This paper aims to present an algorithm to tackle hypergraph colouring. Given that this is a an NP Complete problem, it is obvious that the worst case complexity of any such algorithm would be exponential. However, the worst cases for colouring hyper-graphs arise when there are too many vertices that can be coloured with the same colour and choices need to be made so as to find the least number of colours required (to determine chromatic number). Brute force methods, tend to check each combination one by one without prioritising any vertex over another. This section presents an algorithm that prioritises colouring vertices with higher degrees first. For instance, if a particular colour can be repeated for two vertices, brute force strategy suggests checking both cases one by one and completing rest of the colouring. Both the outcomes are analysed and then the better one is chosen. This has very high computational complexity. The presented algorithm sacrifices optimum colouring for colouring the hyper-graph in reasonable time.

The algorithm begins by sorting the given hypergraph vertices in descending order of degrees. A colour counter is set, indicating the number of colours used yet. We start off with number of colours as zero. We colour the first uncoloured vertex (according to sorted degree) with the colour ‘1’. We maintain a list of vertices with the same colour. Thereafter, we look for the next most connected vertex (in order) that is compatible with our current vertices with this colour that we have stored in the list mentioned above. Compatibility here can be defined as follows:

Vertex A = [1 0 1 0 1]
Vertex B = [0 1 1 0 0]

Here, both vertices are a part of the third edge. Hence, a colour used for A cannot be used for B. However, If the vertices were of the following nature:

Vertex A = [1 0 1 0 1]
Vertex B = [0 1 0 0 0]

None of the edges that contain A, also contain B. Therefore, the same colours may be used for both. Mathematically, this can be checked as,

Summation (Ea*Ba)

If this value is zero for a pair of vertices, then and only then can they be considered compatible. Now, once the algorithm comes across such a vertex that is compatible with all the vertices already assigned a particular colour, that colour is readily assigned to that vertex. This vertex is now added to the list maintaining the record of all the vertices with the same colour. Now the algorithm moves forward to check the next vertex. In case you reach the end of the hypergraph, and colouring has not been completed, a new colour ‘2’ will be taken up. The process is so repeated till all the vertices are coloured.

Since, each colour used up can iterate at most once over the hypergraph matrix, the traversal will always have complexity of the order O(colours). However, we know that the number of colours needed to colour a hypergraph will always be less

than or equal to the number of total vertices. Hence, this traversal will take complexity of the order O(n). Since we sort the hypergraph based on degrees in the beginning, the calculation of degrees requires an O(n^2) complexity.

Hence, the overall time complexity is of the order O(n^2).

In a few cases, the colouring returned by the algorithm may not be the optimal solution. For instance, hypergraph may have chromatic number as 10, while the algorithm might return 11. In such cases, the solution returned is still a proper solution to the hypergraph colouring problem, however, not the optimal solution.

For such cases, the following error function has been defined:

Xo denotes the chromatic number of a hypergraph.
X’ denotes the number of colours used for colouring by the algorithm.

Error = (X’-Xo)/Xo