

ACFHMTW-Product_Power_Throttling

last edited Oct 22, 2020, 2:41:23 PM by hogben

☐ Typeset
☐ Load 3-D Live
☐ Use java for 3-D

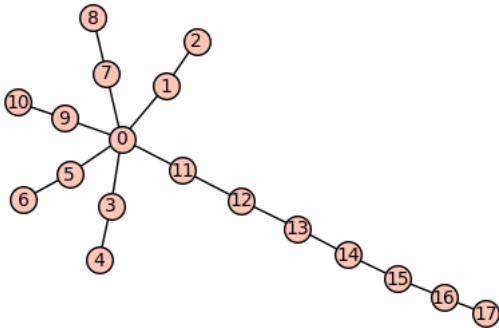

```
#####
# This Sage worksheet is written by Leslie Hogben
# based on code written by numerous other people, some of whom are listed.
# It contains examples from the paper
# Product throttling for power domination
# by Sarah Anderson, Karen Collins, Daniela Ferrero, Leslie Hogben, Carolyn Mayer, Ann Trenk, Shanise Walker
# Please report errors to hogben@aimath.org
#####
```

```
#####
# To run the examples, you must got to the bottom of this sheet
# and run all the function cells before running the examples.
#####
```

```
#####
# Example 2.4
#####
```

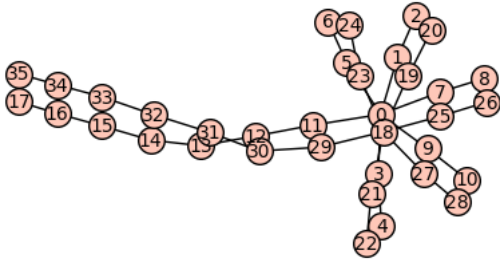
```
g=Graph({1:[0,2],3:[0,4],5:[0,6],7:[0,8],9:[0,10],0:[11],12:[11,13],14:[13,15],15:[14,16],17:[16]})
show(g)
print 'dom set=', g.dominating_set(), 'gamma_p=', PowerDom(g), 'ppt(g)=' ,ppt(g)
print thpdx(g,18)
```

```
dom set= [0, 1, 3, 5, 7, 9, 13, 16] gamma_p= 1 ppt(g)= 7
th_PD^x(G) 4 k 2
4
```



```
#####
# Example 6.2
#####
```

```
p=Graph({18:[19]})
gp=p.cartesian_product(g)
gp.relabel()
show(gp)
```



```
gds=gp.dominating_set()
print 'dom set=', gds, 'gamma(gp)=', len(gds)
print 'gamma_p=', PowerDom(gp), 'ppt(gp)=', ppt(gp)
```

```
dom set= [0, 11, 15, 20, 22, 24, 26, 28, 31, 35] gamma(gp)= 10
gamma_p= 2 ppt(gp)= 7
```

```
pptS(gp, [0, 18, 15, 33])
```

```
2
```

```
# Thus  $th\_PD^x(gp) \leq 8$ .
# By Remark 3.8(2), need consider  $2 \leq k \leq 3$  for  $th\_PD^x(gp, k) = th\_PD^x(gp)$ 
# Since  $ppt(gp, 4) \geq 2$  and  $ppt(gp, \{0, 18, 15, 33\}) = 2$ ,  $ppt(gp, 4) = 2$ .
# As above,  $ppt(gp, 2) = 7$ .
```

```
'ppt(gp, 3)=', pptk(gp, 3)
```

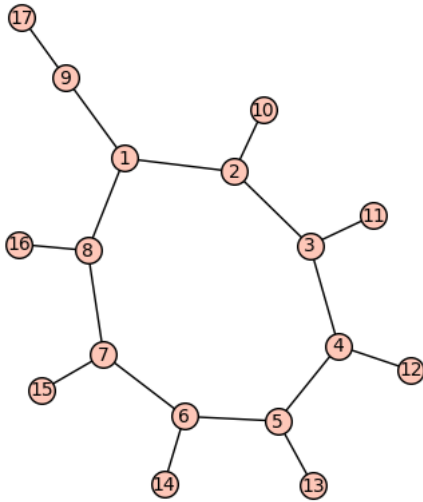
```
('ppt(gp, 3)=', 4)
```

```
# Thus  $th\_PD^x(gp) = 8$ .
```

```
#####
# Example 6.3
#####
```

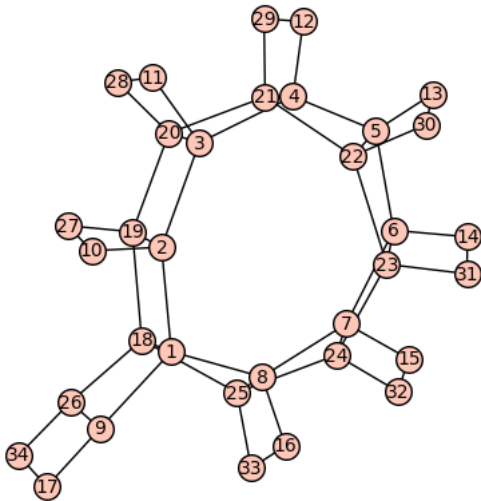
```
g=Graph({9: [1, 17], 2: [1, 3, 10], 3: [4, 11], 4: [5, 12], 5: [6, 13], 6: [7, 14], 7: [8, 15], 8: [1, 16]})
show(g)
print 'dom set=', g.dominating_set(), 'gamma_p=', PowerDom(g), 'ppt(g)=', ppt(g)
print thpdx(g, 17)
```

```
dom set= [2, 3, 4, 5, 6, 7, 8, 9] gamma_p= 3 ppt(g)= 2
th_PD^x(G) 6 k 3
6
```



```
# Thus  $\text{th\_PD}^x(g) \leq 6$ .
```

```
p=Graph({18:[19]})
gp=p.cartesian_product(g)
gp.relabel([1..34])
show(gp)
```



```
gds=gp.dominating_set()
print 'dom set=', gds, 'gamma(gp)=', len(gds)
print 'gamma_p=', PowerDom(gp), 'ppt(gp)=', ppt(gp)
```

```
dom set= [1, 4, 11, 13, 15, 21, 23, 27, 31, 33, 34] gamma(gp)= 11
gamma_p= 3 ppt(gp)= 7
```

```
pptS(gp, [1,5,18,20,24])
```

```
2
```

```
# Thus  $\text{th\_PD}^x(gp) \leq 10$ .
# By Remark 3.8(2), need consider  $3 \leq k \leq 5$  for  $\text{th\_PD}^x(gp, k) = \text{th\_PD}^x(gp)$ 
# and  $k=3, k=5$  have been done.
```

```
'ppt(gp,4)=', pptk(gp,4)
```

```
('ppt(gp,4)=' , 3)
```

```
# Thus  $\text{th\_PD}^x(\text{gp}) = 10$ .
```

```
#####  
# Theorem 6.11  
#####
```

```
#####  
# Cartesian products of  $P_n[]P_m$  for  $n = 4,5,6$   
#  
# For  $n = 4$  case, check  $m = 5,6,9$   
# For  $n = 5$  case, check  $m = 8,12$   
# For  $n = 6$  case, check  $m = 6,10$   
#####
```

```
p4 = graphs.PathGraph(4)  
p5 = graphs.PathGraph(5)  
p6 = graphs.PathGraph(6)  
c4 = graphs.CycleGraph(4)  
c5 = graphs.CycleGraph(5)  
c6 = graphs.CycleGraph(6)
```

```
# n = 4:
```

```
print 'domination number p4[]p5 =', dom(p4.cartesian_product(p5))  
print 'throttle product p4[]p5 =', thpdx(p4.cartesian_product(p5))  
print 'domination number p4[]p6 =', dom(p4.cartesian_product(p6))  
print 'throttle product p4[]p6 =', thpdx(p4.cartesian_product(p6))  
print 'domination number p4[]p9 =', dom(p4.cartesian_product(graphs.PathGraph(9)))  
print 'throttle product p4[]p9 =', thpdx(p4.cartesian_product(graphs.PathGraph(9)))
```

```
domination number p4[]p5 = 6  
throttle product p4[]p5 = 6  
domination number p4[]p6 = 7  
throttle product p4[]p6 = 7  
domination number p4[]p9 = 10  
throttle product p4[]p9 = 10
```

```
print 'domination number c4[]c5 =', dom(c4.cartesian_product(graphs.CycleGraph(5)))  
print 'domination number c4[]c6 =', dom(c4.cartesian_product(graphs.CycleGraph(6)))  
print 'domination number c4[]c9 =', dom(c4.cartesian_product(graphs.CycleGraph(9)))  
print 'domination number p4[]c6 =', dom(p4.cartesian_product(graphs.CycleGraph(6)))
```

```
domination number c4[]c5 = 5  
domination number c4[]c6 = 6  
domination number c4[]c9 = 9  
domination number p4[]c6 = 6
```

```
print 'domination number p4[]c5 =', dom(p4.cartesian_product(graphs.CycleGraph(5)))
print 'throttle product p4[]c5=', thpdx(p4.cartesian_product(graphs.CycleGraph(5)))
print 'domination number p4[]c9=', dom(p4.cartesian_product(graphs.CycleGraph(9)))
print 'throttle product p4[]c9=', thpdx(p4.cartesian_product(graphs.CycleGraph(9)))
```

```
domination number p4[]c5 = 6
throttle product p4[]c5= 6
domination number p4[]c9= 10
throttle product p4[]c9= 10
```

```
# n = 5:
```

```
print 'domination number p5[]c8 =', dom(p5.cartesian_product(graphs.CycleGraph(8))), ', ceil(5m/4)=', 5*8/4
print 'domination number p5[]c12 =', dom(p5.cartesian_product(graphs.CycleGraph(12))), ', ceil(5m/4)=', 5*12/4
print 'domination number c5[]p8 =', dom(c5.cartesian_product(graphs.PathGraph(8))), ', ceil(5m/4)=', 5*8/4
print 'domination number c5[]p12 =', dom(c5.cartesian_product(graphs.PathGraph(12))), ', ceil(5m/4)=', 5*12/4
print 'domination number c5[]c8 =', dom(c5.cartesian_product(graphs.CycleGraph(8))), ', ceil(5m/4)=', 5*8/4
print 'domination number c5[]c12 =', dom(c5.cartesian_product(graphs.CycleGraph(12))), ', ceil(5m/4)=', 5*12/4
```

```
domination number p5[]c8 = 10 , ceil(5m/4)= 10
domination number p5[]c12 = 15 , ceil(5m/4)= 15
domination number c5[]p8 = 10 , ceil(5m/4)= 10
domination number c5[]p12 = 14 , ceil(5m/4)= 15
domination number c5[]c8 = 10 , ceil(5m/4)= 10
domination number c5[]c12 = 13 , ceil(5m/4)= 15
```

```
# p5[]p8:
```

```
print 'domination number =', dom(p5.cartesian_product(graphs.PathGraph(8)))
print 'throttle product =', thpdx(p5.cartesian_product(graphs.PathGraph(8)))
```

```
domination number = 11
throttle product = 11
```

```
# n = 6:
```

```
print 'domination number p6[]c6 =', dom(p6.cartesian_product(graphs.CycleGraph(6))), ', ceil(6m/4)=', 6*6/4
print 'domination number p6[]c10 =', dom(p6.cartesian_product(graphs.CycleGraph(10))), ', ceil(6m/4)=', 6*10/4
print 'domination number c6[]p6 =', dom(c6.cartesian_product(graphs.PathGraph(6))), ', ceil(6m/4)=', 6*6/4
print 'domination number c6[]p10 =', dom(c6.cartesian_product(graphs.PathGraph(10))), ', ceil(6m/4)=', 6*10/4
print 'domination number c6[]c6 =', dom(c6.cartesian_product(graphs.CycleGraph(6))), ', ceil(6m/4)=', 6*6/4
print 'domination number c6[]c10 =', dom(c6.cartesian_product(graphs.CycleGraph(10))), ', ceil(6m/4)=', 6*10/4
```

```
domination number p6[]c6 = 9 , ceil(6m/4)= 9
domination number p6[]c10 = 15 , ceil(6m/4)= 15
domination number c6[]p6 = 9 , ceil(6m/4)= 9
domination number c6[]p10 = 14 , ceil(6m/4)= 15
domination number c6[]c6 = 8 , ceil(6m/4)= 9
domination number c6[]c10 = 14 , ceil(6m/4)= 15
```

```
# p6[]p6
```

evaluate

```
print 'domination number =', dom(p6.cartesian_product(graphs.PathGraph(6)))
print 'throttle product =', thpdx(p6.cartesian_product(graphs.PathGraph(6)))
```

```
domination number = 10
throttle product = 10
```

```
# Special cases p5[]p12 and p6[]p10:
#
# For p5[]p12 and p6[]p10, the domination # is 16 and the size of a minimum power dominating set is 2.
```

```
print 'domination number p5[]p12 =', dom(p5.cartesian_product(graphs.PathGraph(12)))
print 'domination number p6[]p10 =', dom(p6.cartesian_product(graphs.PathGraph(10)))
print 'gamma_P(p5[]p12)=', len(minPowerDominatingSet(p5.cartesian_product(graphs.PathGraph(12))))
print 'gamma_P(p6[]p10)=', len(minPowerDominatingSet(p6.cartesian_product(graphs.PathGraph(10))))
```

```
domination number p5[]p12 = 16
domination number p6[]p10 = 16
gamma_P(p5[]p12)= 2
gamma_P(p6[]p10)= 2
```

```
# For p5[]p12 and p6[]p10, we need only check that ppt(G,|S|) > t for the following (|S|,t) pairs: (3,5), (5,3)
```

```
print 'ppt(p5[]p12,3)=', pptk((graphs.PathGraph(5)).cartesian_product(graphs.PathGraph(12)),3)
print 'ppt(p6[]p10,3)=', pptk((graphs.PathGraph(6)).cartesian_product(graphs.PathGraph(10)),3)
```

```
ppt(p5[]p12,3)= 6
ppt(p6[]p10,3)= 9
```

```
print 'ppt(p5[]p12,5)=', pptk((graphs.PathGraph(5)).cartesian_product(graphs.PathGraph(12)),5)
```

```
ppt(p5[]p12,5)= 4
```

```
print 'ppt(p6[]p10,5)=', pptk((graphs.PathGraph(6)).cartesian_product(graphs.PathGraph(10)),5)
```

```
ppt(p6[]p10,5)= 5
```

```
#####
# The function cells are below.
# Enter each function cell in order before
# trying to run the examples above.
#####
```

```
# F1
# Load main 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/1/.sage/temp/sage.math.iastate.edu/1054/tmp_rHjEmY.pyx...
Loading Zq.py...
Loading zero_forcing_64.pyx...
Compiling
/home/sageuser/1/.sage/temp/sage.math.iastate.edu/1054/tmp_89lBLg.pyx...
```

```

Loading zero_forcing_wavefront.pyx...
Compiling
/home/sageuser/1/.sage/tmp/sage.math.iastate.edu/1054/tmp_BvPjca.pyx...
Loading minrank.py...
Loading inertia.py...

```

```

# F2
# Function for standard propagation time of set S, pt(G,S)
# adapted by Leslie Hogben from Steve Butler's skew propagation time code (F-10)
# input: a graph G and a set S of vertices
# output: pt(G,S) = prop time of S in G (if -1 is returned then S is not a zero forcing set)
def ptz(G,S):
    V=set(G.vertices())
    count = 0
    done = False
    active = set(S)
    filled = set(S)
    for v in V:
        N=set(G.neighbors(v))
        if v in active and N.issubset(filled):
            active.remove(v)
        if (v in filled) and (v not in active) and (len(N.intersection(filled)) == G.degree(v)-1):
            active.add(v)
    while not done:
        done = True
        new_active = copy(active)
        new_filled = copy(filled)
        for v in active:
            N=set(G.neighbors(v))
            if len(N.intersection(filled)) == G.degree(v)-1:
                if done:
                    done = False
                    count += 1
                    N.symmetric_difference_update(N.intersection(filled))
                    u=N.pop()
                    new_active.remove(v)
                    new_active.add(u)
                    new_filled.add(u)
        active = copy(new_active)
        filled = copy(new_filled)
        # print filled
        for v in V:
            N=set(G.neighbors(v))
            if v in active and N.issubset(filled):
                active.remove(v)
            if (v in filled) and (v not in active) and (len(N.intersection(filled)) == G.degree(v)-1):
                active.add(v)
    if len(filled)==len(V):
        return count
    return -1

```

```

# F3
# More functions for standard propagation time
# pt(G,k) = min prop time over zero forcing sets of G of size k
# adapted by Leslie Hogben from Steve Butler's skew propagation time code (F-10)
# this is the function used to compute propagation time of G and also throttling
# input: a graph G and a positive integer k >= Z(G)
# output: pt(G,k) = min prop time over zero forcing sets G of size k (if order+1 is returned then k<Z(G))
def ptk(G,k):
    V=set(G.vertices())
    sets = subsets(V,k)
    pt=len(V)+1
    for S in sets:
        count = 0
        done = False
        active = set(S)
        filled = set(S)
        for v in V:
            N=set(G.neighbors(v))
            if v in active and N.issubset(filled):
                active.remove(v)
            if (v in filled) and (v not in active) and (len(N.intersection(filled)) == G.degree(v)-1):
                active.add(v)
        while not done:
            done = True
            new_active = copy(active)
            new_filled = copy(filled)
            for v in active:
                N=set(G.neighbors(v))
                if len(N.intersection(filled)) == G.degree(v)-1:
                    if done:
                        done = False
                        count += 1
                        N.symmetric_difference_update(N.intersection(filled))
                        u=N.pop()
                        new_active.remove(v)
                        new_active.add(u)
                        new_filled.add(u)
            active = copy(new_active)
            filled = copy(new_filled)
            # print filled
            for v in V:
                N=set(G.neighbors(v))
                if v in active and N.issubset(filled):
                    active.remove(v)
                if (v in filled) and (v not in active) and (len(N.intersection(filled)) == G.degree(v)-1):
                    active.add(v)
            if len(filled)==len(V):
                if count < pt:
                    pt=count
        return pt

# propagation time pt(G)
# input: a graph G
# output: propagation time pt(G)=pt(G,Z(G))
def pt(g):
    ptg=ptk(g,Z(g))
    return ptg

```

```

# F4
# Power dominating set functions
# by Brian Wissman

# input: a graph G and a subset of its vertices V
# output: true/false depending if V is a power dominating set of G
def isPowerDominatingSet(G,V):
    N=[]
    for i in V:
        N+=G.neighbors(i)
    NVert=uniq(N+list(V))

    A=zerosgame(G,NVert)
    if len(A)==G.order():
        return True
    else:
        return False

# input: a graph G
# output: a power dominating set of G that achieves the power domination number
def minPowerDominatingSet(G):
    V = G.vertices()
    for i in range(1,len(V)+1):
        S = subsets(V,i)
        for s in S:
            if isPowerDominatingSet(G,s):
                return s

# input: a graph G
# output: the power domination number of G, gamma_P(G)
def PowerDom(G):
    V = G.vertices()
    for i in range(1,len(V)+1):
        S = subsets(V,i)
        for s in S:
            if isPowerDominatingSet(G,s):

```

```

# F5
# Find power dominating sets of order k

# input: a graph G and a and an integer k >= gamma_p(G)
# output: printout of all power dominating sets of G of cardinality k
def kpds(G,k):
    V = G.vertices()
    S = subsets(V,k)
    for s in S:
        if isPowerDominatingSet(G,s):

```

```

# F6
# Functions for power propagation time

# function to compute closed neighborhood N[S]
# input: a graph G and a list S of vertices
# output: set N[S]
def nhbd(G,S):
    NS=set(S)
    for x in S:
        nx=G.neighbors(x)
        NS=NS.union(set(nx))
    return NS

# function to compute ppt(G,S) = power propagation time of a set S
# input: a graph G and a subset S of vertices
# output: ppt(G,S)
def pptS(G,S):
    ord=G.order()
    V = G.vertices()
    NS=nhbd(G,S)
    ptNS=ptz(G,NS)
    if ptNS>-1:
        ptNS=ptNS+1
    return ptNS

# function to compute ppt(G,k) = minimum power propagation time over sets of size k
# input: a graph G and a positive integer k
# output: ppt(G,k)
def pptk(G,k):
    ord=G.order()
    V = G.vertices()
    S = subsets(V,k)
    ptm = -1
    for s in S:
        ptms=pptS(G,s)
        if (ptm < 0):
            ptm=ptms
        if (ptms >= 0) and (ptms < ptm):
            ptm=ptms
    return ptm

# function to compute ppt(G) = ppt(G,gamma_P(G))
# input: a graph G and a positive integer k
# output: ppt(G)

```

```

# F7
# compute domination number of graph g
def dom(g):
    gds=g.dominating_set()
    ds=len(gds)
    return ds

```

```

# F8
# Functions for product power throttling number
#
# This function computes  $\text{th\_PD}^x(G)$  by minimizing  $\text{th\_PD}^x(G,k)$ 
#
# input: a graph G
# output: power product throttling number  $\text{th\_PD}^x(G)$  and least k such that  $\text{th\_PD}^x(G) = \text{th\_PD}^x(G,k)$ 
def thpdx(g):
    z=PowerDom(g)
    t=pptk(g,z)
    thz=z*t
    kmax=floor(dom(g)/2)
    for k in [z..kmax]:
        ptkg=pptk(g,k)
        if k*ptkg<thz:
            thz=k*ptkg
            ko=k
    thz = min(thz, dom(g))
    return thz

# This function computes  $\text{th\_PD}^x(G)$  by minimizing  $\text{th\_PD}^x(G,k)$  for
#  $\gamma_P(G) \leq k \leq \min(\gamma(G), b)/2$ 
# and reports both
#  $\text{th\_PD}^x(G)$  and ko such that  $\text{th\_PD}^x(G) = \text{th\_PD}^x(G, ko)$ 
#
# input: a graph G and a bound b where it is known  $\text{th\_PD}^x(G) \leq b$ 
# (if known be known, use n)
# output: returns power product throttling number  $\text{th\_PD}^x(G)$ 
# and prints  $\text{th\_PD}^x(G)$  and least k such that  $\text{th\_PD}^x(G) = \text{th\_PD}^x(G,k)$ 
def thpdx(b):
    d=dom(g)
    if d<b:
        b=d
    z=PowerDom(g)
    t=pptk(g,z)
    ko=z
    thz=z*t
    kmax=floor(b/2)
    for k in [z..kmax]:
        ptkg=pptk(g,k)
        if k*ptkg<thz:
            thz=k*ptkg
            ko=k
    thz = min(thz, dom(g))
    if thz==dom(g):
        ko=1
    print 'th_PD^x(G)', thz, 'k', ko
    return thz

```

