Optimization and Comparison of Human Methods for Solving the Rubik’s Cube

Govind Sharma   
*Something*  
*University of Waterloo*Waterloo, Canada  
[g45sharma@uwaterloo.ca](mailto:g45sharma@uwaterloo.ca)

Govind Sharma   
*Something*  
*University of Waterloo*Waterloo, Canada  
[g45sharma@uwaterloo.ca](mailto:g45sharma@uwaterloo.ca)

*Abstract*— Rubik’s Cubes are popular for both general solvers and computer scientists. There are many methods of solving the cube that designed for either humans or computers. Some popular methods for human solvers are CFOP, Roux, Petrus, and ZZ, etc. And some popular methods used in computer science for the cube are the Thistlethwaite’s algorithm, Kociemba’s algorithm and the IDA\* method, etc. This project focusses on striking a balance between the two groups of methods. Using the most appropriate computer techniques to facilitate better solves of human methods is the primary objective. Applying the IDA\* algorithm on smaller problems within the cube, optimal solutions to CFOP and Roux methods were found with decent interpretability for humans in both cases.

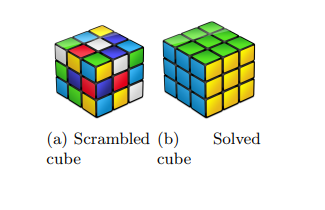
Keywords—component, formatting, style, styling, insert (key words)

# Introduction

The Rubik’s Cube (3D combination puzzle) was invented in 1974 by an Architecture professor Erno Rubik. Ever since its invention and conception as a toy, it has taken over the toy and puzzle industry by storm. It was a household object in the 1980s when everyone from children to adults were trying their hand at a Rubik’s Cube. Although its popularity declined at the turn of the century but recently it has picked up again as this toy has turned into a competitive game under the WCA (World Cubing Association) which organizes thousands of yearly events worldwide with events related to Rubik’s Cube puzzles of different shapes, sizes and stipulations.

The inspiration behind designing the cube was for professor Rubik to demonstrate a stable 3-dimensional design structure that can move in all directions. The first ever design was made using wood and rubber bands. But through many versions, he finally settled on a primitive version of the cube we see today. The 6 sides of the cube were colored for aesthetic purpose. He had no intentions of the structure being conceived as a puzzle. But after having done a few twists and turns on the cube, the professor soon realized that it was not at all straight forward to put the pieces back into their original position so as to have a solid color on each side of the cube. It was then that he realized that he has invented a revolutionary puzzle which both very easy to conceive and annoyingly difficult to solve. In fact, it is this innate simplicity of the puzzle that makes it so tempting as it firmly remains intractable for most people of try it. To say that Rubik’s Cube was a success would be an understatement of the highest order. With over 350 million copies of the cube being solved worldwide, the Rubik’s Cube takes the crown as the most successful toy/puzzle ever to sold in human history.

The classic version of the Rubik’s Cube also known as the Magic Cube, has the dimensions of 3x3x3. This simply means that the cube with 6 faces has 9 stickers of the same color on each side. The most common color scheme for the cube includes the colors yellow, green, white, blue, orange and red even though other color variations do exist. The most natural way of visualizing the cube is to see it in layers that can move together. The puzzle can be scrambled by rotating any of its layers on the fixed central pieces together. The rotations are 90 degree turns in any direction. A combination of these rotations of any arbitrary number of layers causes cube to be in a scrambled state.



Even though the number of ways to turn the cube is small but even with the limited possibilities of turning, the cube can have 43,252,003,274,489,856,000 combinations in total. Such a large number of possible legal states makes the cube an interesting challenge even for computers.

## Application Domain

This project delves into the application domain of Game Playing. As discussed, Rubik’s Cube is a competitive game nowadays and there are many methods suitable for humans to solve the cube as quickly as possible. These methods are broken down into various intuitively recognizable stages in order to make the whole solution tractable. The intended results from this project can help draw new insights into how each stage in a solution can be optimized and therefore possibly improve the current methods. The mechanics of the Rubik’s cube allow the pieces to move in conjunction to each other hence making it a classic example of a problem in Group Theory.

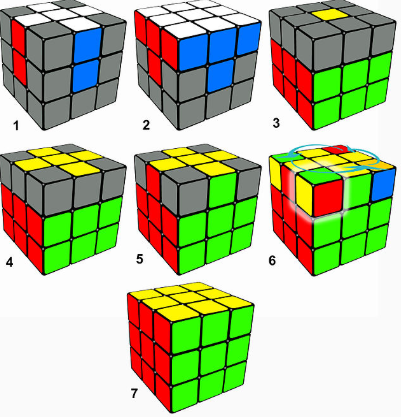
## Problem To Be Solved

The following problems are of interest to this project:

1. Can there be a system that can produce solutions to the Rubik’s Cube which can be interpreted or learned by human solvers?

There are already many solutions of the Rubik’s Cube that solve it completely and as efficiently as mathematically possible (God’s Number). Some of these methods use popular search techniques like IDA\* in conjunction with popular algorithms designed specifically for computer solutions of the Rubik’s Cube such as the Thistlethwaite’s algorithm (and its variations). But the solutions produced by these methods are beyond our comprehension. There is not much intuitive interpretation to the moves in these solutions. The major reason for this lack of interpretability is the fact that these computer algorithms by design try to put as many pieces as possible back into their intended positions on the cube in one go i.e. the rotations are intended to effect as many pieces as possible. Therefore, the purpose of each move becomes very obscure from a human perspective.

This issue could be solved if the instead of solving the entire cube using these efficient techniques, we solve the cube partially. Specifically, the solution can be divided into stages that correspond to popular human methods. Each stage of the solve can be attempted using intelligent search techniques to get a sequence of moves that solve only that stage of the solution. Knitting the sequence of moves generated for each stage together will produce a solution for the entire cube and by design this solution should be close to what a human can comprehend.

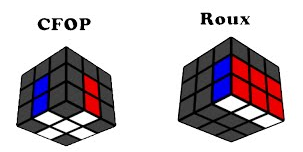


1. Which family of methods: Block Building or Layer by Layer produce better solutions using intelligent heuristic search techniques.

There are many popular methods of speed solving the Rubik’s Cube. Some of these methods are CFOP (Fridrich method), Roux method, ZZ method, Petrus method, etc. But most of these methods fall into two categories or families. These are layer by layer category of methods and the block building family of methods. In the layer by layer category, the methods involved focus on solving each layer of the cube at a time. The CFOP method, for example solved the first two layers together before moving on to the last layer. On the other hand, the block building methods do not limit themselves to consider only layers. They rather focus on building blocks of solid colors and of varying shapes and sizes across the cube as efficiently as possible and then join these blocks to get to completed layers of the cube. An example of a method in this category is the Roux method. It has been empirically established in the speed cubing community that the block building methods on average require a smaller number of moves to solve the cube as compared to the layer by layer methods. But the layer by layer methods are generally faster as they involve a much more rigid procedure which can be practiced into muscle memory of the solvers and produce better times. This happens because the block building methods leave more space for the solver to innovate and build blocks with freedom and since there is more thinking involved in these methods, the solves themselves tend to be longer. The layer by layer methods don’t require as much thinking as the block building methods and therefore, are faster generally.

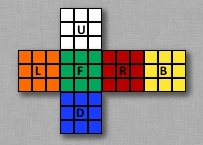
It should be noted that even though the speed and efficiency tradeoff between the two families of methods is well established in the cubing community, the margins between these methods are very small and that is why both types of methods enjoy a heathy subscription. And since speed is the primary objective of solving the cube, the layer by layer methods are the most popular.

Due to the difference in approach of these methods, the stages involved are different. Therefore, it is interesting to see how the intended results from this project effect different types of methods. To be specific, comparisons will be made between the two most popular methods of each type i.e. CFOP for the layer by layer type and Roux for the block building type. Since, the solutions are produced by computers and time of the solves are irrelevant, we will focus on the move count for the solutions.



## Move Notation and Color Configuration

The Rubik’s Cube has six colors and although many configurations of colors are available, the official and the most prevalent color configuration contains the colors yellow, white, green, blue, red and orange. Moreover, the colors are in a set order on the cube. Yellow color is always opposite the white color, green is always opposite the blue color and red is always opposite the orange color. When facing the green color in front and white color on top, red color is on the right side. This is the official configuration and all scrambles are applied from this position only.



A layer on the Rubik’s Cube can be turned in one of three ways, a) 90 degrees clockwise, b) 90 degrees counter-clockwise and c) 180 degrees in any direction. 90 degrees turns are called quarter turns since doing 4 of these in succession returns the cube in its original position. 180 degrees rotations are called half turns as doing 2 of these in succession returns the cube in its original position. Since there are 6 layers on the cube, we have a total of 18 such moves. Each layer on the cube is given a letter notation. The front side (the side facing the solver) is denoted with F, right side is denoted with R, left side is L, top side or the side facing Up is denoted with U, the back side is B and the down side is D. If a clockwise turn of a side is to be conveyed then the letter for that side is used directly. For example, R means quarter turn the right side clockwise. A counter clockwise turn is conveyed by the layer letter followed by a small “i” signifying “inverted”. So, Ri means quarter turn the right side in counter clockwise direction. Half turns are conveyed using the layer letter followed by the number 2 signifying to turn the side twice in any direction. So, R2 means a half turn for the right side. Following this notation, we get the following 18 moves: R, Ri, R2, U, Ui, U2, F, Fi, F2, L, Li, L2, D, Di, D2, B, Bi and B2.

Apart from these 18 moves, we need to consider 3 more moves pertaining to the middle layer of the cube. The middle layer is denoted by the letter M and the three moves corresponding to this layer are M, Mi and M2. The directions of these moves are considered according to the right side. So, M moves like R and Mi moves like Ri. This layer is used in the Roux method as discussed later in the report. Another slice move of the middle layer is denoted by E. This layer is between the Up layer and the down layer.

Sometimes, two layers of the cube are needed for rotation together. These are called fat moves. These are represented by the lowercase letter corresponding to the outer layer involved in the fat turn. For example, turning the M and R layer together is notated by r. The appending for signifying direction remains the same for these moves as well.

After denoting layer/slice moves, there needs to be notations to rotate the entire cube. These moves are called orientation moves. This controls which way the cube is to be handled before applying a sequence. When whole cube rotated along the R side it is denoted by X. When whole cube is rotated along the U side it is denoted by Y. When whole cube is rotated along the F side it is denoted by Z.

## God’s Number

God’s Number refers to the least number of moves required to solve any legal scramble of a 3x3 Rubik’s Cube. The notion of this number being referred to as the God’s Number came from the idea that if God was given a Rubik’s Cube in a scrambled state then no matter how well the cube is scrambled, God (being omniscient) will be able to solve the cube in the most efficient way possible using the least number of moves required. Therefore, the number of moves required by God to solve the hardest scramble ever possible for a Rubik’s Cube is called the God’s Number.

The value of God’s Number for a 3x3 Rubik’s Cube was in speculation for long period of time. Many clever attempts were made to find a lower bound on this number over the years. But finally, in 2010, this lower bound was found using in parts, the Kociemba’s algorithm (discussed later) and running exhaustive searches on servers at Google. The God’s Number is 20. This means that no matter how hard you scramble a Rubik’s Cube it will at worst take 20 moves to solve.

The knowledge of this number is important as it can be used to prune off branches in depth-based searches for a solution of cube in graph traversal/search methods. For example, if solution path goes beyond 20 moves, it can be halted as a solution path with 20 moves or less is guaranteed to be present at some other branch of the search.

# Techniques to solve the problem

## Current Techniques

There have been many (successful) attempts of coming up with efficient solutions to the Rubik’s Cube. Plenty of research has been done in this regard and there many programs across the web that can solve the Rubik’s Cube in the most efficient way possible. Following sections offer a brief summary of some of techniques described across the literature pertaining this particular topic.

### Depth First Search

Depth First Search is a fundamental search/traversal technique popular in the domain of maze finding. Straight forward implementation and simplicity of the data structures involved in DFS lead it to be a very popular technique used in Artificial Intelligence. As mentioned, this technique is applied the most in graph search/traversal.

The solution to a Rubik’s Cube can also be thought of as a graph problem where the starting node of the graph is the cube in its given scrambled position. Each rotation of the cube leads to another node in the graph with the rotation being the edge between the two nodes. The goal is of course to find a path from the starting node to a solved state in this graph. Since it is not advisable to have the entire graph of all possible nodes and moves in memory, the solution can be obtained by growing the tree as various move sequences are applied from the initial scrambled stage. By the very nature of the Depth First Search, one branch of the graph is to be fully exhausted in search of the solution before moving onto a another one. By design, the only path stored in memory for this type of search will be the path between the starting node and the current node at the end of the branch (specified to a specific depth). The time complexity of this search is where is the branching factor i.e. the average number of moves (edges) that can be applied to a given state of the cube in the graph and is the specified depth for graph. The space complexity is where is the branching factor and is the maximum number of moves in a branch.

The obvious advantage of this technique is the efficient (linear) space complexity as only the nodes in one branch are needed in memory for this solution to work. Another benefit is that the solution can be found without exploring the whole tree/graph. But since the state space of the Rubik’s Cube is extremely large, it is probable that DFS will spend most of the its time searching in branches that do not actually contain the solution and since branches can be infinitely long (as there is no limit to the number of moves that can be applied to a Rubik’s Cube) this could take a very long time even if the branches are chopped off at a certain depth (knowing that if a solution were to be found, it would have happened before a certain move count i.e. God’s Number). Such pruning of the branches can be applied to reduce the time complexity but since this method only keeps track of a single branch there is no way of determining if the current branch has morphed into a state that previously seen and subsequently discarded in a previous branch.

### Breadth First Search

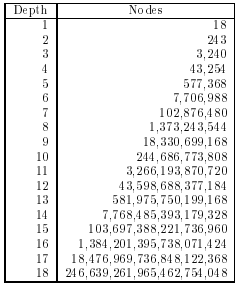
Breadth First Search is another fundamental graph search/traversal technique applied heavily in Artificial Intelligence and game playing domain. As was the case with DFS, BFS too has a straight forward implementation requiring simple data structures. Perceiving the solution to the Rubik’s Cube as a graph search problem is easy as described above. And therefore, BFS can be applied to find the solution.

Unlike in DFS, Breadth First Search looks at all nodes of the graph at a given level before moving in depth in the tree. This means that graph is grown in level by level instead of just growing a single branch. And since the state space for the Rubik’s Cube is very large, the nodes stored in memory for BFS could be very large and often not tractable even for super computers of moderate processing power and memory. The advantage of using Breadth First Search is that it will always find the most efficient solution by design i.e. a solution that uses the least number of moves between the scrambled state and solved state of the cube. This means that if the solution to scramble exist, BFS will find it given enough time and space. The time complexity of this method depends on the number of nodes in the state space and it also depends on the branching factor and the allowed depth of the graph . The time complexity is . But the space complexity of BFS is not linear (unlike in DFS) as all the nodes in the graph are needed to be in memory until a solution is found. This makes the space complexity to also be .

As mentioned, the inherent advantage of using BFS is that it will find the shortest path to the solution and that is complete i.e. if a solution exists, BFS will find it eventually. Since all the nodes in the graph are stored in memory this makes it easier to check for loops in the search space i.e. ignore states that are already present in the graph. But since space required is too much, BFS alone can not be used to solve the cube even on decently powered machines.

### *Iterative Deepening A\**

Exhaustive searches like BFS and DFS are not very effective specially on problems with large state spaces. In order to tackle the problem of large search spaces where exhaustive and blind searches are not practical, we need to apply search methods that make use of the domain knowledge and nudge the searches in the right direction so as to ensure that the solution state is found as soon as possible without having to waste processing time and memory on branches or parts of the graph that are away from the solution. Algorithms that use heuristics to nudge the search towards the solution have better performance over such state spaces. A\* and IDA\* are algorithms of this kind. The following table shows the number of nodes at different depth level of an exhaustive search in the Rubik’s Cube search space.



Even with A\* algorithm, the state space for the Rubik’s Cube is too large and the algorithm is bound to get stuck due to memory limitations. Therefore, it is prudent to apply a heuristic based approach but one which involves an exhaustive search. IDA\* algorithm (as the name suggests) performs an iterative search that grows in depth during subsequent iterations. This algorithm is based on depth search but the gain in effectiveness are made by the pruning of branches based on heuristic functions.

Deciding which heuristics to use is very important consideration. One possible heuristic is Manhattan distance applied to a flattened-out version of the 3-dimensional cube. This includes finding the minimum number of moves needed to place each piece of the puzzle in it right place (as in the solved state) with the correct orientation (position of the colors in the piece). To make this heuristic admissible, this value has to be divided by 8 because each move changes the position of 8 pieces in total (4 corners and 4 edges). A better variation of this heuristic is to calculate the Manhattan distance of corners and edges separately. The expected value of this heuristic is 5.5 (5 for 12 corners each 3 for the 8 edges).

Using these primitive heuristics increase the efficiency of the solution but only to a particular extent (specifically, search is fast till depth of 14 and beyond that the node count supersedes the efficiency gains). A better heuristic function is needed for IDA\* to be more effective in this setting. Heuristic are generally considered to be functions that are computed for each node as it is encountered in the search space. But it is often the case for large enough search spaces that the heuristics are precomputed and stored in look up tables. This ensures that calculation of heuristic for each node in the graph takes a constant and cheap time and thus improving the efficiency of the algorithm greatly. Consider a single corner piece on the Rubik’s Cube. The maximum number of moves required to solve a piece (put it in its solved position in the correct color orientation) for each possible initial position of the piece can be found by applying a simple breadth first search on just that one piece (leaving out the rest of the pieces in the cube). Performing such reduces searches enables the enumeration of the maximum move count of the solution for each corner of the cube. The same search can be performed on the edge pieces as well. Finally, the results of these searches can be cumulated in look up tables that can be made accessible to the IDA\* algorithm during its search for the solution of the entire cube. Such lookup tables for heuristics are referred to as pattern tables. Since, the maximum move count is considered separately for each piece, the heuristic calculation for each move on the Rubik’s Cube at a given state can only be done by considering the maximum of the move counts of the individual pieces. This will ensure that the overall heuristic is admissible.

Using the lookup tables as heuristics and applying the IDA\* algorithm on 10 scrambles (scrambled using 100 random moves each), the results as reported in the original text are promising as each scramble was solved in reasonable time with search space remaining stable in terms of the node count at each level.

### Thistlethwaite's Algorithm

Morwen B. Thistlethwaite is a professor of mathematics out of Britain. He invented the Thistlethwaite’s algorithm in 1981. The algorithm is notable because despite producing very shot move count solution, it has the potential of being morphed into a variation that can be replicated for human solvers. The original method although remains rather complex and is beyond human comprehension. This algorithm in its original form is meant for computer application. There are variations to this algorithm like the Human Thistlethwaite and Heise Method that human interpretable while being very close to the efficiency of the original. The major selling point of this algorithm is that it does not rely heavily on search algorithms to solve the cube. This is partly because at the time its conception, the computing power available was not apt enough to handle search procedures on the problem the size of the Rubik’s Cube. Rather it focuses on the mathematics of the Rubik’s Cube and applies graph theory in order to solve the cube. This algorithm is more important from theoretical standpoint (as better procedures are available) but for a long time, it remained the method that produced the lowest move count solutions to the Rubik’s Cube.

This method approaches the solution in a very different way than the popular layer by layer or block building approaches. It does not focus on putting together groups of pieces (layer or block) in place sequentially. Rather it looks to improve the position of all of the pieces of cube simultaneously. It works by restricting the possibilities for each piece at each step and finally having only one possible solved state. As discussed in the notation section, to any given state of the cube, we can apply 18 different moves. In other words, a combination of these 18 moves can solve any state of the cube. Thistlethwaite’s algorithm works on reducing the size of this set of moves needed to solve the cube at each stage. The first step is to arrive at a position of the cube that can be solved using just the <U, D, L, R, F2, B2> moves. This step is also termed as edge orientation because after completion of this step, all edges in the cube are oriented correctly. Next stage is reduction to <U, D, L2, R2, F2, B2>. This known as corner orientation. Next stage is reduction to <U2, D2, L2, R2, F2, B2>. This is followed by the final step which solves the cube i.e. no moves are required to solve the cube further.

The original algorithm solved the cube in at most 52 moves which is way more than the God’s Number. Improvements on this algorithm such as the Kociemba’s algorithm reduced this move count further.

### Kociemba’s algorithm

Kociemba’s algorithm was an improvement over the Thistlethwaite’s algorithm. It was designed by Herbert Kociemba. He was part of the team which proved the God’s Number to be 20.

This method also focusses on reduction of the move set required to solve the cube but it improves the original algorithm by dividing the solution into two steps. The gain in efficiency are due to the algorithm incorporating search techniques to get more efficient solutions to the partial stages of the whole solution. The search techniques are used to remove symmetries from the search tree and solve the cube with the shortest move count possible. The fundamental idea behind this approach is that it is easier to solve two shorter problem instead of solving a big one. That is why the cube is divided into two stages. The second stage is always the solved state of the cube but the first stage can be anything and it often varies. As an example, the first stage can be considered as the first two layers of the cube being solved. Thus, in the first stage, this algorithm will work to solve the first two layers of the cube and in the second half it will continue to solve the cube from complete state of the first stage. Iterative deepening search techniques are used in the first stage followed by a heavy use of lookup tables for the second stage.

## Technique Adopted for this Project

The method adopted for this project takes inspirations from all of the methods described in the previous section. The complete solution to the Rubik’s Cube is already done and to replicate it using the current technique is not the intention. Rather, the focus is on applying efficient search techniques on smaller subproblems within the cube. In general, the method used follows this procedure.

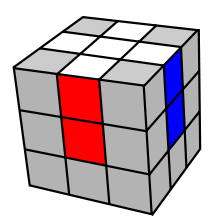
* Start with a scrambled Rubik’s Cube.
* For each stage in the human method:
  + Solve to achieve the stage using IDA\*
* Return the complete solution by concatenating solutions from the individual stages.

Before providing a description of IDA\* used to solve each stage, let’s first look at the two human methods of speed solving that are to be used. More precisely, the different stages of each method will be enumerated for reference.

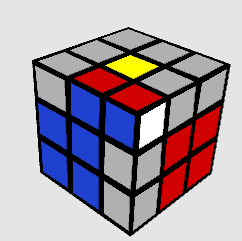
### CFOP (The Fridrich Method)

CFOP stands for Corners, F2L, OLL and PLL. Here F2L simply means First Two Layers, OLL means Orientation of the Last Layer and PLL means Permutation of the Last Layer. This method was invented by Jessica Fridrich and it has emerged as the most popular methods among speedcubers today. This is a layer by layer method which essentially accelerates the traditional method of building one layer at a time by combining the construction of the first two layers followed by a two-step process of finishing off the last layer. The different stages involved in this method are:

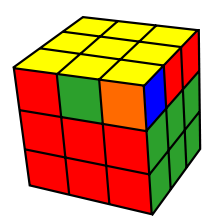
#### Cross: This is the first step and it involves forming a cross shaped on one side (mostly white) of the cube. More specifically, we join the centres of four adjoing faces to the centre of the base face using the correct four edges thus forming a cross that extends to each center. In reality, most solves are color neutral meaning that the cross can be formed on any side of the cube (usually the side which requires least number of moves to get to the cross for that particular scramble). But for the sake of simplicity, only white cross will be considered for this method.



1. F2L: The first two layers are completed in four steps. In each one of these sub steps, a pair is made joining a corner and an edge piece (of matching colors) and inserting them between one of the four slots created by the cross.



1. OLL: The last layer pieces are all oriented correctly. This means that all colors on the top face of the last layer are of the same color. The color depends on what color cross the solution began with. For this method, since the white cross was fixed, this color will always be yellow.



1. PLL. The final step of this method which results in the solved cube is the Permutation of the Last Layer. In this step, all the pieces in the last layer which already in their correction orientations are permuted around to their correct position. This results in a solid color on all sides of the cube.

In CFOP, the last two steps i.e. OLL and PLL are done using predefined algorithms (sequence of moves that move certain pieces of the cube around without effecting others). Since these two steps already have predefined optimal algorithms, we will use them in the solution from a lookup table. The method will only focus on the cross and the four sub steps of the F2L.

The search procedure for CFOP will proceed as follows:

* Start with a scrambled Rubik’s Cube.
* Solve to complete white Cross using IDA\*.
* Solve to complete -green F2L pair using IDA\*.
* Solve to complete the orange-blue F2L pair using IDA\*.
* Solve to complete the blue-red F2L pair using IDA\*
* Solve to complete the red-green F2L pair using IDA\* thus marking the completion of the First Two Layers (F2L).
* Lookup the sequence of moves for OLL.
* Lookup the sequence of moves for PLL.
* Return all the solution moves in correct sequence as the complete solution of the cube.

### Roux Method

The Roux method was developed by invented by Gilles Roux. This method follows the block building approach. It averages a lower move count than the CFOP method and at the same time has similar solve times. But since it is difficult master, most speedcubers don’t practice this method. This method does not focus on completing layers in order to solve the cube. The reason being that by building layers, the move of the cube is restricted and subsequent moves have to keep in mind the previous progress. This hinders the whole solve. To overcome this, Roux method tries keep as many free layers as possible. One of the most important difference between this method and CFOP method is the heavy use of M moves (moving the middle layer of the cube this effecting the position of centers as well). The stages involved in this method are:

1. 3x2x1 block on the left side: This stage is achieved by building block so as to have a mini layer on the left side of the cube. After this step is complete, L moves are no longer needed for the remainder of the solve except for the last algorithmic part. Since a strict cross is not needed, blocks can be formed with liberty and often require fewer number of moves.



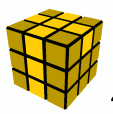
1. 3x2x1 block on the right side: This stage is achieved by building block so as to have a mini layer on the right side of the cube. After this step is complete, R moves are no longer needed for the remainder of the solve except for the last algorithmic part. Since cross pieces are not fixed and M moves are allowed, blocks can be for formed with great freedom and often require very a small number of moves.



1. Top corners: In this step the four corners of the top layer are oriented and permuted at the same time using a small algorithm. Since this step is not done intuitively, it will be done using lookup tables in the method.



1. Solve the remaining 6 edges and centers: In this step the last remaining edges and centers are fixed using a combination of just U and M moves. Since this step too is done using algorithms, it will be completed using lookup tables. After the completion of this step, the whole cube is solved.



The search procedure for Roux will proceed as follows:

* Start with a scrambled Rubik’s Cube.
* Solve to complete a 2x2x1 block on the left side using IDA\*.
* Solve an F2L pair using IDA\* and add it to the solved block to complete a 3x2x1 block on the left side.
* Solve to complete a 2x2x1 block on the right-side using IDA\*.
* Solve an F2L pair using IDA\* and add it to the solved block to complete a 3x2x1 block on the right-side.
* Lookup the sequence of moves to solve the top four corners.
* Lookup the sequence of moves to solve the remaining edges and centers.
* Return all the solution moves in correct sequence as the complete solution of the cube.

### State Representation

Representing the Rubik’s Cube in a computer program is simple but is very easy to not do it in an efficient way. There are two approaches of representing the cube in a programming language. It can be done so as to minimize the space consumed by each state or it can be done so that least amount of computation is needed in order to change states after performing a rotation on the cube.

In the space efficient setting each sticker on the cube is represented using 3 bits. This is because each sticker can have only 6 possible values corresponding to the 6 different colors on the cube. Packing a total of 48 stickers (8 on each side ignoring the center since they are fixed) on the cube into a 48x3 bit structure is as efficient a way to represent the cube as possible. Along with storing the face stickers on the cube, each state also needs to store a trail of moves that lead to that particular state from the starting scrambled state. Since there are 18 total moves, we need it 5 bits to encode every possible move. This representation is not efficient for computations though. This is because each rotation consists of switching stickers around (a turn on the cube switches 21 stickers in total). This means for each turn the machine has to switch around 21x3 bits. But since machines don’t work bitwise, each assignment of 3 bits to the stickers means the machine utilizing a complete machine word to do so. As such the overhead of doing 21x3 bit shifts for every rotation is large.

In the computation efficient state representation, the focus is to represent the cube in terms of pieces, instead of stickers. Each piece on the cube has a position on the cube (out of a total of 20 possible positions) and each cube has three color attributes which represent the piece’s colors in the three dimensions. For edge pieces, one of the colors in null, for center pieces 2 of the colors are none while the corner pieces have all colors defined. What makes this representation computation efficient is the fact that each turn on the cube can be represented as vector multiplication. On each turn, 8 pieces switch places and for each piece two of the colors swap dimensions. Thus, each rotation of the cube can be calculated very swiftly. This representation is a modification of the representation given in the pyglass implementation.

After experimenting with both approaches it was clear that no matter how memory efficient the states of the cube are made out to be, eventually the number of nodes generated by the search graphs will be too many to handle for a laptop machine. And since the whole point is to avoid large search spaces, it is better to go with the representation that offers better computation times instead of trying to save space as that would be futile.

### Uniqueness of States

Referring back to figure 1, the number of nodes generated at level 19 is 246,639,261,965,462,754,048. This number represents the number of possible states this depth of the graph. Keep in mind that this graph was generated by carefully avoiding obvious duplicates at consecutive moves on the cube. The total number of actual possible (legal) positions/states that a 3x3 Rubik’s Cube can have is 43,252,003,274,489,856,000. And the number of moves needed to reach all possible states (God’s Number) is 20. This means it should take 20 moves to enumerate all possibilities. But according to the figure even at move 19, the graph has generated more state than possible. This could only mean one thing that the graph has many duplicate states. Having duplicate states is an obvious by product of the very nature of the cube. As an example, repeating the moves R2U2 on a cube six times (a total of 12 moves) will bring the cube to its original state (from where it started this sequence). This kind of cycles are plentiful in such a graph and they are very expensive to predict and avoid. Keeping such cycles in the graph is very dangerous as it could lead to infinite loops in the search space and no algorithm will ever halt.

One simple fix of this problem is to have some sort of uniqueness in the mechanism. This means that every time a new state is generated in the search space, a quick lookup is performed to see if the new state has been seen before or not. If the new state is seen before then it is omitted from the graph. If the new state is a unique one and has never been seen before then it is kept in the graph. The most efficient way of achieving this sort of test would be come up with a system that can assign a unique hash to each cube state. Once such a hash is obtained, a simple hash table can be kept against which each new state is checked. This hash should have the following two properties:

1. It should be unique for each cube state.
2. It should be quick to calculate.

Generating a unique hash could be as simple as assigning a number to each color and then concatenating the numbers for a state as per the colors in a fixed order. Since only 5 sides of the cubes are enough to fully define it, we would only require to concatenate numbers (or bits for memory efficiency) for 5x8=40 stickers. This method is producing a unique hash for each state but the hash codes produced are difficult to manage. For instances at 3 bits per color and a total of 40 stickers would require a code of 120 bits. A number of this size is not managed by the default data types of most programming languages. Another problem here is the time required to calculate this hash. Since a number can’t be maintained for such a large number, the hash has to be maintained as a string of bits and even if strings can of bits can be compressed into characters, the conversion for each state would take a lot of time.

So, instead of focusing on the hash being absolutely unique for each state, we focus on defining a hash which is very close to being unique for a state but is fast to calculate and easy to store and manage. This can be done by representing each face of cube with a number that is unique to the colors on that face. The numbers for each face can then be concatenated. In order to keep the concatenated number manageable, not all colors on a face are taken into account. To be specific, out of the 8 colors on a face (ignoring the center) only 4 are taken into account. The 4 colors could be on any position but generally either the four corner or the four edges are considered. The numbers generated in this way are then concatenated to form a single number for the whole cube. This number is fast to calculate and easier to store using default numeric data types of any programming language. Consequently, managing a hash table of such numbers is also faster.

Leaving out 4 colors on each face means that the codes generated for each cube aren’t unique. This will defeat the purpose of having numbers to represent each state of the Rubik’s Cube. But in this method, we are interested in solving stages of the cube and not the whole cube. So, at any point in the search we care about only a few pieces of the puzzle. For example, during the cross phase, we only care about the four white edges i.e. a total of 8 colors on 5 faces. Thus, by only caring about a few pieces on the cube, we restrict the total unique combination of the cubes with respect to only the few pieces by a lot. And this reduced number of unique combinations are aptly represented by the hashing system described here. Notice that since in every stage we care about different pieces, the hashing mechanism changes accordingly for each stage. The change is just in the selection of pieces (at most 4 to keep the numbers tractable) to on each face in order to calculate a unique number for that particular face.

Now, the only question remaining for the hash system to work is to determine a way to associate a single number to a face. This can be done by assigning a number to each color and adding the numbers for all the colors (selected) on a particular face. To ensure that a unique combination of colors on a face result in a unique sum we assign the number to colors in such a particular way. The color yellow is always given the number 0. After that the number of any other color is determined by the number of pieces to be selected from a face. Since at max we can afford to select only 4 colors from a face, we multiply the number of the previous color by 4 and add 1 to it. So if the number for yellow was 0 then the number for green will be 0\*4+1 = 1, the number for white will be 1\*4+1 = 5, the number for blue will be 5\*4+1 = 21, the number for orange will be 21\*4+1=85 and finally, the number for red will be 85\*4+1=341. Choosing this numbering scheme ensure that a unique combination of numbers results in a unique sum of the face. While it is possible for a face to have same combination of colors but in a different permutation, but since 5 faces of the cube is considered, this difference in an individual face will reveal itself in other faces thus making the combination unique for each state.

### Identification of Stage Completion

The search algorithm must stop when the cube is solved. In case of this method, the search algorithm should stop when a stage is completed. The check to see if the stage is completed is to be made every time a node is expanded further. Therefore, it is important that this check remains as cheap as possible so as to not pile up to the total time of the search. Fortunately, this is already achieved using the above hashing system. Like all different states of the cube considering only the relevant pieces for a stage, the solved state of these pieces (signifying that the stage is complete) will also have a unique hash associated with it. And at every iteration of the search we just need to check if the state at hand has the same hash as that of the solved state or not. For example, consider the cross stage. The solved cross stage has a cube where all 8 stickers are present at specific positions on the 5 faces and thus the number for this state will be unique and each state in the search for a completed cross can be compared with this hash to know that the search is complete.

### Heuristics

Choosing a heuristic function is important in determining the performance and success of IDA\* algorithm. Since the purpose is finding solutions with interpretability, the first heuristic has to mimic in some manner what a human solver looks for while solving the cube. Performing several solves on the Rubik’s Cube made it clear that one of fundamental need for intuitive solves is to align pieces so that they can be put in their respective pieces with single moves (quarter or half turns). Such pieces are considered to be correctly oriented and not necessarily correctly permuted. A piece once in its correct orientation can be easily permuted. Consider the cross stage. The goal is to place the 4 white edges in their correct place on the white layer. Before placing them into their correct places, all four white edges should be oriented properly no matter where on the cube they end up being. Setup moves are needed to turn a badly oriented piece into an oriented one (For badly oriented edges, an F or B move makes the orientation correct). To have separate setup moves to turn every badly oriented edge individually is not optimum. A move should be chosen if it is able to turn as many badly oriented edges into correct orientation as possible. So, heuristic value of a state at this stage could be the number of badly oriented pieces. This heuristic is not admissible because in many cases turning pieces into correct orientation is not required at all. Badly oriented pieces can still be placed correctly. Also, it has been mathematically proven that any cross can be solved in 8 moves or less and therefore, adding moves just to orient pieces can prove to be wasteful. Using this heuristic is thus not optimal.

Referring back to the modified Manhattan distance heuristic used by R.E. Korf, we know since at every stage of the solve, only a few pieces are of relevance, the Manhattan distance for all the pieces of the cube is not needed. Again, considering the cross stage; at each state of the cube at this stage the heuristic should be the maximum of the Manhattan distance of each of the four white edge pieces. As discussed before, taking a maximum of the Manhattan distances of the pieces involved keeps the heuristic admissible and therefore, the search results are guaranteed to be optimal. The heuristic values for every state cannot be calculated on the fly while the search is in progress as that would take a long time. Instead, the values are precomputed and kept in memory as a lookup to be used by search. Optimal solutions are not necessarily interpretable. This statement is truer when a larger part of the cube is being solved optimally. Solving a small portion like a cross or one F2L pair, optimality of the solution can be interpreted as the move count is low and the number of pieces involved in each stage comprehensible for the human brain.

# Results

Let’s look at one example solve for each method with moves in the solution produced by IDA\* search. The same scramble is used for both solving methods. The scramble is R' U R2 F2 L2 F2 U' R2 F2 U B2 U R2 L' D' F D L' D2 U' B2. After applying this scramble (keeping white on top and green in front), the cube is in the following state:



Note that the orange color is represented in pink in my implementation due to lack of availability of the orange color.

## Example Solve for CFOP

After using the above scrambled cube, the following move sequences for each stage of the CFOP method are generated:

1. Cross: D Ri B2 L Fi
2. Reorient for F2l: Z2
3. 1st F2L Pair: F2 Ui F U F2
4. 2nd F2L Pair: U2 R B2 Ri B2
5. 3rd F2L Pair: R2 U2 Ri U2 R2
6. 4th F2L Pair: U2 R B U2 Bi U2 Ri
7. OLL: U2 r U Ri U R U2 ri
8. PLL: Ui R2 u Ri U Ri Ui R ui R2 Fi U F

Applying the above moves in tandem will solve the entire cube from its starting scrambled position. The total move count for the solution is 49.

Looking at the efficiency of the solve sequence produced above, it is clear that for the stages solved using the IDA\* algorithm, the resulting sequences solve the corresponding stage in the most optimal way possible. But this was expected as the heuristic used for IDA\* was admissible. Another thing to observe here is that the searches for each stage get complete fairly quickly as the search tree did not grow very large. At worst the IDA\* had to go to the depth of 7 (for the last F2L pair) for this scramble in order to complete that stage. So, as expected, efficiency wise, IDA\* produces the best possible results for any particular stage. Attempting the different stages on 10 different scrambles, the move count for each stage for different scrambles has been summarized below:

|  |  |  |  |
| --- | --- | --- | --- |
| Stage | Min | Max | Avg |
| Cross | 5 | 7 | 5.7 |
| 1st F2L | 4 | 6 | 5.3 |
| 2nd F2L | 3 | 7 | 5.0 |
| 3rd F2L | 4 | 7 | 5.4 |
| 4th F2l | 7 | 8 | 7.1 |
|  |  |  |  |

Next the intuition or interpretability of the move sequences in each stage is to be examined. For any cross on a 3x3 Rubik’s Cube, the solution is at worst 8 moves long. Top cubers manage to find this optimal solution more often then not during their speed solve. This is perhaps because cross is the easiest step in the CFOP method and all sides can be moved freely as no other pieces other than the 4 white edges matter at this point. For average level solvers, the intuitive to solve the cross is to break it up in two stages i.e. try to put 2-3 pieces back into place and then worry about the remaining 1-2 pieces. Generally, the move count for average level solvers is around 6-10 steps. The solution produced here is only 5 moves long. Finding 5 move cross solution intuitively is not very rare. In fact, it depends mostly on how hard the scramble sequence is. The rotations in the move sequence even though very efficient are still interpretable. The sequence is trying to align each edge piece with one of its corresponding centers and once all 4 edges are aligned in this way, they are slotted back into place completing the cross. The cross sequence move count for the 10 different scrambles varied from 5 to 7 as shown in the table above with an average move count of 5.7. Considering that this move count is quite common for solvers of intermediate to advanced level and that the sequences do tend to follow human intuition, it can be stated that the IDA\* suggests human interpretable moves for the cross stage in CFOP method.

F2L pair completion using IDA\* produces a move count which is consistently less than what a solver may find during their solves. One of the reasons for this is that the F2L stage of the solve happens after the cross and the solver does not have time to plan it. It has to happen on the fly based on fixed patterns of intuitive moves. But besides that, the other difference is the approach suggested by IDA\*. The move sequences do not seem to mind breaking up the cross in order to put it back in a different manner so as to also incorporate a completed F2L pair. This approach of breaking the cross is common for speed solvers but not that common at the intermediate level. So, it can be said that the F2L solutions are interpretable at the advanced level or they are still a little obscure for intermediate solvers. Another thing to notice here is that the last F2L pair’s average move count is consistently higher than all other stages. This is obvious because at the last F2L pair, the freedom to move pieces around is the most restricted as most of the first two layers are complete and therefore, the search has to go deeper in order to find a sequence that completes the last pair while preserving the prior progress.

## Example Solve for Roux

After using the above scrambled cube, the following move sequences for each stage of the Roux method are generated:

1. Left 1x2x2 Block: Z2 X Ui R F U Bi
2. Left 1x2x3 Block: F2
3. Right 1x2x2 Block: R
4. Right 1x2x3 Block: R U Ri Ui r U Ri
5. Top Corners: Ui R U2 Ri Ui R U Ri Ui R U Ri Ui R Ui Ri
6. Final Edges: Ui Mi Ui M Ui M2 Ui M U2 M Ui E2 M E2 Mi

Applying the above moves in tandem will also solve the entire cube from its starting scrambled position. The total move count for the solution is 46.

Even for the Roux method, the suggested moves are the most optimal. This is again expected because the heuristic used is the same and is admissible. Also, each stage search is completed fairly quickly the as search tree depth did not cross 7 for this particular scramble. But perhaps the most important thing to notice here is that the second and third stage got completed in just 1 move for this scramble. In fact, the summary of the move counts for each stage across 10 scrambles shown in the table below also shows that the 2nd and 3rd stages have very low average move count. This is because, by design the Roux method allows for more freedom to pair pieces together. Specially the use of M slices helps to cut down the move count when joining different pieces together. The rigidness of keeping the centers at a fixed position in the CFOP method is not followed in Roux. By virtue of the slice moves, the centers also move around the cube. This provides liberty to pair the pieces to form blocks and therefore, the move count is expectantly low. Such low move counts are not common even for advanced solvers of the Roux method.

|  |  |  |  |
| --- | --- | --- | --- |
| Stage | Min | Max | Avg |
| 1st 1x2x2 | 3 | 6 | 4.1 |
| 1st 1x2x3 | 1 | 4 | 2.8 |
| 2nd 1x2x2 | 1 | 6 | 3.8 |
| 2nd 1x2x3 | 4 | 8 | 5.8 |

The intuitive interpretability of the move sequences suggested by IDA\* for the different stages of the Roux method is slightly less than that of CFOP method. Since there is more freedom to move pieces around from anywhere on the cube and build the blocks in any manner, it is hard to see how the move sequence took shape from an intuitive point of view. To be more specifically, it is hard to see the first couple of moves in the sequence. But the first couple of moves are in, then the rest of the steps do follow a natural continuation which can be understood and hence mimicked by human solvers. The reason why getting the first few moves right intuitively is again the fact that in this method, the blocks can be built from any direction and as such there are many options to consider and to determine which option will yield the most efficient block is hard for a human. But locking into one particular block (as determined by the IDA\* search), the rest is very intuitive.

## CFOP vs Roux

In terms of move optimality, Roux method has a clear advantage over CFOP. The search technique is able to find much smaller sequences of moves to build blocks in Roux compared to the stages of CFOP. This could be attributed to the fact that CFOP is more restrictive in the order of piecing things together which forces the move sequence to complete stages in a certain manner thus potentially overlooking shorted ways of getting closer to the solution.

In terms of interpretability of the move sequence in the suggested solution, CFOP method has the advantage over Roux. This because CFOP by being more rigid in its procedure allows for a simpler train of thought in the heads of the solvers and the search techniques by being restricted by this relatively rigid procedure also find solutions that make more intuitive sense.

## Advantages and Disadvantages of this Method

One of the major problems of using a search technique on a problem of this size is that as the tree grows, the number of nodes in the tree increase exponentially and very quickly, the tree becomes too much to handle specially for a single laptop computer. The search happens fairly quickly if somehow the number moves required to solve a particular problem is kept small i.e. limit the tree depth to a manageable number. That is where dividing the solution into small stages plays a very important role. Not only are the smaller stages easier to understand, but solution is produced quickly because the tree has a limited search space (on account of the fact that a smaller number of pieces are involved in each stage). But as we try to increase the complexity of a stage (by increasing the number of pieces to be tracked), the problem becomes large and hence difficult to manage. Therefore, using IDA\* we are only limited to smaller stages on a Rubik’s Cube. For example, some advanced solvers combine the cross and the first 2 F2L pairs into step. Similarly, many Roux solvers complete the 1x2x3 block in one go. Finding solutions to such stages is problematic specially with limited computing power. Basically, the current approach works well for small stages but is not scalable to more stages and therefore not applicable on methods that more intuitive like the Heise method or the Snyder method (where solvers try to fix more and more pieces together).

Another advantage of this technique is while the solutions are optimal, they still retain some intuitive understanding. As such the solvers can try to incorporate some of the patterns generated using this method into their own solves. For example, during the F2L stage, the move sequences generated by this method are more liberal in terms of breaking apart the cross in order to complete a pair. Similarly, during the Roux method, blocks are built from obscure angles which is generally not preferred by cubers as the moves required to build such blocks may not be ergonomically efficient. In order to take advantage of such obscure blocks, solvers can focus on examining the scrambled from different angles and not focus just on familiar angles like holding the white side down and looking to build blocks around it.

# Conclusion

In conclusion, the IDA\* algorithm using the modified Manhattan distance when applied separately on smaller stages of a complete Rubik’s Cube solving method produces optimal move sequences for the individual stages. The move sequences generated have for the most part decent amount of intuitive interpretability for humans and this technique could be used to find new interesting patterns that can help improve human solvers of the cube.

Since interpretability is the primary objective, this technique is more effective on restrictive methods like CFOP as compared to Roux method as the results produced for the former are easier to grasp from a human perspective.

Extending this method on more complex method is not plausible as with stages involving more pieces, the search tree can grow to intractable sizes and methods of reducing tree sizes using symmetries and patterns have not been incorporated.

## Future Work

This technique could be made more usable, scalable and flexible by including a more sophisticated mechanism of identifying states that are similar (in symmetry) to the states seen previously in the search space. Currently, each state with a different colour scheme is considered a different state and processed separately. But as Rubik’s Cube is a closed group, there is great potential in cutting the number of possible states by applying some principles of group theory. More precisely, the ideas used in finding the God’s Number can be incorporated to form a more capable solution.

The heuristic used in this project is admissible but it does not incorporate any strategic notions. Heuristics like calculating the number of oriented pieces in a state prove to be ineffective here but there is potential of finding other heuristics like this which focus on a strategy of solving the cube rather than simply counting in absolute terms how far a state is from being a solved stage.

Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

## Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

## Units

* Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive”.
* Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
* Do not mix complete spellings and abbreviations of units: “Wb/m2” or “webers per square meter”, not “webers/m2”. Spell out units when they appear in text: “. . . a few henries”, not “. . . a few H”.

Identify applicable funding agency here. If none, delete this text box.

* Use a zero before decimal points: “0.25”, not “.25”. Use “cm3”, not “cc”. (*bullet list*)

## Equations

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled.

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus ( / ), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

*a**b* 

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1)”, not “Eq. (1)” or “equation (1)”, except at the beginning of a sentence: “Equation (1) is . . .”

## Some Common Mistakes

* The word “data” is plural, not singular.
* The subscript for the permeability of vacuum **0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o”.
* In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
* A graph within a graph is an “inset”, not an “insert”. The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
* Do not use the word “essentially” to mean “approximately” or “effectively”.
* In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
* Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
* Do not confuse “imply” and “infer”.
* The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
* There is no period after the “et” in the Latin abbreviation “et al.”.
* The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [7].

# Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Authors and Affiliations

**The template is designed for, but not limited to, six authors.** A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

### For papers with more than six authors: Add author names horizontally, moving to a third row if needed for more than 8 authors.

### For papers with less than six authors: To change the default, adjust the template as follows.

#### Selection: Highlight all author and affiliation lines.

#### Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

#### Deletion: Delete the author and affiliation lines for the extra authors.

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

## Figures and Tables

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**