COMMUNITY DETECTION

In [170]:
```python
import pandas as pd
import igraph as ig
import matplotlib.pyplot as plt
from matplotlib import gridspec
import networkx as nx
import os
import dataframe_image as dfi
from copy import deepcopy

from sklearn import metrics
from sklearn.metrics.cluster import contingency_matrix, pair_confusion_matrix

from networkx.algorithms import community
import networkx.algorithms.community as nx_comm

from collections import defaultdict


%matplotlib inline
```

Useful functions

In [2]:
```python
#functions to read the files given

def read_clu(path, output_type):
    with open(path, 'r') as f:
        lines = f.readlines()

    if output_type == 'membership':
        return membership_list(lines)

    if output_type == 'clusters':
        return clusters_list(lines)

def membership_list(lines):
    #returns a list of membership (to a cluster) for the nodes (lines)

    membership_list=[]

    if isinstance(lines, str):
        lines = iter(lines.split('\n'))
    lines = iter([line.rstrip('\n') for line in lines])

    while lines:
        try:
            l = next(lines)
        except:  # EOF
            break
        if l.lower().startswith("*vertices"):
            l, nnodes = l.split()
            for vertice in range(int(nnodes)):
                l = next(lines)
                membership_list.append(int(l))
        else:
            break

    return membership_list

def clusters_list(lines):
    #return a list of the clusters

    if isinstance(lines, str):
        lines = iter(lines.split('\n'))
    lines = iter([line.rstrip('\n') for line in lines])

    while lines:
        try:
            l = next(lines)
        except:  # EOF
            break
        if l.lower().startswith("*vertices"):
            l, nnodes = l.split()
            communities = defaultdict(list)
            for vertice in range(int(nnodes)):
                l = next(lines)
                community = int(l)
                communities.setdefault(community, []).append(vertice)
        else:
            break

    return [ set(v) for k,v in dict(communities).items() ]
```

In [54]:
```python
def clusters_to_membership(c):
    #converts a list of clusters into a membership list
    membership_list=[]
    n=0
    clusters=[sorted(list(s)) for s in c]
    for i,cluster in enumerate(clusters):
        for node in range(len(cluster)):
            membership_list.append(i)
    return membership_list


def write_clu(membership_list,name):
    #to store a membership list as a .clu file
    textfile = open('clu_files/'+name+'.clu', "w")
    textfile.write('*Vertices {} \n'.format(len(membership_list)))
    for element in membership_list:
        textfile.write(str(element) + "\n")
    textfile.close()
```

In [3]:
```python
#import and store all the networks

nx_nets={}
ig_nets={}
clu ={} #membership
clu_clusters = {}
clu_names=[]
nets_folder_path="A3-networks/"
with os.scandir(nets_folder_path) as folders_list:
    for folder in folders_list:
        if folder.is_dir():
            networks=os.listdir(nets_folder_path+folder.name)
            for net in networks:
                if net.endswith(".net"):
                    file=nets_folder_path+folder.name+'/'+net
                    nx_nets[('{}'.format(net[:-4]))]=nx.Graph(nx.read_pajek(file))
                    ig_nets[('{}'.format(net[:-4]))]=ig.read(file,format="pajek").simplify()
                elif net.endswith(".clu"):
                    clu_names.append(net[:-4])
                    file=nets_folder_path+folder.name+'/'+net
                    clu[('{}'.format(net[:-4]))]= read_clu(file, 'membership')
                    clu_clusters[('{}'.format(net[:-4]))]= read_clu(file, 'clusters')
```

In [104]:
```python
#keys_to_clu={'graph3+1+3':'graph3+1+3', '20x2+5x2':'20x2+5x2','graph4+4':'graph4+4',
#              'star':'star', 'cat_cortex_sim':'cat_cortex_sim',
#              'zachary_unwh':'zachary_unwh-real', 'dolphins':'dolphins-real',
#              'football':'football-conferences','256_4_4_4_13_18_p':'256_4_4_4_13_18_p',
#              'rb125':'rb125-1', '256_4_4_2_15_18_p':'256_4_4_2_15_18_p'
#             }
#rb125 can be sent also to 'rb125-2' or 'rb125-3'

clu_to_nets={'graph3+1+3': 'graph3+1+3',
             '20x2+5x2': '20x2+5x2',
             'graph4+4': 'graph4+4',
             'star': 'star',
             'cat_cortex_sim': 'cat_cortex_sim',
             'zachary_unwh-real': 'zachary_unwh',
             'dolphins-real': 'dolphins',
             'football-conferences': 'football',
             '256_4_4_4_13_18_p': '256_4_4_4_13_18_p',
             'rb125-1': 'rb125',
             'rb125-2': 'rb125',
             'rb125-3': 'rb125',
             '256_4_4_2_15_18_p': '256_4_4_2_15_18_p'}
```

Clustering algorithms

In [250]:
```python
def comm_fastgreedy(name):

    f=deepcopy(ig_nets[name])
    v=f.community_fastgreedy()
    clusters = v.as_clustering()
    return clusters

def comm_optimal_modularity(name):
    f=deepcopy(ig_nets[name])
    v=f.community_optimal_modularity()
    return v

def comm_leading_eigenvector(name):
    f=deepcopy(ig_nets[name])
    v=f.community_leading_eigenvector()
    return v

def comm_label_propagation(name):
    f=deepcopy(ig_nets[name])
    v=f.community_label_propagation()
    return v

def plot_color_coded(name, membership, target=None):
    f=deepcopy(ig_nets[name])
    pal=ig.drawing.colors.ClusterColoringPalette(len(membership))
    f.vs['color']=pal.get_many(membership)
    if target!=None:
        if 'x' in ig_nets[name].vs.attributes():
            ig.plot(f, bbox=(0, 0, 400, 200), vertex_size=5, edge_width=0.5, target=target)
        else:
            ig.plot(f, bbox=(0, 0, 400, 200), layout='kamada_kawai',
                    vertex_size=5, edge_width=0.5, target=target)
    else: ig.plot(f, bbox=(0, 0, 400, 200), layout='kamada_kawai',
                    vertex_size=5, edge_width=0.5)
```

In [276]:
```python
def fastgreedy(labels_true, name):
    labels_pred=comm_fastgreedy(name).membership
    write_clu(labels_pred,name+'-fastgreedy')
    return ['fastgreedy',
            jaccard_index(labels_true,labels_pred),
            ig.compare_communities(labels_true,labels_pred, method='nmi'),
            normalized_vi(labels_true,labels_pred)]

def optimal_modularity(labels_true, name):
    labels_pred=comm_optimal_modularity(name).membership
    write_clu(labels_pred,name+'-optimal_modularity')
    return ['optimal_modularity',
            jaccard_index(labels_true,labels_pred),
            ig.compare_communities(labels_true,labels_pred, method='nmi'),
            normalized_vi(labels_true,labels_pred)]

def leading_eigenvector(labels_true, name):
    labels_pred=comm_leading_eigenvector(name).membership
    write_clu(labels_pred,name+'-lead_eig')
    return ['leading_eigenvector',
            jaccard_index(labels_true,labels_pred),
            ig.compare_communities(labels_true,labels_pred, method='nmi'),
            normalized_vi(labels_true,labels_pred)]

def label_prop(labels_true, name):
    labels_pred= comm_label_propagation(name).membership
    write_clu(labels_pred,name+'-label_prop')
    return ['label_propagation',
            jaccard_index(labels_true,labels_pred),
            ig.compare_communities(labels_true,labels_pred, method='nmi'),
            normalized_vi(labels_true,labels_pred)]
```

In [301]:
```python
def jaccard_index(C1,C2):
    #C1, C2 are membership lists of the clusters
    C=pair_confusion_matrix(C1,C2)
    den=(C[1,1]+C[1,0]+C[0,1])
    if den==0:
        return 'inf'
    index=C[1,1]/den
    return index

from math import log

def normalized_vi(C1, C2):
    #C1, C2 are membership lists of the clusters
    X=list(ig.Clustering(C1))
    Y=list(ig.Clustering(C2))
    n = float(sum([len(x) for x in X]))
    S = 0.0
    for x in X:
        p = len(x) / n
        for y in Y:
            q = len(y) / n
            r = len(set(x) & set(y)) / n
            if r>0.0:
                S += r * (log(r / p, 2) + log(r / q, 2))
    return abs(S)/log(n)
```

Comparison measures

In [305]:
```python
comparison={}
for clu_name in clu_to_nets:
    name=clu_to_nets[clu_name]
    comparison[name] = pd.DataFrame(columns=('algorithm', 'Jaccard', 'NMI', 'NVI' ))

    labels_true=clu[clu_name]
    comparison[name].loc[0]=fastgreedy(labels_true, name)
    comparison[name].loc[1]=leading_eigenvector(labels_true, name)
    comparison[name].loc[2]=label_prop(labels_true, name)
    #comparison[name].loc[3]=optimal_modularity(labels_true, name)
```

In [306]:
```python
for name in comparison:
    print('')
    print(name)
    print(comparison[name])
    dfi.export(comparison[name].style.set_caption(name),'comparison_measures/'+name+'.png')
```

```
graph3+1+3
             algorithm   Jaccard      NMI       NVI
0           fastgreedy  0.666667  0.80954  0.238237
1  leading_eigenvector  0.666667  0.80954  0.238237
2    label_propagation  0.666667  0.80954  0.238237

20x2+5x2
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.941176  0.938345  0.051124
1  leading_eigenvector  0.941176  0.938345  0.051124
2    label_propagation  1.000000  1.000000  0.000000

graph4+4
             algorithm  Jaccard  NMI  NVI
0           fastgreedy      1.0  1.0  0.0
1  leading_eigenvector      1.0  1.0  0.0
2    label_propagation      1.0  1.0  0.0

star
             algorithm  Jaccard  NMI  NVI
0           fastgreedy      1.0  1.0  0.0
1  leading_eigenvector      1.0  1.0  0.0
2    label_propagation      1.0  1.0  0.0

cat_cortex_sim
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.542169  0.656873  0.296602
1  leading_eigenvector  0.547872  0.618651  0.332598
2    label_propagation  0.257239  0.000000  0.481994

zachary_unwh
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.683274  0.692467  0.217697
1  leading_eigenvector  0.505495  0.677092  0.269804
2    label_propagation  0.462845  0.225967  0.351184

dolphins
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.504125  0.572700  0.271613
1  leading_eigenvector  0.329314  0.448914  0.423804
2    label_propagation  0.943044  0.888836  0.049311

football
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.362153  0.697732  0.385871
1  leading_eigenvector  0.350324  0.698670  0.407185
2    label_propagation  0.545143  0.848038  0.210807

256_4_4_4_13_18_p
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  1.000000  1.000000  0.000000
1  leading_eigenvector  1.000000  1.000000  0.000000
2    label_propagation  0.269841  0.680851  0.338132

rb125
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.281143  0.825443  0.287653
1  leading_eigenvector  0.031742  0.000000  0.967777
2    label_propagation  0.324100  0.887932  0.198671

256_4_4_2_15_18_p
             algorithm   Jaccard       NMI       NVI
0           fastgreedy  0.483871  0.869708  0.166303
1  leading_eigenvector  0.542431  0.924071  0.102676
2    label_propagation  1.000000  1.000000  0.000000
```

Modularity

In [69]:
```python
modularity_nets = pd.DataFrame(columns=('fastgreedy', 'leading_eig', 'label_prop'))

for name in ig_nets:
    modularity_nets.loc[name]=[comm_fastgreedy(name).modularity,
                               comm_leading_eigenvector(name).modularity,
                               comm_label_propagation(name).modularity]
```

In [121]:
```python
modularity_nets
```

Out[121]:

	fastgreedy	leading_eig	label_prop
graph3+1+3	0.367188	0.367188	0.367188
20x2+5x2	0.542579	0.542579	0.541586
graph4+4	0.423077	0.423077	0.423077
grid-p-6x6	0.401235	0.000000	0.277778
star	0.000000	0.000000	0.000000
cat_cortex_sim	0.260436	0.255355	0.000000
zachary_unwh	0.380671	0.393409	0.364809
dolphins	0.495491	0.491199	0.392073
airports_UW	0.662490	0.639231	0.516750
football	0.549741	0.492606	0.540977
256_4_4_4_13_18_p	0.696773	0.696773	0.664728
256_4_4_2_15_18_p	0.765660	0.752151	0.781804
rb125	0.608733	0.000000	0.583748

In [111]:
```python
modularity_clu = pd.DataFrame(columns=('modularity',))
for clu_name in clu_to_nets:
    name=clu_to_nets[clu_name]
    vc=ig.VertexClustering(ig_nets[name], membership=clu[clu_name])
    modularity_clu.loc[clu_name]=[vc.modularity]
```

In [112]:
```python
modularity_clu
```

Out[112]:

	modularity
graph3+1+3	0.351562
20x2+5x2	0.541586
graph4+4	0.423077
star	0.000000
cat_cortex_sim	0.245996
zachary_unwh-real	0.371466
dolphins-real	0.373482
football-conferences	0.553973
256_4_4_4_13_18_p	0.696773
rb125-1	0.600595
rb125-2	0.558144
rb125-3	0.553147
256_4_4_2_15_18_p	0.781804

In [310]:
```python
dfi.export(modularity_nets.style.set_caption('partitions found'),'modularity_nets.png')
dfi.export(modularity_clu.style.set_caption('reference'),'modularity_clu.png')
```

Color-coded plots

In [190]:
```python
def plots_alg(name):
    plot_color_coded(name, comm_fastgreedy(name).membership, 'color_plots/'+name+'-fastgreedy.png' )
    plot_color_coded(name, comm_label_propagation(name).membership, 'color_plots/'+name+'-label_prop.png')
    plot_color_coded(name, comm_leading_eigenvector(name).membership, 'color_plots/'+name+'-lead_eig.png')
```

In [251]:
```python
#store the plots in color_plots directory
for name in ig_nets:
    plots_alg(name)
```

NetworkX community detection

In [272]:
```python
from networkx.algorithms.community import greedy_modularity_communities, k_clique_communities, label_propagation_communities

def greedy_mod_nx(labels_true, name):
    c = greedy_modularity_communities(nx_nets[name])
    labels_pred=clusters_to_membership(c)
    write_clu(labels_pred,'nx_'+name+'-greedy_mod')
    return ['greedy_modularity',
            jaccard_index(labels_true,labels_pred),
            metrics.normalized_mutual_info_score(labels_true, labels_pred),
            normalized_vi(labels_true, labels_pred)]

#def k_clique_nx(labels_true, name, k):
#    c=k_clique_communities(nx_nets[name], k)
#    labels_pred=clusters_to_membership(c)
#    write_clu(labels_pred,'nx_'+name+'-k_clique')
#    return ['k-clique',
#            jaccard_index(labels_true,labels_pred),
#            metrics.normalized_mutual_info_score(labels_true, labels_pred),
#            normalized_vi(labels_true, labels_pred)]

def label_prop_nx(labels_true, name):
    c=label_propagation_communities(nx_nets[name])
    labels_pred=clusters_to_membership(c)
    write_clu(labels_pred,'nx_'+name+'-label_prop')
    return ['label_propagation',
            jaccard_index(labels_true,labels_pred),
            metrics.normalized_mutual_info_score(labels_true, labels_pred),
            normalized_vi(labels_true, labels_pred)]
```

In [307]:
```python
comparison_nx={}
for clu_name in clu_to_nets:
    name=clu_to_nets[clu_name]
    comparison_nx[name] = pd.DataFrame(columns=('algorithm', 'Jaccard', 'NMI', 'NVI' ))

    labels_true=clu[clu_name]
    comparison_nx[name].loc[0]=greedy_mod_nx(labels_true, name)
    #comparison_nx[name].loc[1]=k_clique_nx(labels_true, name, k)
    comparison_nx[name].loc[2]=label_prop_nx(labels_true, name)
```

In [308]:
```python
for name in comparison_nx:
    print('')
    print(name)
    print(comparison_nx[name])
    dfi.export(comparison_nx[name].style.set_caption(name),'comparison_measures/nx_'+name+'.png')
```

```
graph3+1+3
           algorithm   Jaccard      NMI       NVI
0  greedy_modularity  0.666667  0.80954  0.238237
2  label_propagation  0.666667  0.80954  0.238237

20x2+5x2
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.941176  0.938345  0.051124
2  label_propagation  1.000000  1.000000  0.000000

graph4+4
           algorithm  Jaccard  NMI  NVI
0  greedy_modularity      1.0  1.0  0.0
2  label_propagation      1.0  1.0  0.0

star
           algorithm  Jaccard  NMI  NVI
0  greedy_modularity      1.0  1.0  0.0
2  label_propagation      1.0  1.0  0.0

cat_cortex_sim
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.571930  0.702341  0.257299
2  label_propagation  0.257239  0.000000  0.481994

zachary_unwh
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.379009  0.289122  0.503219
2  label_propagation  0.403361  0.254134  0.494754

dolphins
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.245068  0.010911  0.628715
2  label_propagation  0.203363  0.041141  0.717362

football
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.058675  0.147975  1.086858
2  label_propagation  0.044610  0.238707  1.113253

256_4_4_4_13_18_p
           algorithm   Jaccard     NMI       NVI
0  greedy_modularity  1.000000  1.0000  0.000000
2  label_propagation  0.413333  0.6993  0.263123

rb125
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.281143  0.825443  0.287653
2  label_propagation  0.159132  0.754033  0.420127

256_4_4_2_15_18_p
           algorithm   Jaccard       NMI       NVI
0  greedy_modularity  0.461653  0.845852  0.196748
2  label_propagation  0.245902  0.773341  0.266577
```