig. 22 this is indicated by the crossed out squares.
9.3 The Complete Pattern

Combining the above “Flow-rules” gives a “Flow-diagram”.

In Fig. 23 the connections from “8” to “2” in odd rows are removed (crossed out in Fig. 22) as they are redundant once you are inside the pattern. It is not possible to remove the crossed out positions/connections in Fig. 21 as these are necessary for the Flow in Fig. 23.

Start anywhere in the pattern and you are forced to follow the Flow-direction.

In the illustration is used the convention that each “beam” is associated with an Odd Value \((2M - 1)\) and that lower \(M\) is above higher \(M\) in the “layering” i.e. the beam with the lower \(M\) is visible when two beams cross.

Notice how the “tartan-like” pattern gradually fills out as more beams are added in Fig. 24.
9.4 The Collatz Flow-Diagram

In Fig. 24 it was attempted to use a unique color $M$ for each “odd beam” $(2M - 1)$ but even a supercomputer would run out of individual colors to use, and Fig. 24 do actually not show much about the Flow except that all the individual colored beams are “similar” in shape.

For clarity all beams $3(2M - 1)$ in Fig. 25 are “folded in the Diagonal” erasing the beams (grays in Fig. 24) from the upper part of the “picture”. In a Flow-sense they are not in The Pattern.

Please notice, that if the $3(2M - 1)$-beams are “folded back” to Fig. 24 it is possible to make a similar “shadow-print” for any $p(2M - 1)$ for odd $p$, showing an identical mirror-pattern with “cells” in the mirror-pattern of size $p(2M - 1) * 2p(2M - 1)$. Actually removing any of these mirror-patterns for $p \neq 3^d$ would interrupt The Pattern thus breaking the Flow.

In the CFD illustrated here is used eight different color-codes:

The black “backbone” represents the values $(2M - 1)^2$ for $M = 1$ i.e. $2^z$ where we see the Origo 1 for $z = 0$
The color Grey indicates that the Value $\equiv 3 \pmod{6}$ incl $\{3, 9, 15\} \pmod{18}$
The color Red indicates that the Value $\equiv 1 \pmod{18}$
The color Purple/Blue indicates that the Value $\equiv 5 \pmod{18}$
The color Orange/Light Brown indicates that the Value $\equiv 7 \pmod{18}$
The color Turquoise/Light Blue indicates that the Value $\equiv 11 \pmod{18}$
The color Yellow indicates that the Value $\equiv 13 \pmod{18}$
The color Green indicates that the Value $\equiv 17 \pmod{18}$

Again please notice the “tartan-like” pattern gradually filling out when more beams are added in Fig. 26.
9.5 The Copenhagen Labyrinth (TCL)

The last figure is an earlier version of a flow-diagram, made before realizing that the Adjacency-Matrix was of course the “natural habitat” for such a Flow-Diagram (i.e. the CFD).

Please notice that all colored positions are associated with a direction. TCL is special in the sense that it is impossible to get lost in TCL and from every position in TCL there is exactly one Route to The Origo 1.

TCL is actually “a compressed version of a geometrically based projection” from a model that has the integer number-line as “backbone” and the value 1 as Origo. The model has some interesting features e.g. it is possible to trace the Columns in Matrix Alpha as endpoints of the vertical connections/links that “line up”. This becomes more clear with increasing Distance. Also it seems to be possible to construct formulas predicting the sum-total of Up- and Down-arrows.

TCL is a subject for future work as an in-depth analysis of TCL is outside the scope of the present work.

Figure 27: “Rigid pipeline” a.k.a. TCL, Close-up-view

As a Thought-experiment we can imagine building a Real-world model with gravity pointing in the same direction as the arrows in the first row (for 1) i.e. the direction “Down”.

We can then fill The Pipeline with water, pumping it in at the Origo, while we monitor the Level reached and the Volume of water needed to reach the Level.

It will take one Unit to fill up Level 1

It will take one Unit to fill up Level 3 (two Total)

It will take five Units to fill up Level 5 as water now spills over from Pipe 1 (vertical) to Pipe 5 and also fill up “3” (seven Total)

It takes two Units to fill each of the Levels 7, 9 and 11.

It takes six Units to fill Level 13 (nineteen Total)

It takes three Units to fill Level 15 as Pipe 13 now also fills up (twenty-two Total)

It takes twenty-four Units to fill Level 17 as Pipe 13 spills over to Pipe 17 which again spills over to Pipe 11 and Pipe 7. Now all odd \( N < 15 \) are “activated” so (like 5) 17 is some kind of Threshold-value

There is an unlimited amount of Threshold-values, e.g. 3077 will need “a lot” of water as Pipe 27, 31, 41 etc. gets “activated”.

The model has a metric in two dimensions and when Time and Flow are introduced we have “a well-defined function” for the Volume used vs. Level reached.
“From the pattern shown in the flow-diagram TCL, The pipeline is 'construct-able!' (new pumps, pipes & fittings etc. when needed).

It seems that the pattern shown in the illustrations must go on forever i.e. the TCL-flow-diagram is describing one Pipeline including all odd values. A loop is not possible in the pattern illustrated.
10 Appendix B. The Algorithms

The following examples are coded in Thonny Python

The Appendix contains five different Algorithms

1. The Complete Collatz Rule Series (CCS)
2. The Collatz Odd Series (COS)
3. The Complete TCT Series (CTS)
4. The TCT Odd Series (TOS)
5. The TCT “UP” Algorithm (TUP)

10.1 The Complete Collatz Rule Series (CCS)

This algorithm uses the Rule in Collatz Conjecture (RCC) to generate
# The Complete Collatz Series (CCS) including both odd and even values

```python
Total : int = 0
Stop : int = 0
Odd : int = 0

Test = int(input("Please input an integer Testvalue "))
print(" ")

while Stop == 0 :

    if Test % 2 == 1 : # Rule for Odd Values
        Test = int(3*Test + 1)
        Odd = Odd + 1
        Total = Total + 1
        print("#Total Operations: ",Total," #Odd Operations: ",Odd," Value in Series: ",Test)
        print(" ")
    if Test % 2 == 0 : # Rule for even Values
        Test = int(Test / 2)
        Total = Total + 1
        print("#Total Operations: ",Total," #Odd Operations: ",Odd," Value in Series: ",Test)
        print(" ") # If the line above (print) is deactivated with # then the COS is generated

    if Test == 1 :
        print("The total amount of odd Operations: ", Odd)
        print("The total amount of Operations was: ", Total , " odd and even operations")
        Stop = 1

CCS-example with 15 as Start-value
```

Please input an integer Testvalue  15

```
#Total Operations:   1    #Odd Operations:   1    Value in Series:   46
#Total Operations:   2    #Odd Operations:   1    Value in Series:   23
#Total Operations:   3    #Odd Operations:   2    Value in Series:   70
#Total Operations:   4    #Odd Operations:   2    Value in Series:   35
#Total Operations:   5    #Odd Operations:   3    Value in Series:  106
#Total Operations:   6    #Odd Operations:   3    Value in Series:   53
#Total Operations:   7    #Odd Operations:   4    Value in Series:  160
#Total Operations:   8    #Odd Operations:   4    Value in Series:   80
#Total Operations:   9    #Odd Operations:   4    Value in Series:   40
#Total Operations:  10    #Odd Operations:   4    Value in Series:   20
#Total Operations:  11    #Odd Operations:   4    Value in Series:   10
#Total Operations:  12    #Odd Operations:   4    Value in Series:    5
```
#Total Operations: 13  #Odd Operations: 5  Value in Series: 16
#Total Operations: 14  #Odd Operations: 5  Value in Series: 8
#Total Operations: 15  #Odd Operations: 5  Value in Series: 4
#Total Operations: 16  #Odd Operations: 5  Value in Series: 2
#Total Operations: 17  #Odd Operations: 5  Value in Series: 1

The total amount of odd Operations: 5
The total amount of Operations was: 17 odd and even operations

This shows that CCS for 15 as initial value is:

\[
15 - 46 - 23 - 70 - 35 - 106 - 53 - 160 - 80 - 40 - 20 - 10 - 5 - 16 - 8 - 4 - 2 - 1
\]

Please notice that the two values \(p_{End}\) and \(n_{End}\) are known when the Algorithm terminates.

Please notice that CCS and COS are identical except the one line of code that prints the TEO

10.2 The Collatz Odd Series (COS)

# This algorithm uses the Rule in Collatz Conjecture (RCC) to generate
# The Collatz Odd Series (COS) including only odd values

```
Total : int = 0
Stop : int = 0
Odd : int = 0

Test = int(input("Please input an integer Testvalue "))
print(" ")

while Stop == 0 :
    if Test % 2 == 1 :
        print("#Odd Operations: ", Odd , " Value in Series: ", Test )
        print(" ")
        Test = int(3*Test + 1)
        Odd = Odd + 1
        Total = Total + 1
    if Test % 2 == 0 :
        Test = int(Test / 2)
        Total = Total + 1
    if Test == 1 :
        print("#Odd Operations: ", Odd , " Value in Series: ", Test )
        print(" ")
        print("The total amount of odd Operations was: ", Total , " odd and even operations")
        Stop = 1
```

COS-example with 15 as Start-value

Please input an integer Testvalue 15

#Odd Operations: 0  Value in Series: 15
#Odd Operations: 1  Value in Series: 23
#Odd Operations: 2  Value in Series: 35
#Odd Operations: 3  Value in Series: 53
#Odd Operations: 4  Value in Series: 5
#Odd Operations: 5  Value in Series: 1

The total amount of odd Operations: 5
The total amount of Operations was: 17 odd and even operations

This shows that COS for 15 as initial value is:

\[
15 - 23 - 35 - 53 - 5 - 1
\]
10.3 The Complete TCT Series (CTS) a.k.a. Algorithm Alpha

This algorithm gives the Complete TCT-Series (CTS) to The Root 1 for any odd input-value
# The TCT-Series describes the Route in Matrix Alpha (Node to adjacent Node) from the input to 1
# Please notice that the "Branching-code" (position in Binary Tree) is generated as well

Detail = str("Odd grows either 0 or 1 per Step and Total grows either 2 or 3 per Step")

Total: int = 0
Step: int = 0
Odd: int = 0
T = int(input("Input odd value: "))
High = T
Start = T

while T > 1:
    if T > High :
        High = T
    if T % 8 == 1 :
        T = int(3*(T+7)/4 - 5) # T = (8X-7), i.e (4X+1) for X even, steps to (6X-5)
        Total = Total + 3
        Step = Step + 1
        Odd = Odd + 1
        print("Step: ", Step,"N: R ", int(T) ," Total: ",Total," Odd: ",Odd)
    elif T % 4 == 3:
        T = int(3*(T+1)/2 - 1) # T = (4X-1), incl (8X-1) & (8X-5), steps to (6X-1)
        Total = Total + 2
        Step = Step + 1
        Odd = Odd + 1
        print("Step: ", Step,"N: L ", int(T) ," Total: ",Total," Odd: ",Odd)
    elif T % 8 == 5:
        T = int((T-1)/4) # T = (8X-3), i.e (4X+1) for X odd, steps to X
        Step = Step + 1
        Total = Total + 2
        print("Step: ", Step,"N: B ", int(T) ," Total: ",Total," Odd: ",Odd)
    if T == 1:
        Total = Total + 3
        Odd = Odd + 1
        print("Step: ", Step,"N:Root 1"," Total: ",Total," Odd: ",Odd)
        print(" "
        print("Odd Start-value: ",Start)
        print(" ")
        print("Highest value in series: ",High)
        print(" ")
        print("Number of only odd iterations: ",Odd)
        print("Number of steps in Matrix Alpha: ",Step)
        print("Total number of all iterations: ",Total)
        print(" ")
        print("Detail: ",Detail)

CTS-example with 15 as Start-value

Input odd value: 15
Step: 1 N: L  23 Total:  2 Odd:  1
Step: 2 N: L  35 Total:  4 Odd:  2
Step: 3 N: L  53 Total:  6 Odd:  3
Step: 4 N: B  13 Total:  8 Odd:  3
Step: 5 N: B  3 Total:  10 Odd:  3
Step: 6 N: L  5 Total:  12 Odd:  4
Step: 7 N: B  1 Total:  14 Odd:  4
Step: 7 N:Root 1 Total:  17 Odd:  5

Odd Start-value:  15

Highest value in series:  53
Number of only odd iterations:  5
Number of steps in Matrix Alpha:  7
Total number of all iterations:  17

Detail:  Odd grows either 0 or 1 per Step and Total grows either 2 or 3 per Step
10.4 The TCT Odd Series (TOS)

This algorithm gives the TCT-Odd-Series (TOS) to 1 for any odd input-value. The TCT generated TOS is identical to the CCS generated COS.

```python
Total: int = 0
Step: int = 0
Odd: int = 0
T = int(input("Input odd value: "))
High = T
Start = T
while T > 1:
    if T > High :
        High = T
    if T % 8 == 1 :
        T = int(3*(T+7)/4 - 5) # T = (8X-7), i.e (4X+1) for X even, steps to (6X-5)
        Total = Total + 3
        Step = Step + 1
        Odd = Odd + 1
        print("Odd: ",Odd," N: R ", int(T) ," Total: ",Total," Step: ", Step)
    elif T % 4 == 3 :
        T = int(3*(T+1)/2 - 1) # T = (4X-1), incl (8X-1) & (8X-5), steps to (6X-1)
        Total = Total + 2
        Step = Step + 1
        Odd = Odd + 1
        print("Odd: ",Odd," N: L ", int(T) ," Total: ",Total," Step: ", Step)
    elif T % 8 == 5:
        T = int((T-1)/4) # T = (8X-3), i.e (4X+1) for X odd, steps to X
        Step = Step + 1
        Total = Total + 2
    if T == 1:
        Total = Total + 3
        Odd = Odd + 1
        print("Odd: ",Odd," N:Root", int(T) ," Total: ",Total," Step: ", Step)
        print(" ")
        print("Highest value in series: ", High)
        print(" ")
        print("Number of only odd iterations: ", Odd)
        print("Number of steps in Matrix Alpha: ", Step)
        print("Total number of all iterations: ", Total)

TOS-example with 15 as Start-value

Input odd value: 15
Odd: 1 N: L 23 Total: 2 Step: 1
Odd: 2 N: L 35 Total: 4 Step: 2
Odd: 3 N: L 53 Total: 6 Step: 3
Odd: 4 N: L 5 Total: 12 Step: 6
Odd: 5 N:Root 1 Total: 17 Step: 7

Odd Start-value: 15

Highest value in series: 53
Number of only odd iterations: 5
Number of steps in Matrix Alpha: 7
Total number of all iterations: 17

This shows that TOS for 15 as initial value is:

15 − 23 − 35 − 53 − 5 − 1

The very important point here is the fact, that the TOS generated from TCT is absolutely identical to the COS generated from RCC. This shows that algorithms based on two different sets of rules can generate identical Odd-Series.
10.5 The TCT “UP” Algorithm (TUP)

This algorithm generates the next children in The Copenhagen Tree for any Odd value.
The Origo is the Root 1, R is Right-child, L is Left-child and B is Branch-child.

T: int = 1

while T > 0:
    R: int = 0 ; L: int = 0 ; B: int = 0
    RR: int = 0 ; RL: int = 0 ; RB: int = 0
    LR: int = 0 ; LL: int = 0 ; LB: int = 0
    BR: int = 0 ; BL: int = 0 ; BB: int = 0
    RRR: int = 0 ; RRL: int = 0 ; RRB: int = 0
    RLR: int = 0 ; RLL: int = 0 ; RLB: int = 0
    RBR: int = 0 ; RBL: int = 0 ; RBB: int = 0
    LRR: int = 0 ; LRL: int = 0 ; LRB: int = 0
    LLR: int = 0 ; LLL: int = 0 ; LLB: int = 0
    LBR: int = 0 ;LBL: int = 0 ; LBB: int = 0
    BRR: int = 0 ; BRL: int = 0 ; BBR: int = 0
    BLR: int = 0 ; BLL: int = 0 ; BBL: int = 0
    BBB: int = 0

    T = int(input("Input Odd value: "))

# Children
# For T
    if T % 6 == 1 :
        R = int(4*(T+5)/3 - 7)
        print(" N: ", T ," => ","R ", R )
    elif T % 6 == 5 :
        L = int(2*(T+1)/3 - 1)
        print(" N: ", T ," => ","L ", L )
    if T % 2 == 1: 
        B = int(4*T + 1)
        print(" N: ", T ," => ","B ", B )

# Grand-Children
# For R
    if R % 6 == 1 :
        RR = int(4*(R+5)/3 - 7)
        print(" N: ", R ," => ","RR ", RR )
    elif R % 6 == 5 :
        RL = int(2*(R+1)/3 - 1)
        print(" N: ", R ," => ","RL ", RL )
    if R % 2 == 1: 
        RB = int(4*R + 1)
        print(" N: ", R ," => ","RB ", RB )

# For L
    if L % 6 == 1 :
        LR = int(4*(L+5)/3 - 7)
        print(" N: ", L ," => ","LR ", LR )
    elif L % 6 == 5 :
        LL = int(2*(L+1)/3 - 1)
        print(" N: ", L ," => ","LL ", LL )
    if L % 2 == 1: 
        LB = int(4*L + 1)
        print(" N: ", L ," => ","LB ", LB )

# For B
    if B % 6 == 1 :
        BR = int(4*(B+5)/3 - 7)
        print(" N: ", B ," => ","BR ", BR )
    elif B % 6 == 5 :
        BL = int(2*(B+1)/3 - 1)
        print(" N: ", B ," => ","BL ", BL )
    if B % 2 == 1: 
        BB = int(4*B + 1)
        print(" N: ", B ," => ","BB ", BB )
# Great-Grand-Children

## For RX

```python
# For RR
if RR % 6 == 1:
    RRR = int(4*(RR+5)/3 - 7)
    print(" N:", RR ," => ","RRR ", RRR )
elif RR % 6 == 5:
    RRL = int(2*(RR+1)/3 - 1)
    print(" N:", RR ," => ","RRL ", RRL )
if RR % 2 == 1:
    RRB = int(4*RR + 1)
    print(" N:", RR ," => ","RRB ", RRB )

## For RL
if RL % 6 == 1:
    RLR = int(4*(RL+5)/3 - 7)
    print(" N:", RL ," => ","RLR ", RLR )
elif RL % 6 == 5:
    RLL = int(2*(RL+1)/3 - 1)
    print(" N:", RL ," => ","RLL ", RLL )
if RL % 2 == 1:
    RLB = int(4*RL + 1)
    print(" N:", RL ," => ","RLB ", RLB )

## For LB
if LB % 6 == 1:
    LBR = int(4*(LB+5)/3 - 7)
    print(" N:", LB ," => ","LBR ", LBR )
elif LB % 6 == 5:
    LBL = int(2*(LB+1)/3 - 1)
    print(" N:", LB ," => ","LBL ", LBL )
if LB % 2 == 1:
    LBB = int(4*LB + 1)
    print(" N:", LB ," => ","LBB ", LBB )
```

## For LX

## For LR

## For LL

## For LB

# For BX
# For BR
if BR % 6 == 1:
    BRR = int(4*(BR+5)/3 - 7)
    print("N: ", BR, " => ", "BRR ", BRR )
elif BR % 6 == 5:
    BRL = int(2*(BR+1)/3 - 1)
    print("N: ", BR, " => ", "BRL ", BRL )
if BR % 2 == 1:
    BRB = int(4*BR + 1)
    print("N: ", BR, " => ", "BRB ", BRB )

# For BL
if BL % 6 == 1:
    BLR = int(4*(BL+5)/3 - 7)
    print("N: ", BL, " => ", "BLR ", BLR )
elif BL % 6 == 5:
    BLL = int(2*(BL+1)/3 - 1)
    print("N: ", BL, " => ", "BLL ", BLL )
if BL % 2 == 1:
    BLB = int(4*BL + 1)
    print("N: ", BL, " => ", "BLB ", BLB )

# For BB
if BB % 6 == 1:
    BBR = int(4*(BB+5)/3 - 7)
    print("N: ", BB, " => ", "BBR ", BBR )
elif BB % 6 == 5:
    BBL = int(2*(BB+1)/3 - 1)
    print("N: ", BB, " => ", "BBL ", BBL )
if BB % 2 == 1:
    BBB = int(4*BB + 1)
    print("N: ", BB, " => ", "BBB ", BBB )

TUP-example with 15 as Target-value

Input Odd value: 5
N: 5 -> L 3
N: 5 -> B 21
N: 3 -> LB 13
N: 21 -> BB 85
N: 13 -> LBR 17
N: 13 -> LBB 53
N: 85 -> BB 113
N: 85 -> BBB 341

Input Odd value: 53
N: 53 -> L 35
N: 53 -> B 213
N: 35 -> LL 23
N: 35 -> LB 141
N: 213 -> BB 853
N: 23 -> LLL 15
N: 23 -> LLB 93
N: 141 -> LBB 565
N: 853 -> BB 1137
N: 853 -> BBB 3413

It is known that
5 has the “Branching-code” B so
35 has the “Branching-code” (B)LBB and
15 has the “Branching-code” (BLBB)LLL

This is absolutely identical to the (reversed) “Branching-code” seen in the CTS-example and for a good reason; it is the same rules (TUP in direction UP and CTS (inverted rule) in direction DOWN) that define the “Branching-code”.

This last fact is the Corner-stone of the argumentation in the present work.