# ZeroForcingReconfiguration

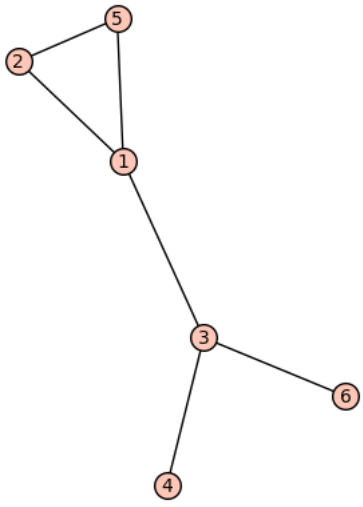
last edited Jun 3, 2023, 6:41:24 PM by hogben



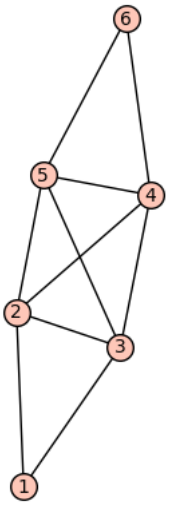
```
###################################
# This Sage worksheet is written by Bryan Curtis and Leslie Hogben hogben@aimath.org
# using code written by numerous other people.
# The focus of this worksheet is token addition and removal (TAR) reconfiguration graphs
# for zero forcing in support of the paper
# Isomorphisms  and  properties of TAR reconfiguration graphs for zero forcing and other $X$-set parameters
# by
# Novi H. Bong, Joshua Carlson, Bryan Curtis, Ruth Haas, and Leslie Hogben
#
# To run computations, you must first enter the function F- cells at the end of this file
###################################
```

```
###################################
# Example 3.1: Graphs G and H have same zero forcing polynomial but Z-TAR(G) is not isomorphic tp Z-TAR(H).
###################################
```

```
G=Graph("Epo_")
G.relabel([1,2,3,4,5,6])
show(G)
```




```
H=Graph("Ez[W")
H.relabel([1,2,3,4,5,6])
show(H)
```




```
print ZFS_poly(G), ZFS_poly(H)
```

```
[0, 0, 8, 13, 6, 1] [0, 0, 8, 13, 6, 1]
```

```
ZF_min_sets(G)
```

```
[{1, 2, 4},
 {1, 2, 6},
 {1, 4, 5},
 {1, 5, 6},
 {2, 4, 5},
 {2, 4, 6},
 {2, 5, 6},
 {4, 5, 6}]
```

```
ZF_min_sets(H)
```

```
[{1, 2, 4},
 {1, 2, 5},
 {1, 3, 4},
 {1, 3, 5},
 {2, 4, 6},
 {2, 5, 6},
 {3, 4, 6},
 {3, 5, 6},
 {2, 3, 4, 5}]
```

```
print zbar(G), zbar(H)
```

```
3 4
```

```
##############################################
# Data for Table 1
##############################################
```

```
nn=2
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb
```

```
1 1
```

```
n(1/1)
```

```
1.00000000000000
```

```
nn=3
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb
```

```
2 2
```

```
n(2/2)
```

```
1.00000000000000
```

```
nn=4
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb
```

```
4 7
```

```
n(4/7)
```

```
0.571428571428571
```

```
nn=5
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb, n(uq/numb)
```

```
7 23 0.000000000000000
```

```
n(7/23)
```

evaluate

```
0.304347826086957
```

```
nn=6
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb
```

```
34 122
```

```
n(34/122)
```

```
0.278688524590164
```

```
nn=7
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb
```

```
303 888
```

```
n(303/888)
```

```
0.341216216216216
```

```
nn=8
uq=uni(nn)
numb=num_noisol(nn)
print uq, numb
```

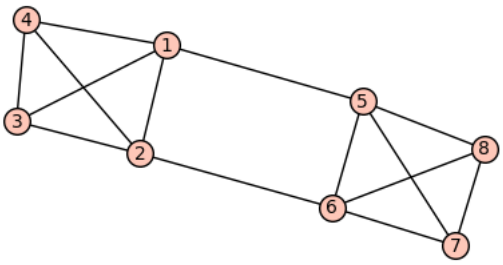
```
5318 11302
```

```
n(5318/11302)
```

```
0.470536188285259
```

```
###################################
# Example H(2) in Figure 3.2 with Zbar + 1 < z_0.
###################################
```

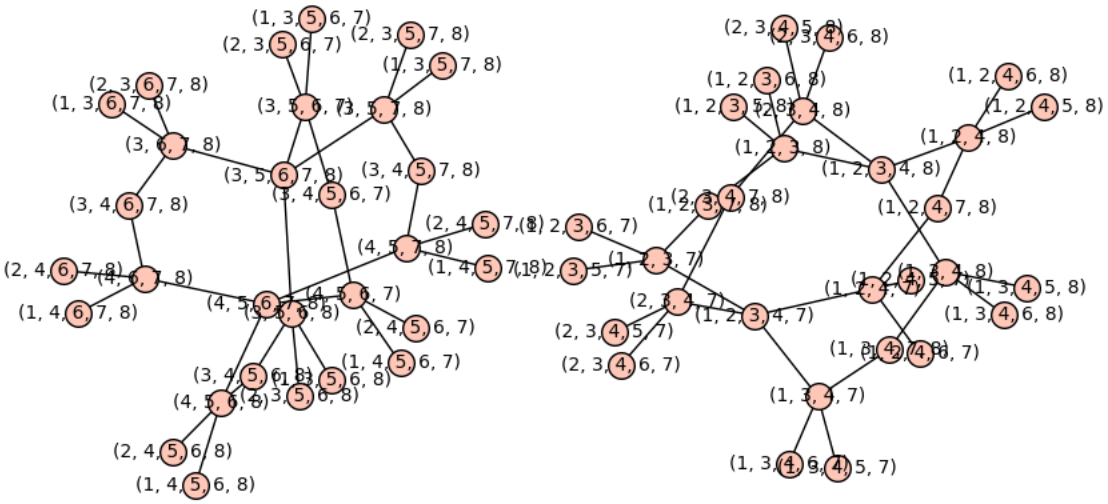
```
h1=graphs.CompleteGraph(4)
h2=graphs.CompleteGraph(4)
h2.relabel([5,6,7,8])
h1.relabel([1,2,3,4])
h=h1.union(h2)
h.add_edges([(1,5),(2,6)])
show(h)
```




```
print Z(h), zbar(h)
ZF_min_sets(h)
```

```
4 4
[{1, 2, 3, 7},
 {1, 2, 3, 8},
 {1, 2, 4, 7},
 {1, 2, 4, 8},
 {1, 3, 4, 7},
 {1, 3, 4, 8},
 {2, 3, 4, 7},
 {2, 3, 4, 8},
 {3, 5, 6, 7},
 {3, 5, 6, 8},
 {3, 5, 7, 8},
 {3, 6, 7, 8},
 {4, 5, 6, 7},
 {4, 5, 6, 8},
 {4, 5, 7, 8},
 {4, 6, 7, 8}]
```

```
targ=TAR_reconfig(h,5)
targ.plot()
```



```
targ.is_connected()
```

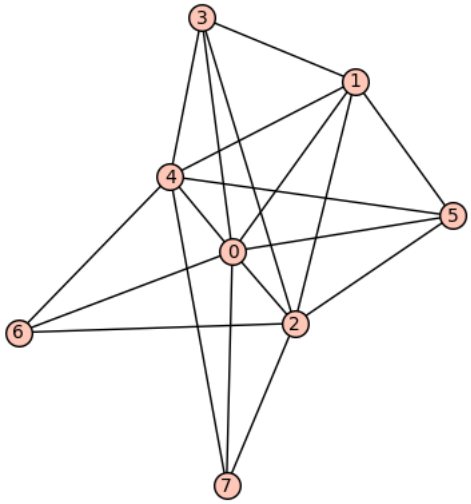
```
False
```

```
targ6=TAR_reconfig(h,6)
targ6.is_connected()
```

```
True
```

```
###################################
# Example H_2 in Figure 3.3 with underline-z_0 < z_0.
###################################
```

```
g=Graph("G~vlTO")
show(g)
```



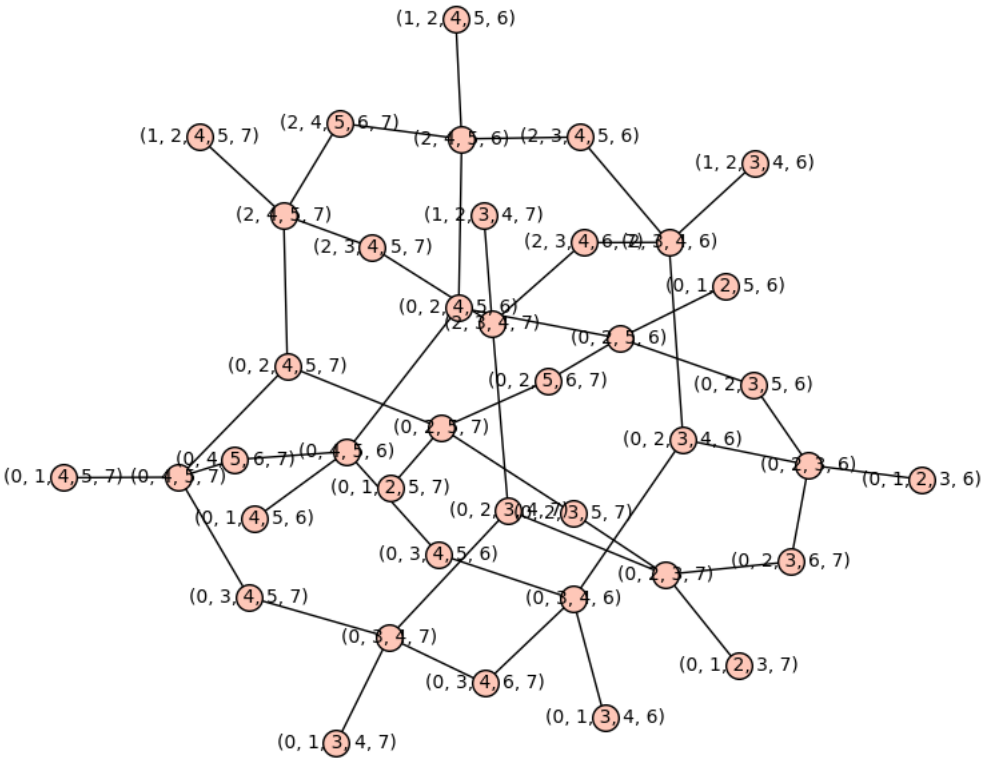

```
ZofG=Z(g)
ZbarG=zbar(g)
print ZofG, ZbarG
print TAR_reconfig(g,ZofG).is_connected()
print TAR_reconfig(g,ZofG+1).is_connected()
print TAR_reconfig(g,ZbarG).is_connected()
print TAR_reconfig(g,ZbarG+1).is_connected()
```

```
4 6
False
True
False
True
```

```
ZF_min_sets(g)
```

```
[{0, 2, 3, 6},
 {0, 2, 3, 7},
 {0, 2, 5, 6},
 {0, 2, 5, 7},
 {0, 3, 4, 6},
 {0, 3, 4, 7},
 {0, 4, 5, 6},
 {0, 4, 5, 7},
 {2, 3, 4, 6},
 {2, 3, 4, 7},
 {2, 4, 5, 6},
 {2, 4, 5, 7},
 {1, 2, 3, 5, 6, 7},
 {1, 3, 4, 5, 6, 7}]
```

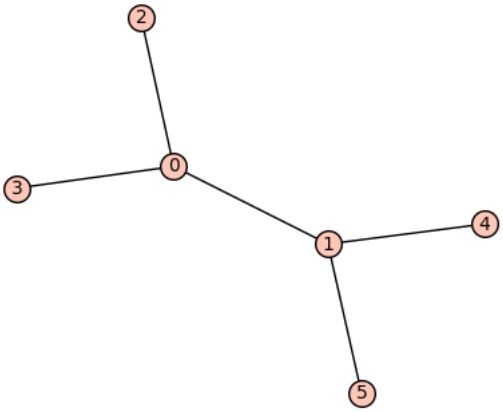
```
targ5=TAR_reconfig(g,ZofG+1)
targ5.plot()
```




```
# Necessarily TAR_reconfig(g,6) is disconnected because new minimal sets are added.
```

```
###################################
# Example 3.7: Removing one of only two twins may not reduce zero forcing number.
###################################
```

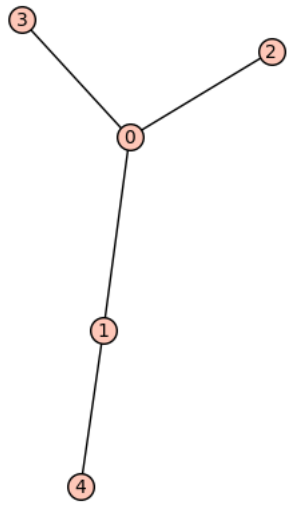
```
g=Graph({0:[1,2,3],1:[4,5]})
show(g)
```




```
print Z(g)
ZF_min_sets(g)
```

```
2
[{2, 4}, {2, 5}, {3, 4}, {3, 5}]
```

```
h=Graph({0:[1,2,3],1:[4]})
show(h)
```




```
print Z(h)
ZF_min_sets(h)
```

```
2
[{1, 2}, {1, 3}, {2, 3}, {2, 4}, {3, 4}]
```

```
# F-1 load main minimum rank/maximum nullity and zero forcing functions
# zero forcing, PSD forcing, and minimum rank code
# developed by S. Butler, L. DeLoss, J. Grout, H.T. Hall, J. LaGrange, J.C.-H. Lin,T. McKay, J. Smith, G. Tims
# updated and maintained by Jephian C.-H. Lin
URL='https://raw.githubusercontent.com/jephianlin/mr_JG/py2/'
files=['Zq_c.pyx','Zq.py','zero_forcing_64.pyx','zero_forcing_wavefront.pyx','minrank.py', 'inertia.py']
for f in files:
    print("Loading %s..."%f);
    load(URL+f)
```

```
Loading Zq_c.pyx...
Compiling
/home/sageuser/3/.sage/temp/sage.math.iastate.edu/12704/tmp_QfsC8X.pyx..\
.
Loading Zq.py...
Loading zero_forcing_64.pyx...
Compiling
/home/sageuser/3/.sage/temp/sage.math.iastate.edu/12704/tmp_FcK93O.pyx..\
.
Loading zero_forcing_wavefront.pyx...
Compiling
/home/sageuser/3/.sage/temp/sage.math.iastate.edu/12704/tmp_R3X6iY.pyx..\
.
Loading minrank.py...
Loading inertia.py...
```

```
# F-2 other zero forcing functions (Leslie Hogben)
#
# Zero forcing number Z(G)
def Z(g):
    z=len(zero_forcing_set_bruteforce(g))
    return z
#
# All zero forcing sets of G of size k
# input: a graph G and a positive integer k
# output: a list of all zero forcing sets G of size k
def ZFsets(G,k):
    ord=G.order()
    V = G.vertices()
    S = subsets(V,k)
    ZFS=[]
    for s in S:
        A=zerosgame(G,s)
        if len(A)==ord:
            ZFS.append(s)
    return ZFS
```

```
# F-R Reconfiguration graph functions
# adapted from code by Chassidy Bozeman


# input: A graph G and a positive integer k
# output: All zero forcing sets G up to size k


def ZFS_up_to_size_k(G,k):
    S=[]
    for i in [Z(G)..k]:
        ZFS_size_i=ZFsets(G,i)
        S=S+ZFS_size_i
    return S


# input: A graph G and a positive integer k
# output: The TAR ZF reconfiguration graph on power dominating set up to size k.

def TAR_reconfig(G,k):

    S=ZFS_up_to_size_k(G,k)
    # creates a list of all zf sets up to size k.

    H=Graph()
    # creates empty graph

    H.add_vertices(S)
    # add zf sets up to size k to the vertices of H

    # The following part determines if the size of two power dominating sets differ
    # by one and if one is a subset of the other. If both are true, an edge is added.


    for i in range(len(S)):
        for j in range(i+1, len(S)):
            if len(S[i])-len(S[j]) in {-1,1}:
                    if set(S[i]).issubset(set(S[j])):
                        H.add_edge(S[i],S[j])
                    else:
                        if set(S[j]).issubset(set(S[i])):
                            H.add_edge(S[i],S[j])
    return H



#input: A graph G and integer k
#output: The TARE PD reconfiguration graph on PDS of size up to k
```

```
def TARE_reconfig(G,k):
    # The first part of this code is the same as that given for TAR above.

    S=ZFS_up_to_size_k(G,k)
    H=Graph()
    H.add_vertices(S)
    for i in range(len(S)):
        for j in range(i+1,len(S)):
            if len(S[i])-len(S[j]) in {-1,1}:
                    if set(S[i]).issubset(set(S[j])):
                        H.add_edge(S[i],S[j])
                    else:
                        if set(S[j]).issubset(set(S[i])):
                            H.add_edge(S[i],S[j])

                # The part below is for token exchange. If two power dominating sets have the
                # same size and they differ by one element, then an edge is added between them.

            if len(S[i])==len(S[j]):
                    if len(set(S[i]).difference(set(S[j])))==1:
                        H.add_edge(S[i],S[j])
    return H

def TE_reconfig(G,k):

    S=ZFsets(G,k)
    #creates a list of all zf sets of size k.

    H=Graph()
    #creates empty graph


    H.add_vertices(S)
    # add power dom sets of size k to the vertices of H

    # The part below is for token exchange. If two power dominating sets have the
    # same size and they differ by one element, then an edge is added between them.

    for i in range(len(S)):
        for j in range(i+1, len(S)):
            if len(S[i])==len(S[j]):
                if len(set(S[i]).difference(set(S[j])))==1:
```

```
#F-Zpoly
def ZFS_poly(G):   # Bryan Curtis
    d = order(G)
    coeff = [len(ZFsets(G,k)) for k in range(1,d+1)]
    return coeff
```

```
#F-unique

# The following code finds the number of graphs (with no isol vtxs) with unique reconfiguration graphs
#
def uni(n):
    # Store all graphs without isolated vertices to a list:
    no_isol = [g for g in graphs(n) if min([g.degree(v) for v in g]) >=1]
    # Create a list of lists - each sublist contains all graphs with the same reconfiguration graph
    # The first element of each sublist is a copy of the reconfiguration graph (directly corresponding to the
second element)
    unique = []
    for g in no_isol:
        flag = False
        tar_g = TAR_reconfig(g,n)
        for h in unique:
            if tar_g.is_isomorphic(h[0]):
                h.append(g)
                flag = True
                break
        if not flag:
            unique.append([tar_g,g])
    # Keep only graphs with unique reconfiguration graph:
    unique = [p for p in unique if len(p) == 2]
    return len(unique)
    # Display examples (may want to suppress output)
    # for g in unique:
    #     g[1].show()
#
def num_noisol(n):
    # Store all graphs without isolated vertices to a list:
    no_isol = [g for g in graphs(n) if min([g.degree(v) for v in g]) >=1]
    return len(no_isol)
```

```
#F-zbar
def get_minimal_subsets(sets):
    sets = sorted(map(set, sets), key=len)
    minimal_subsets = []
    for s in sets:
        if not any(minimal_subset.issubset(s) for minimal_subset in minimal_subsets):
            minimal_subsets.append(s)
    return minimal_subsets

# Outputs all minimal zero forcing sets
def ZF_min_sets(G):
    ord = G.order()
    ZFS = ZFS_up_to_size_k(G,ord)
    mZFS = get_minimal_subsets(ZFS)
    return mZFS

# Outputs the upper minimal zero forcing number
def zbar(G):
    min_list = ZF_min_sets(G)
    zb = max(len(x) for x in min_list)
```