An Implementation of Dsatur

TCSS 543: Advanced Algorithms
Jesse Bannon, Sruthi Jogi

In this document, we show our implementation of Dsatur, the heuristic graph coloring algorithm with an improved runtime than it's naive counterpart.

Preliminary

To improve the running time complexity of the naive $O(n^2)$ heuristic graph coloring algorithm Dsatur [1], we use a TreeMap as our primary data structure; a key-value data structure in the form of a self balancing sorted tree. In addition, we use a HashSet. Their runtime characters are described in Table I.

Table I: Runtime for TreeMap operations

Operation	CONTAINS	GET	PUT	REMOVE
TreeMap Runtime	O(log(n))	O(log(n))	O(log(n))	O(log(n))
HashSet Runtime	O(1)	O(1)	O(1)	O(1)

Our graph G = (V, E) is represented using an adjacency list of vertices with the characteristics shown in Figure I. Each vertex is initialized with a unique name, its respective edges, and the total number of vertices in the graph. Additionally, we keep track of |V| possible colors a vertex can be colored with using a TreeSet. We can remove a possible coloring choice or set any vertex's color in log|V| running time.

Figure I: Vertex characteristics

```
Class Vertex:
    Fields:
        Name: Char                                              # Unique name
        TotalVertices: Int                                      # Total vertices in graph; |V|
        AvailableColors: TreeSet[Int](1, 2, …, TotalVertices)   # Max of 1..TotalVertices colors
        Edges: Array[Vertex]                                    # Connected vertices to this vertex
        Color: Int = 0                                          # Color of graph, initially set to 0
    Methods:
        RemoveColor(color: Int)         = AvailableColors.remove(color)                 # log|V| runtime
        MinColor()                      = return AvailableColors.min()                  # log|V| runtime
        SaturationDegree()              = return TotalVertices - AvailableColors.size()  # c runtime
        AdjacencyDegree()               = return Edges.length()                         # c runtime
```

We store all vertices in our nested data structure TMap, of type TreeMap[Int, HashSet[Vertex]]. The key (Int) represents the saturation degree. The value (HashSet[Vertex]) contains all vertices with that saturation degree, hashed by their unique name. We know the max saturation degree is |V|, and is possible for every vertex to have the same saturation degree. This implies our TMap has the following runtime characteristics.

Table II: Runtime for TMap operations

Operation	CONTAINS	GET	PUT	REMOVE
Runtime	O(log(|V|))	O(log(|V|))	O(log(|V|))	O(log(|V|))

Algorithm

Our algorithm is described using the following psuedo-code in Figure II.

Figure II: Psuedo-code for our Dsatur implementation

```
Dsatur(vertices: Array[Vertex]):
        TMap ← TreeMap[Int, HashSet[Vertex]]
        firstVertex ← vertex FROM vertices WITH MAX AdjacencyDegree()
        firstVertex.color ← firstVertex.MinColor()
        FOR ve IN firstVertex.Edges:
                ve.removeColor(firstVertex.color)
        ENDFOR

        TMap[0] ← v IN vertices WHERE saturationDegree = 0 AND color = 0
        TMap[1] ← v IN vertices WHERE saturationDegree = 1 AND color = 0

        WHILE TMap.size() > 0:
                maxSat ← REMOVE vertex FROM TMap WITH MAX SaturationDegree()
                maxSat.color ← maxSat.MinColor()

                FOR ve IN maxSat.Edges:
                        IF TMap CONTAINS ve:
                                REMOVE ve FROM TMap      # Remove from old saturation degree bucket
                                ve.remove(maxSat.color)  # Update saturation degree
                                ADD ve TO TMap           # Add to new saturation degree bucket
                        ENDIF
                ENDFOR
        ENDWHILE
        RETURN vertices
ENDFUNC
```

We can color the first vertex with greatest adjacency degree and update its edges available colors in |V| + log|V| + |E|log|V| time. Adding all other vertices into TMap takes 2|V| time. Within the while loop, we perform a REMOVE from TMap and color the vertex with max saturation with its minimum available color in 2log|V| time. For each edge of the colored vertex, if its contained in TMap, we remove from TMap, update its available colors, and add back into TMap for a total of 3log|V| time. The worst case number of edges is |E|, making the for-loop runtime a total of 3|E|log|V|. This is repeated |V|-1 times, making the while-loop runtime a total of (|V|-1)(3|E|log|V|).

Thus, our total runtime is |V| + log|V| + |E|log|V| + 2|V| + (|V|-1)(3|E|log|V|) = O(|V|log|V||E|).

Experimental Results

We performed 100 trials for each configuration of {100, 200, …, 1000} X {5, 10, …, 95} vertices and edge density percentage, respectively, and averaged the runtimes in seconds. For a fixed number of vertices, there is roughly linear increase in runtime with respect to edge density, as expected with |E| in the runtime. With fixed edge densities, the runtime is greater than linear, but less than parabolic when increasing vertices. Hence |V|log|V| in its running time complexity.

Figure III: Dsatur runtimes with edge density-centered axis

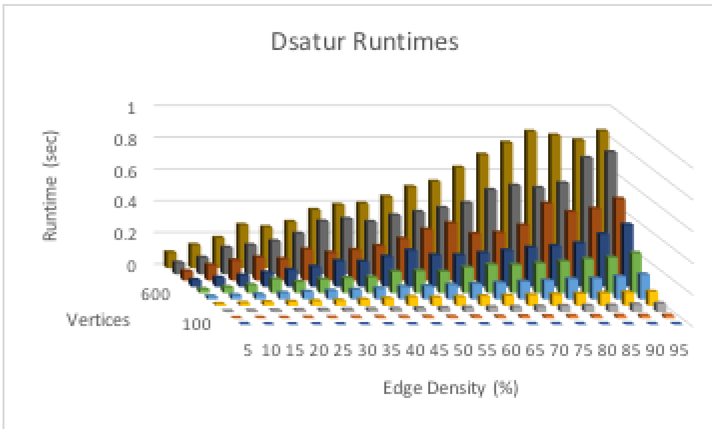
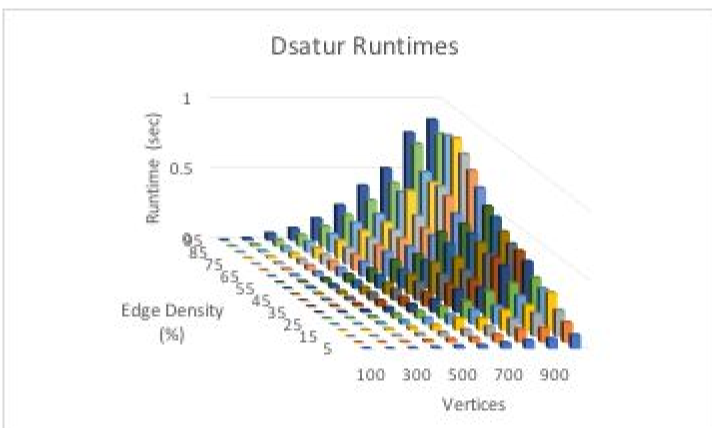


Figure IV: Dsatur runtimes with vertices-centered axis



In addition, we collected the average number of colors used for each configuration. We observe that for a fixed number of vertices, there is parabolic behavior when increasing edge density. The amount would roughly double going from 95% to 100%. With fixed edge density, the number of colors increasing linearly with respect to the number of vertices.

Figure V: Dsatur average color counts with edge density-centered axis

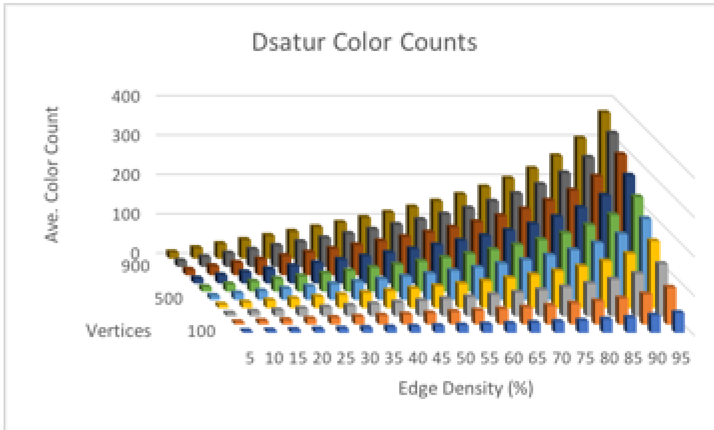
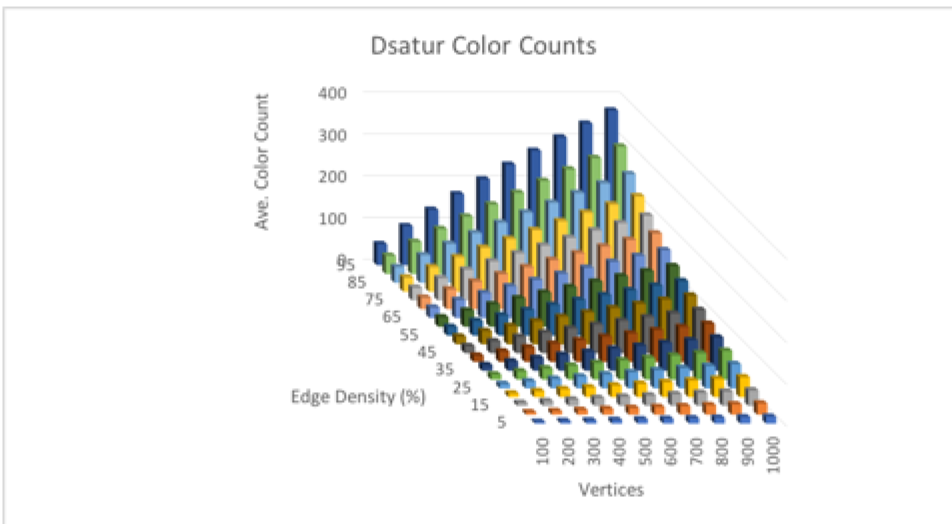


Figure VI: Dsatur average color counts with vertices-centered axis



Conclusion

In this paper, we have shown our implementation of Dsatur with O(|V|log|V||E|) runtime. We perform experiments using edge densities and vertices as parameters to show runtime and color count behavior.

References

[1] Brélaz, Daniel. "New methods to color the vertices of a graph." *Communications of the ACM* 22.4 (1979): 251-256.