CN-Exercise2

May 2, 2016

```
In [1]: import igraph

        g_dolphins = igraph.read("networks\\real\\dolphins.net",format="pajek")
        part_dolphins = igraph.read("networks\\real\\dolphins-real.clu",format="pajek")
        g_football = igraph.read("networks\\real\\football.net",format="pajek")
        part_football = igraph.read("networks\\real\\football-conferences.clu",format="pajek")
        g_zachary = igraph.read("networks\\real\\zachary_unwh.net",format="pajek")
        part_zachary = igraph.read("networks\\real\\zachary_unwh-real.clu",format="pajek")
```

Complex Networks Exercise 2 - Community Detection

Cole MacLean - May 1, 2016

Introduction

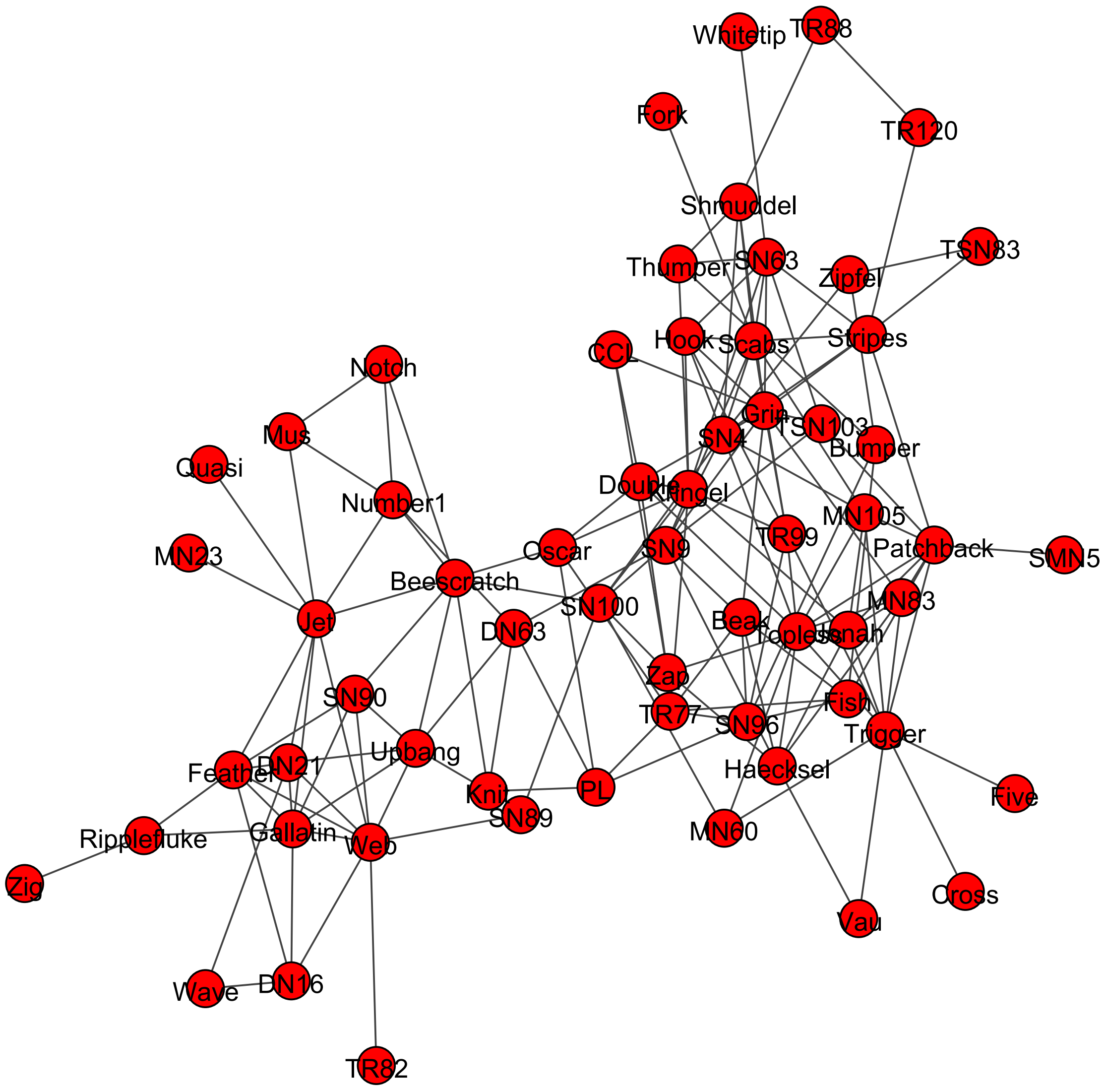
In this analysis, we review various community detection algorithms implemented in the igraph python library. Using the community visualizations, we can comment on the similarities and differences algorithm results.

Networks

In this analysis, we will review 3 real-world networks. These network structures are visualized below.

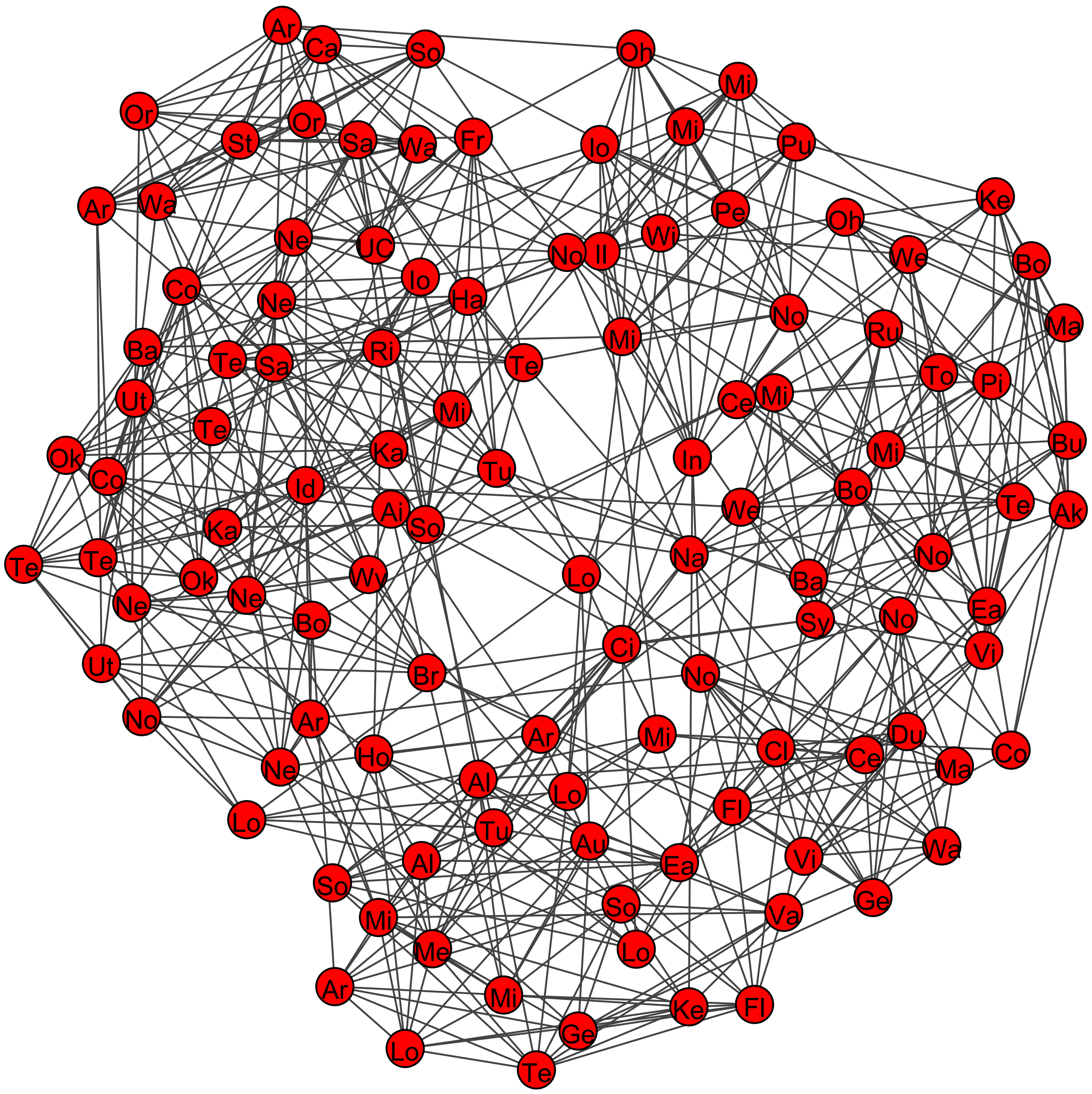
```
In [2]: dolphin_layout = g_dolphins.layout("kk")
        igraph.plot(g_dolphins, layout = dolphin_layout,vertex_label=[vertex['id'] for vertex in g_dolpl
```

```
Out[2]:
```



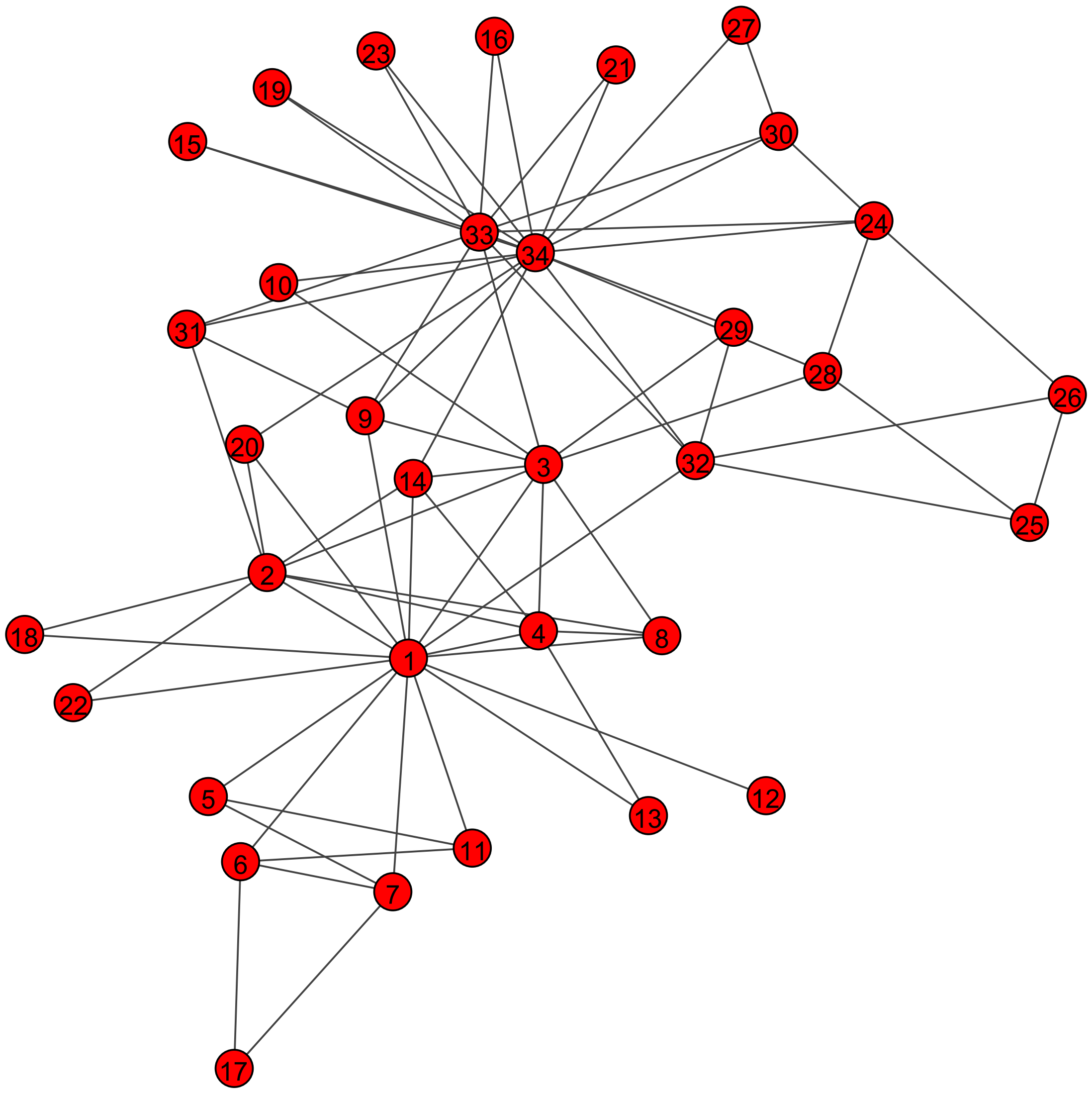
```
In [26]: football_layout = g_football.layout("kk")
         igraph.plot(g_football, layout = football_layout,vertex_label=[vertex['id'][0:2] for vertex in
```

```
Out[26]:
```



```
In [32]: zachary_layout = g_zachary.layout("kk")
         igraph.plot(g_zachary, layout = zachary_layout,vertex_label=[vertex['id'] for vertex in g_zacha
```

```
Out[32]:
```



community_fastgreedy

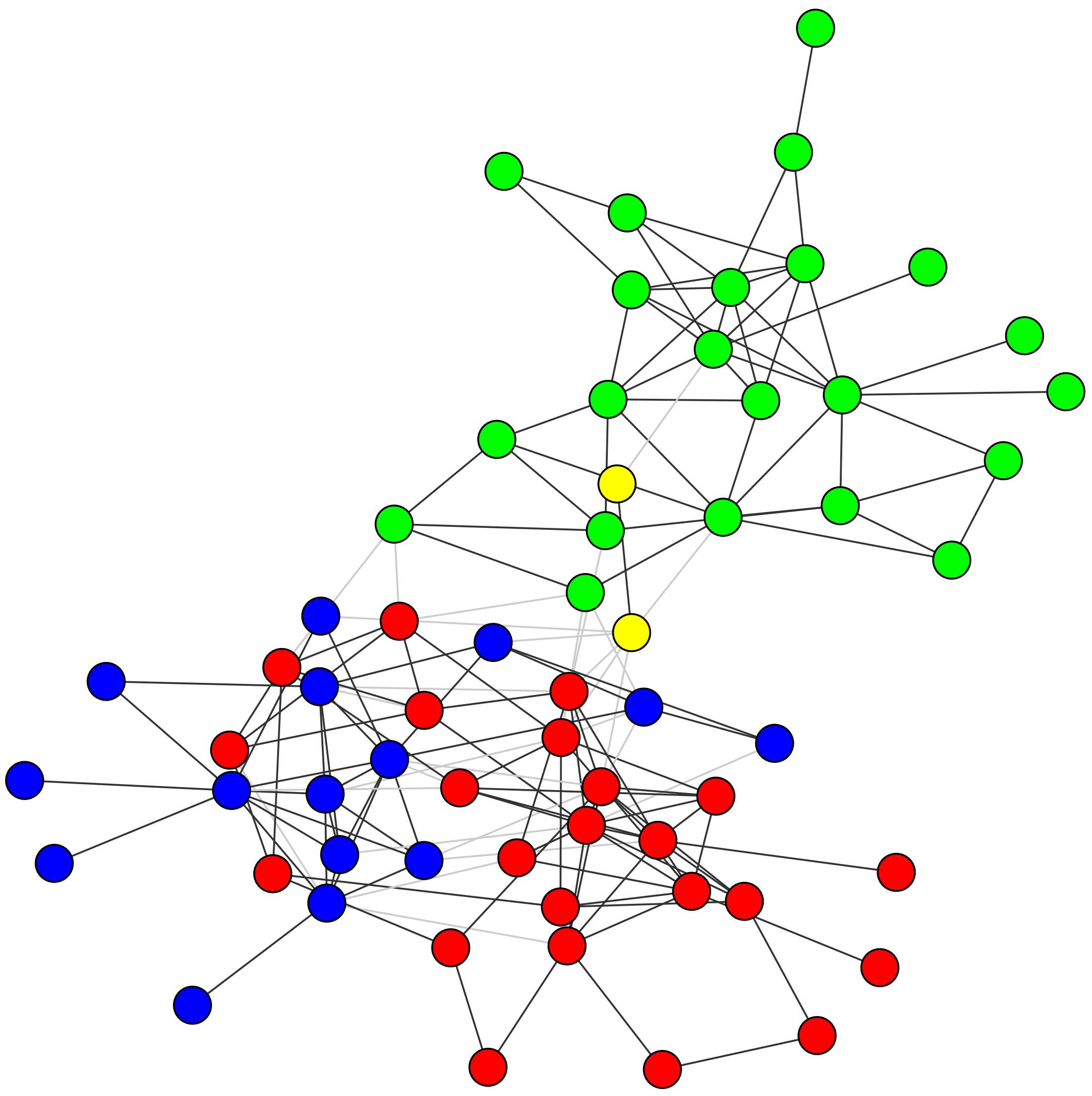
The first algorithm utilized for community detection is the "community_fastgreedy" algorithm implemented in igraph, which uses modularity as the measure of optimal community seperation in a graph. From the igraph documentation, the algorithm:

"Finds the community structure of the graph according to the algorithm of Clauset et al based on the greedy optimization of modularity. This is a bottom-up algorithm: initially every vertex belongs to a separate community, and communities are merged one by one. In every step, the two communities being merged are the ones which result in the maximal increase in modularity"

```
In [29]: fastgreedy_dolphins = g_dolphins.community_fastgreedy()
         fastgreedy_football = g_football.community_fastgreedy()
         fastgreedy_zachary = g_zachary.community_fastgreedy()
```

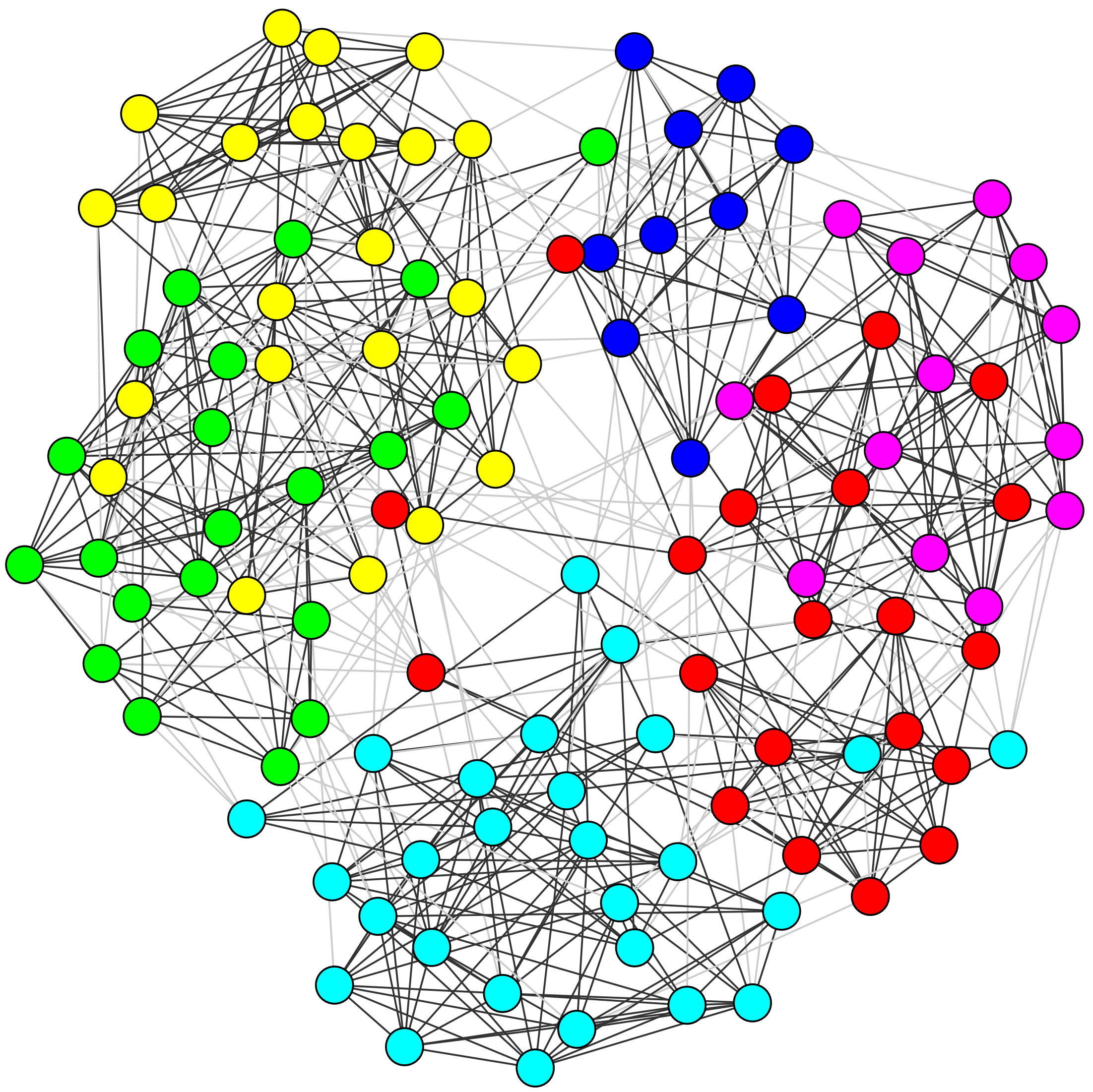
```
In [28]: igraph.plot(fastgreedy_dolphins.as_clustering(),layout=dolphin_layout)
```

```
Out[28]:
```



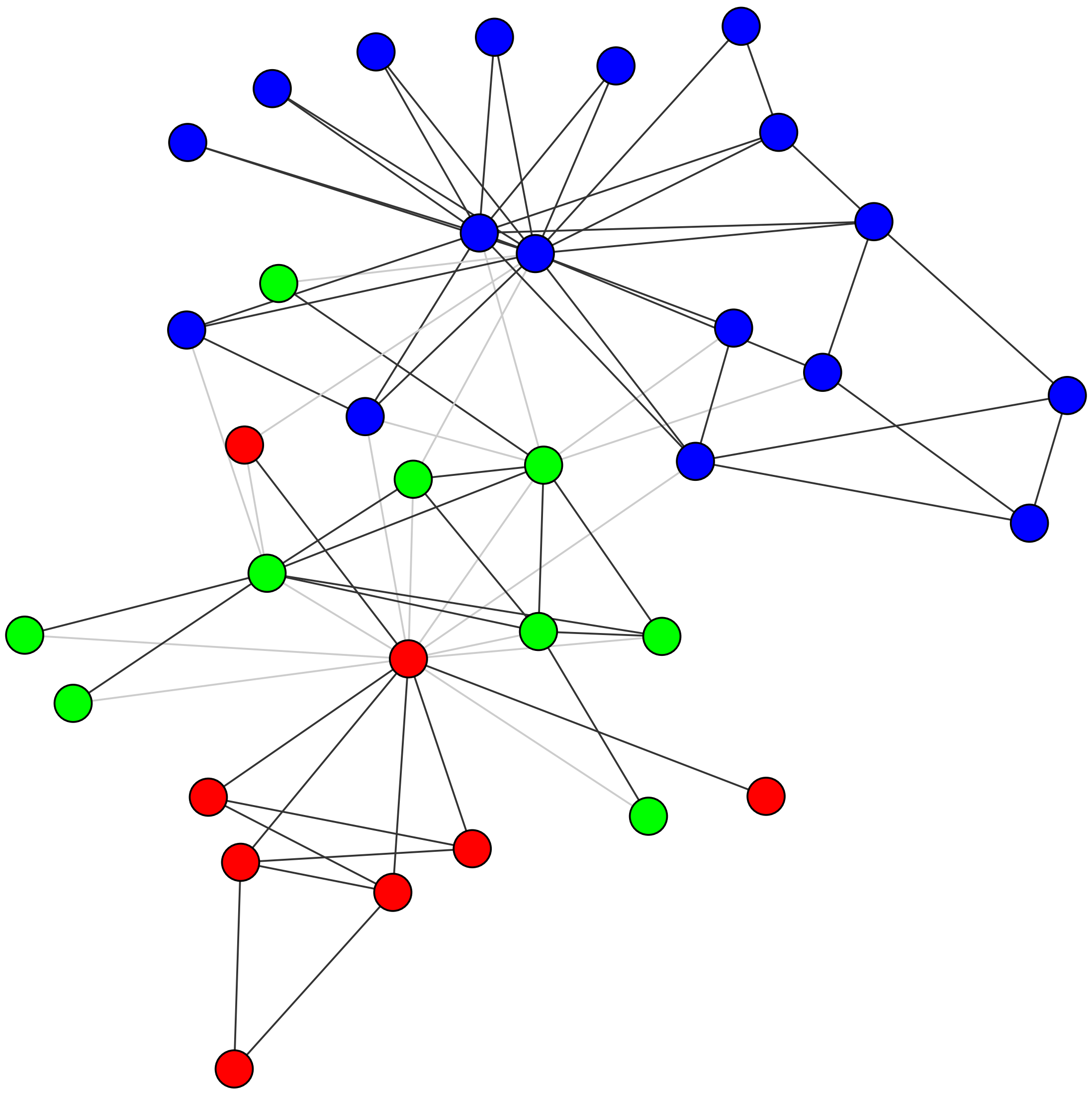
```
In [30]: igraph.plot(fastgreedy_football.as_clustering(),layout=football_layout)

Out[30]:
```



```
In [33]: igraph.plot(fastgreedy_zachary.as_clustering(),layout=zachary_layout)

Out[33]:
```



community_walktrap

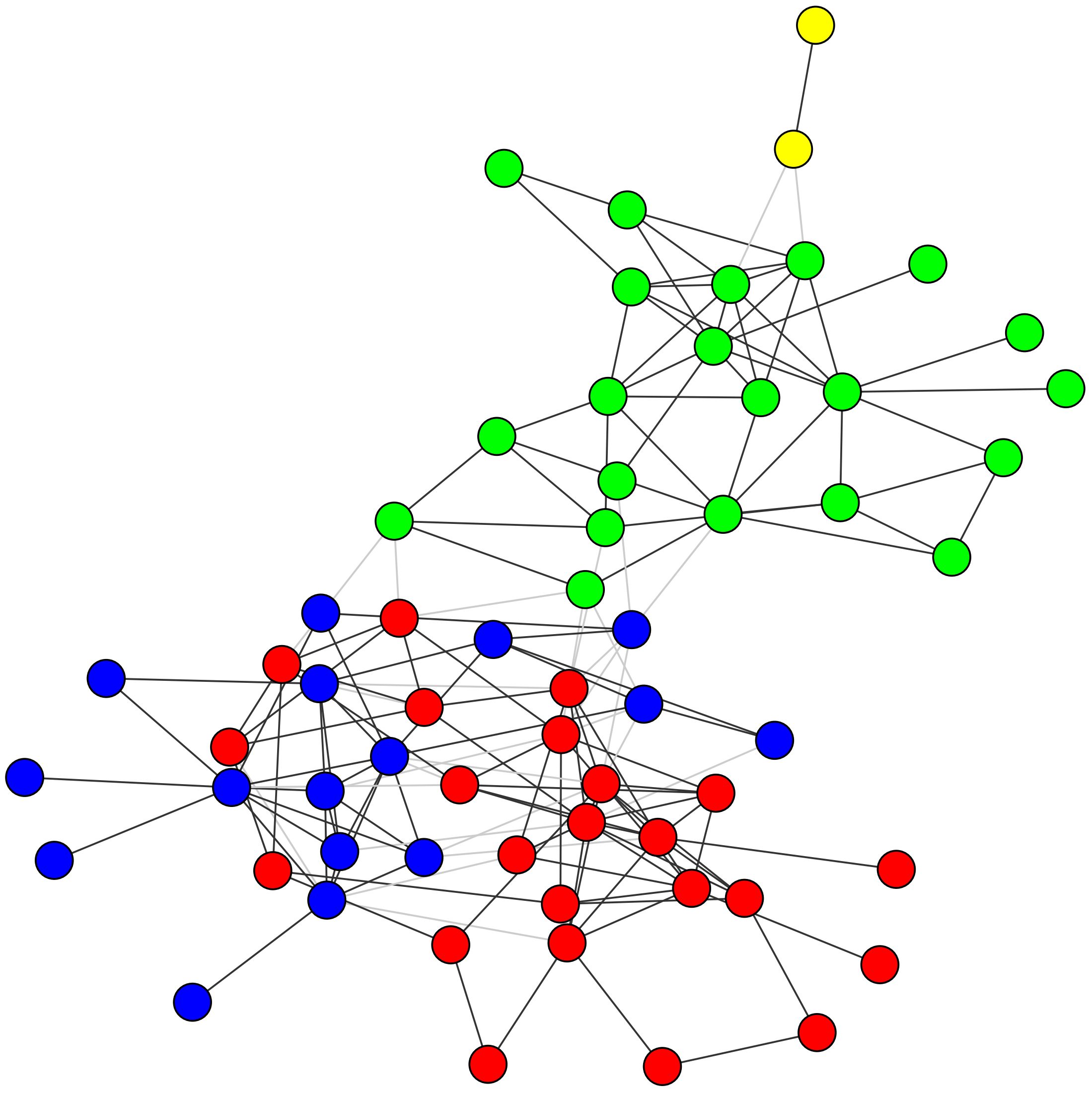
Another community detection algorithm implemented in igraph is the community_walktrap algorithm, which uses the assumption that an agent randomly walking along the edges of a network will fall into and become trapped in the same community as other agents taking similiar length random walks. From the docs:

"Community detection algorithm of Latapy & Pons, based on random walks. The basic idea of the algorithm is that short random walks tend to stay in the same community. The result of the clustering will be represented as a dendrogram."

```
In [34]: walktrap_dolphins = g_dolphins.community_walktrap()
         walktrap_football = g_football.community_walktrap()
         walktrap_zachary = g_zachary.community_walktrap()
```

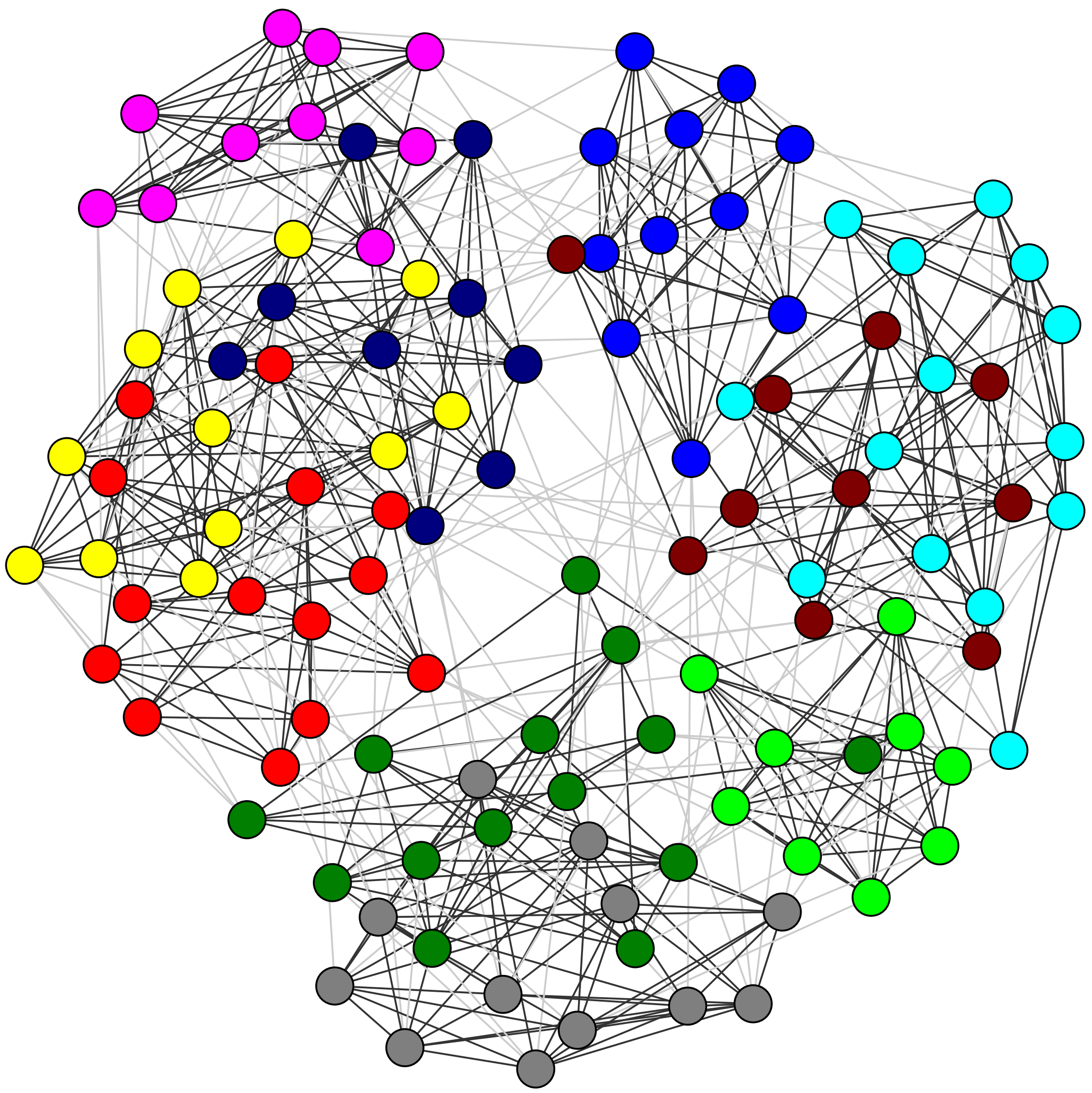
```
In [35]: igraph.plot(walktrap_dolphins.as_clustering(),layout=dolphin_layout)
```

```
Out[35]:
```



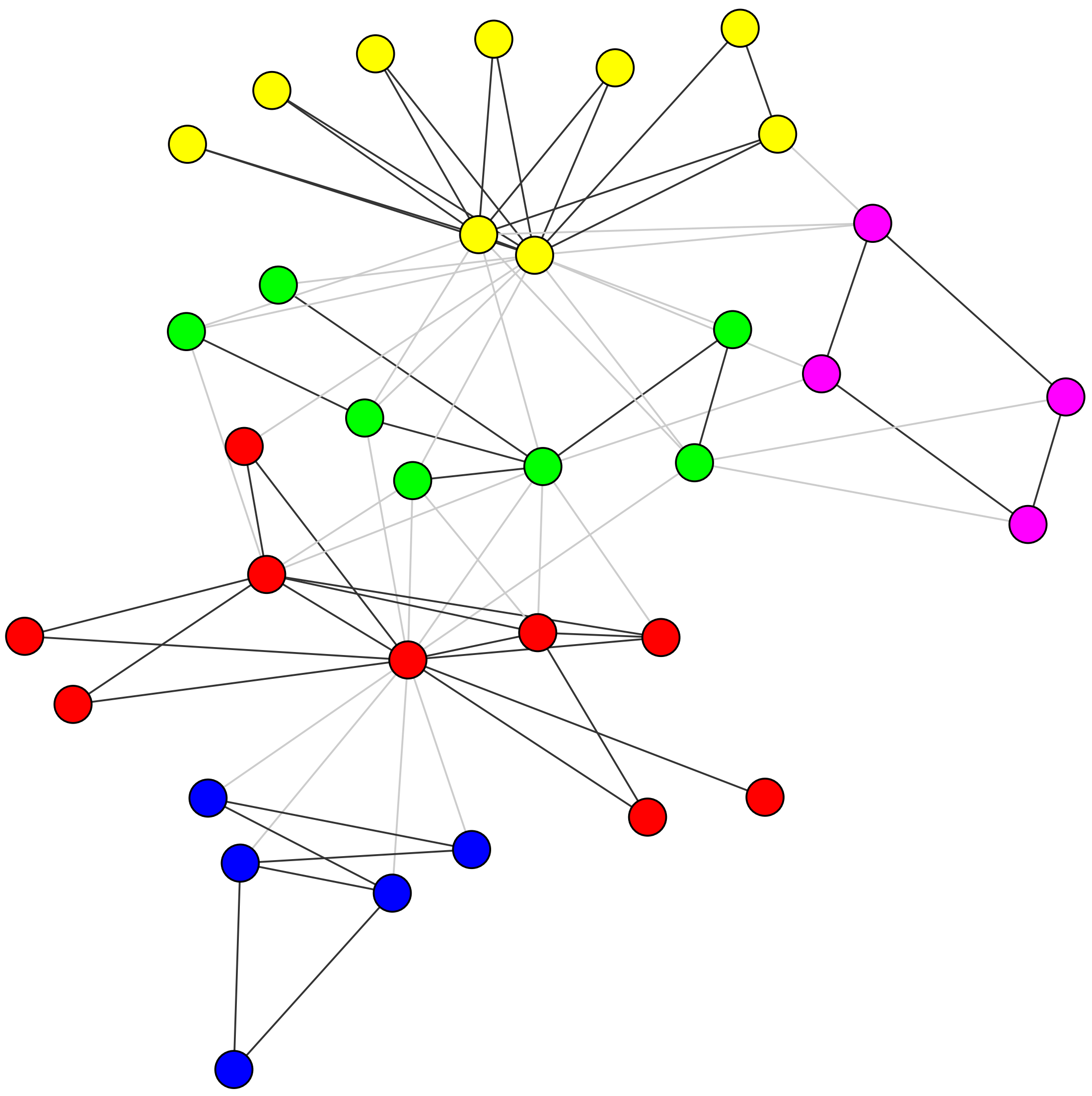
```
In [36]: igraph.plot(walktrap_football.as_clustering(),layout=football_layout)
```

```
Out[36]:
```



```
In [38]: igraph.plot(walktrap_zachary.as_clustering(),layout=zachary_layout)

Out[38]:
```



community_leading_eigenvector

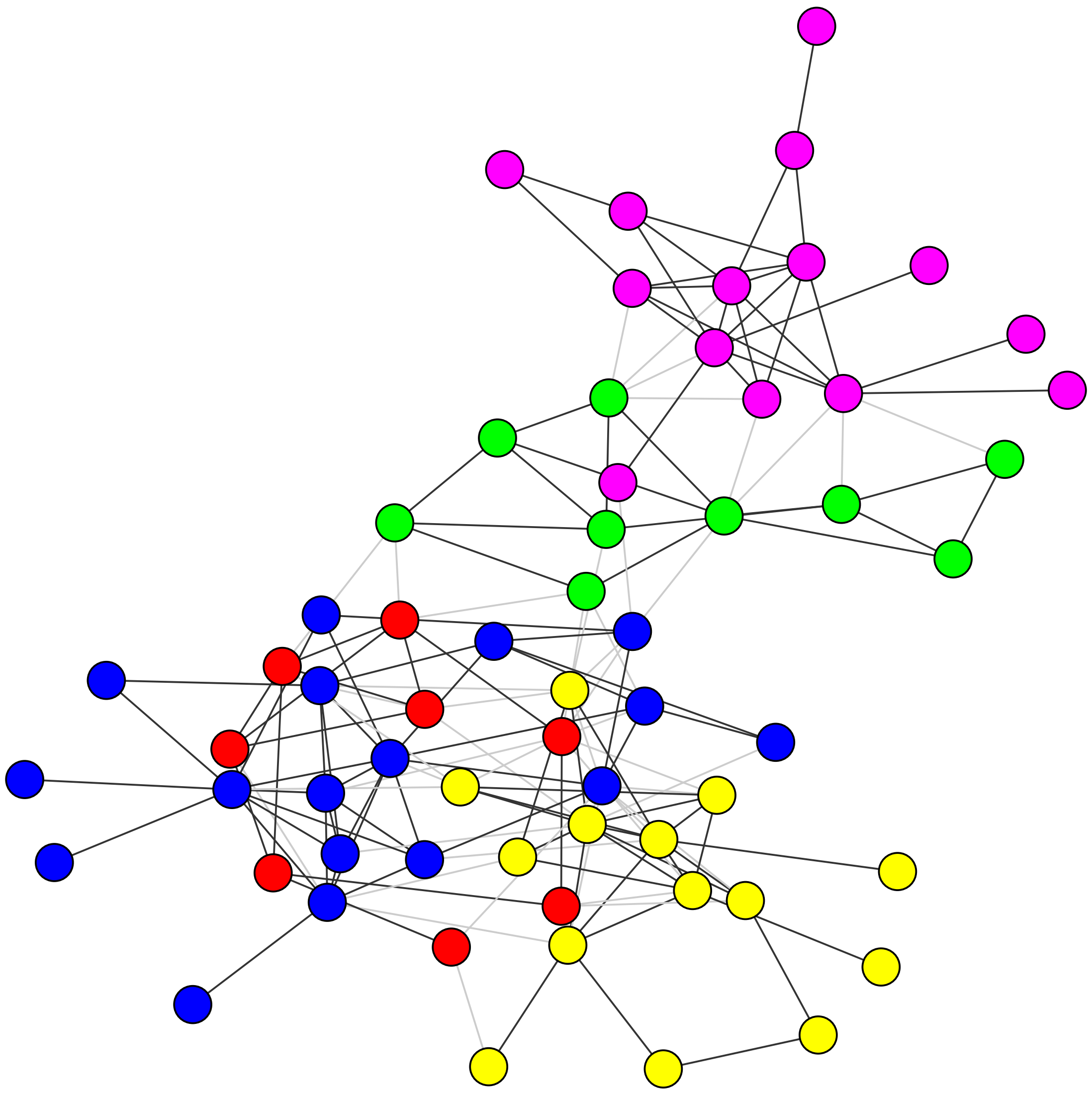
Another community detection algorithm implemented in igraph is the community_leading_eigenvector algorithm, which is another modularity maximization algorithm. From the docs:

"Newman's leading eigenvector method for detecting community structure. This is the proper implementation of the recursive, divisive algorithm: each split is done by maximizing the modularity regarding the original network."

```
In [39]: eigen_dolphins = g_dolphins.community_leading_eigenvector()
         eigen_football = g_football.community_leading_eigenvector()
         eigen_zachary = g_zachary.community_leading_eigenvector()
```

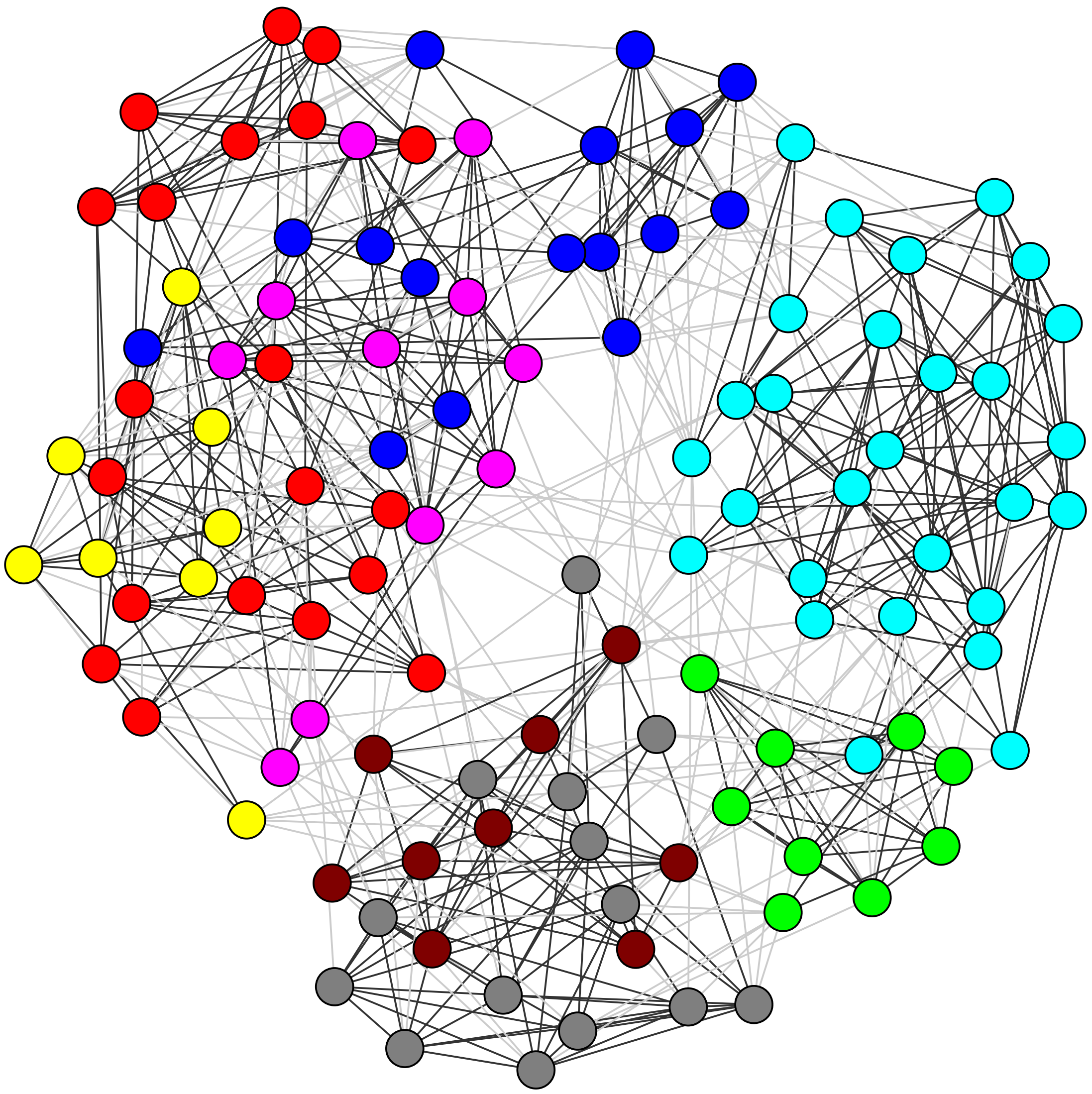
```
In [41]: igraph.plot(eigen_dolphins,layout=dolphin_layout)
```

```
Out[41]:
```



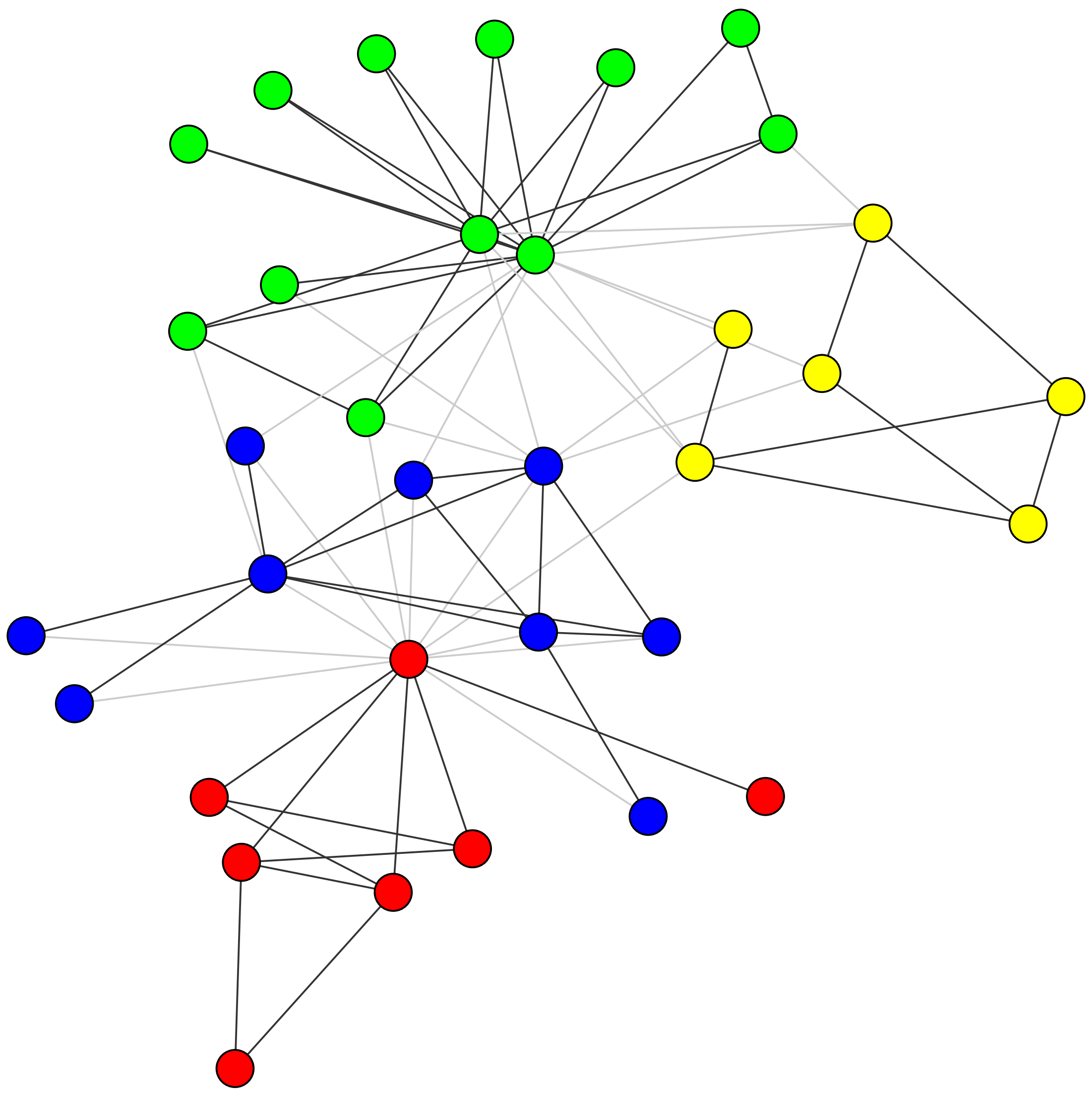
```
In [42]: igraph.plot(eigen_football,layout=football_layout)

Out[42]:
```



```
In [44]: igraph.plot(eigen_zachary,layout=zachary_layout)

Out[44]:
```



Results

Reviewing the different community visualizations, we can see that different algorithms can produce dramatically different community representations for the same network. Both the fastgreedy and and leading eigen vector algorithms produce similiar community count and partitions, where the walktrap algorithm appears to consistently generate additional communities beyond what the other 2 algorithms discover. These results make sense, as both the fastgreedy and eigen vector algorithms are founded on the concept of maximizing the modularity of a network, where the walktrap algorithm only depends on the notion of agents becoming trapped within communities over the course of short random walks.

This analysis outlines the differences in community detection implementation and interpretation, and further supports the argument that some domain knowledge about a network is required in order to interpret it. Especially when considering community detection, the user of these algorithms needs to remember that communities can be found, even when no communities, in reality, exist.