SRG-76-30-8-14 eliminating K6,10

```
#generating all possible lengths of cycles on 20 vertices, ascending
order
types=[]
for k in range(1,7):
    d=[0]*k
    m=int(11-3*k/2)
    pp=m^(k-1)
    for i in range(pp):
        ii=i
        d[k-1]=20-3*k
        for j in range(k-1):
            d[j]=int(ii)%int(m)
            ii=ii//m
            d[k-1]-=(k-j)*d[j]
        if d[k-1]>=0:
            ii=3
            res=[]
            for j in range(k):
                ii+=d[j]
                res.append(ii)
            types.append(res)

#list of alternating 0s and 1s of given length
def alt_gen(k):
    alt = [1]*k
    for i in range((k-1)/2+1):
        alt[2*i]=0
    return alt

#appending some info to res: pair of numbers of edges in the two
components, alternating sequence with duplications at given positions
#dups is a non-decreasing sequence of indexes, may be empty
def duplicate(res,dups,nn):
    if nn<=0:
        return 0
    alt = alt_gen(nn)
    for i in range(len(dups)):
        alt.insert(i+dups[i],alt[i+dups[i]])
    x=0 #counter for consecutive 0s (cyclic)
    y=0 #1s
    for i in range(len(alt)-1):
        if alt[i]==0 and alt[i+1]==0:
            x+=1
        if alt[i]==1 and alt[i+1]==1:
            y+=1
        if alt[0]==0 and alt[len(alt)-1]==0:
                x+=1
```

```
        if alt[0]==1 and alt[len(alt)-1]==1:
                y+=1
    if x>3 or y>3:
        return 0 #no need of 3 or more
    res.append([x,y])
    res.append(alt)
    if x!=y: #isomorphic if x=y, otherwise we also add with 0<->1
interchanged
        alt=[1-i for i in alt]
        res.append([y,x])
        res.append(alt)
    return 0

#generates all (up to cyclic translation) lists of 0s and 1s of length n
with no more than three cyclic occurences of 00 and 11, each would
correspond to an edge in H_1 or H_2
#have to call duplicate at all possible "dups" sets, up to cyclic
translation
#let 0=g_0<=g_1<=...<=g_k be the positions to be duplicated
#the positions have values in 0,...,n-k-2
#can assume that the first index g_0=0, and the distance between first
and second g_1-g_0 is smallest cyclically, i.e. g_{j+1}-g_j>=g_1-g_0=g_1
#g_k<=n-k-2 if g_1=0
#g_k<=n-k-1-g_1 if g_1>=1
#so we introduce gg=min(1,g_1)
#then g_k<=n-k-2+gg-g_1, g_{k-1}<=n-k-2+gg-2*g_1, etc.
#g_1<=n-k-2+gg-k*g_1, so g_1<=(n-k-1)/(k+1)
#more than six elements in dups will clearly return empty result, so
k<=5
def enum_gen(n):
    res = []
    duplicate(res,[],n)
    duplicate(res,[0],n-1)
    for g1 in range(0,(n-2)/2+1):
        duplicate(res,[0,g1],n-2)
    for g1 in range(0,(n-3)/3+1):
        gg=min(1,g1)
        for g2 in range(2*g1,n-4+gg-g1):
            duplicate(res,[0,g1,g2],n-3)
    for g1 in range(0,(n-4)/4+1):
        gg=min(1,g1)
        for g2 in range(2*g1,n-5+gg-2*g1):
            for g3 in range(g2+g1,n-5+gg-g1):
                duplicate(res,[0,g1,g2,g3],n-4)
    for g1 in range(0,(n-5)/5+1):
        gg=min(1,g1)
        for g2 in range(2*g1,n-6+gg-3*g1):
            for g3 in range(g2+g1,n-6+gg-2*g1):
                for g4 in range(g3+g1,n-6+gg-g1):
                    duplicate(res,[0,g1,g2,g3,g4],n-5)
    for g1 in range(0,(n-6)/6+1):
```

```
        gg=min(1,g1)
        for g2 in range(2*g1,n-7+gg-4*g1):
            for g3 in range(g2+g1,n-7+gg-3*g1):
                for g4 in range(g3+g1,n-7+gg-2*g1):
                    for g5 in range(g4+g1,n-7+gg-g1):
                        duplicate(res,[0,g1,g2,g3,g4,g5],n-6)
    return res

eg = [enum_gen(n) for n in range(3,21)]

#srg(76,30,8,14)
p=-4/15
q=7/45

#Gramm matrix of a given double list
def GMat(A):
    n = len(A)
    B = Matrix([[q+(p-q)*A[i][j] for j in range(n)] for i in range(n)])
    for i in range(n):
        B[i,i]=1
    return B

#check if positive definite
def mineig(A):
    sp = A.eigenvalues()
    mv = sp[0]
    for v in sp:
        if v<mv:
            mv=v
    return mv

#double list of incidence of given type and enumeration, ex:[4,16],
[[0,1,..],[1,0,...]]
def ttm(type,colors):
    res = [[0 for i in range(20)] for j in range(20)]
    n = len(type)
    ind = [0] #indexes for blocks defined by type
    k = 0
    for i in type:
        k += i
        ind.append(k)
    #generating enumeration
    color = [0]*20
    for i in range(n):
        for j in range(len(colors[i])):
            color[ind[i]+j]=colors[i][j]
    #all non-edges between diff colors
    for i in range(19):
        for j in range(i+1,20):
            if color[i]!=color[j]:
                res[i][j]=1
```

```
                res[j][i]=1
    #inverting edges between H_1 and H_2
    for i in range(n):
        j1 = ind[i]
        j2 = ind[i+1]-1
        res[j1][j2]=1-res[j1][j2]
        res[j2][j1]=1-res[j2][j1]
        for j in range(j1,j2):
            res[j][j+1]=1-res[j][j+1]
            res[j+1][j]=1-res[j+1][j]
    return res


#main loop
html('<!--notruncate-->')
total_graphs = 0
for type in types:
    n = len(type)
    nn = [len(eg[k-3])/2 for k in type]
    pp = 1
    for a in nn:
        pp *= a
    print "type ", type
    ii = [0]*n
    fe = 0
    fr = 0
    fp = 0
    for i in range(pp):
        k = i
        for j in range(n):
            ii[j] = int(k) % int(nn[j])
            k = k//nn[j]
        x = 0
        y = 0
        for j in range(n):
            x += eg[type[j]-3][2*ii[j]][0]
            y += eg[type[j]-3][2*ii[j]][1]
        if x>3 or y>3 or x!=y:
            fe += 1
        else:
            M = GMat(ttm(type,[eg[type[j]-3][2*ii[j]+1] for j in
range(n)]))
            r = M.rank()
            if r>16:
                fr += 1
            else:
                mev = mineig(M)
                if mev<0:
                    fp += 1
                else:
                    print "through: ",type,[eg[type[j]-3][2*ii[j]+1] for
```

```
j in range(n)],r,mev
    print "graphs:", pp-fe
    total_graphs += (pp-fe)
    print "failed by rank:", fr
    print "failed by posdef:", fp

print "Graphs (matrices) checked:", total_graphs

################################

#checking double lists of incidence of given type and enumeration, with
all possibilities of adding 21st vertex, up to 3 edges to each half
def min_rank_ttma(type,colors):
    total_graphs = 0
    res = [[0 for i in range(21)] for j in range(21)]
    n = len(type)
    ind = [0] #indexes for blocks defined by type
    k = 0
    for i in type:
        k += i
        ind.append(k)
    #generating enumeration
    color = [0]*20
    for i in range(n):
        for j in range(len(colors[i])):
            color[ind[i]+j]=colors[i][j]
    #all non-edges between diff colors
    for i in range(19):
        for j in range(i+1,20):
            if color[i]!=color[j]:
                res[i][j]=1
                res[j][i]=1
    #inverting edges between H_1 and H_2
    for i in range(n):
        j1 = ind[i]
        j2 = ind[i+1]-1
        res[j1][j2]=1-res[j1][j2]
        res[j2][j1]=1-res[j2][j1]
        for j in range(j1,j2):
            res[j][j+1]=1-res[j][j+1]
            res[j+1][j]=1-res[j+1][j]
    #no edges
    minr = GMat(res).rank()
    total_graphs += 1
    #one edge
    for i in range(10):
        for j in range(10,20):
            for ic in range(20):
                res[20][ic]=0
                res[ic][20]=0
            res[20][i]=1
```

```
            res[i][20]=1
            res[20][j]=1
            res[j][20]=1
            rr = GMat(res).rank()
            total_graphs += 1
            if rr<minr:
                minr=rr
                if rr==16:
                    print type,colors,i,j
    #two edges
    for i1 in range(9):
        for i2 in range(i1+1,10):
            for j1 in range(10,19):
                for j2 in range(j1+1,20):
                    for ic in range(20):
                        res[20][ic]=0
                        res[ic][20]=0
                    res[20][i1]=1
                    res[i1][20]=1
                    res[20][j1]=1
                    res[j1][20]=1
                    res[20][i2]=1
                    res[i2][20]=1
                    res[20][j2]=1
                    res[j2][20]=1
                    rr = GMat(res).rank()
                    total_graphs += 1
                    if rr<minr:
                        minr=rr
                        if rr==16:
                            print type,colors,i1,i2,j1,j2
    #three edges
    for i1 in range(8):
        for i2 in range(i1+1,9):
            for i3 in range(i2+1,10):
                for j1 in range(10,18):
                    for j2 in range(j1+1,19):
                        for j3 in range(j2+1,20):
                            for ic in range(20):
                                res[20][ic]=0
                                res[ic][20]=0
                            res[20][i1]=1
                            res[i1][20]=1
                            res[20][j1]=1
                            res[j1][20]=1
                            res[20][i2]=1
                            res[i2][20]=1
                            res[20][j2]=1
                            res[j2][20]=1
                            res[20][i3]=1
                            res[i3][20]=1
```

```
                                res[20][j3]=1
                                res[j3][20]=1
                                rr = GMat(res).rank()
                                total_graphs += 1
                                if rr<minr:
                                    minr=rr
                                    if rr==16:
                                        print type,colors,i1,i2,i3,j1,j2,j3
    return total_graphs, minr


print min_rank_ttma([4,4,4,4,4],[[0,1,0,1],[0,1,0,1],[0,1,0,1],
[0,1,0,1],[0,1,0,1]])
print min_rank_ttma([4,4,4,4,4],[[0,0,1,1],[0,1,0,1],[0,1,0,1],
[0,1,0,1],[0,1,0,1]])
print min_rank_ttma([4,4,4,4,4],[[0,0,1,1],[0,0,1,1],[0,1,0,1],
[0,1,0,1],[0,1,0,1]])
print min_rank_ttma([4,4,4,4,4],[[0,0,1,1],[0,0,1,1],[0,0,1,1],
[0,1,0,1],[0,1,0,1]])
```

```
type  [20]
graphs: 635
failed by rank: 635
failed by posdef: 0
type  [3, 17]
graphs: 450
failed by rank: 450
failed by posdef: 0
type  [4, 16]
graphs: 312
failed by rank: 312
failed by posdef: 0
type  [5, 15]
graphs: 260
failed by rank: 260
failed by posdef: 0
type  [6, 14]
graphs: 233
failed by rank: 233
failed by posdef: 0
type  [7, 13]
graphs: 204
failed by rank: 204
failed by posdef: 0
type  [8, 12]
graphs: 152
failed by rank: 152
failed by posdef: 0
type  [9, 11]
graphs: 184
failed by rank: 184
failed by posdef: 0
type  [10, 10]
```

```
graphs: 145
failed by rank: 145
failed by posdef: 0
type  [3, 3, 14]
graphs: 130
failed by rank: 130
failed by posdef: 0
type  [4, 4, 12]
graphs: 160
failed by rank: 160
failed by posdef: 0
type  [5, 5, 10]
graphs: 50
failed by rank: 50
failed by posdef: 0
type  [6, 6, 8]
graphs: 87
failed by rank: 87
failed by posdef: 0
type  [3, 4, 13]
graphs: 190
failed by rank: 190
failed by posdef: 0
type  [4, 5, 11]
graphs: 110
failed by rank: 110
failed by posdef: 0
type  [5, 6, 9]
graphs: 76
failed by rank: 76
failed by posdef: 0
type  [6, 7, 7]
graphs: 72
failed by rank: 72
failed by posdef: 0
type  [3, 5, 12]
graphs: 76
failed by rank: 76
failed by posdef: 0
type  [4, 6, 10]
graphs: 119
failed by rank: 119
failed by posdef: 0
type  [5, 7, 8]
graphs: 52
failed by rank: 52
failed by posdef: 0
type  [3, 6, 11]
graphs: 128
failed by rank: 128
failed by posdef: 0
type  [4, 7, 9]
graphs: 96
failed by rank: 96
```

```
failed by posdef: 0
type  [3, 7, 10]
graphs: 72
failed by rank: 72
failed by posdef: 0
type  [4, 8, 8]
graphs: 90
failed by rank: 90
failed by posdef: 0
type  [3, 8, 9]
graphs: 84
failed by rank: 84
failed by posdef: 0
type  [3, 3, 3, 11]
graphs: 82
failed by rank: 82
failed by posdef: 0
type  [4, 4, 4, 8]
graphs: 98
failed by rank: 98
failed by posdef: 0
type  [5, 5, 5, 5]
graphs: 14
failed by rank: 14
failed by posdef: 0
type  [3, 4, 4, 9]
graphs: 88
failed by rank: 88
failed by posdef: 0
type  [4, 5, 5, 6]
graphs: 42
failed by rank: 42
failed by posdef: 0
type  [3, 5, 5, 7]
graphs: 26
failed by rank: 26
failed by posdef: 0
type  [3, 3, 4, 10]
graphs: 74
failed by rank: 74
failed by posdef: 0
type  [4, 4, 5, 7]
graphs: 56
failed by rank: 56
failed by posdef: 0
type  [3, 4, 5, 8]
graphs: 46
failed by rank: 46
failed by posdef: 0
type  [3, 5, 6, 6]
graphs: 48
failed by rank: 48
failed by posdef: 0
type  [3, 3, 5, 9]
```

```
graphs: 48
failed by rank: 48
failed by posdef: 0
type  [4, 4, 6, 6]
graphs: 93
failed by rank: 93
failed by posdef: 0
type  [3, 4, 6, 7]
graphs: 64
failed by rank: 64
failed by posdef: 0
type  [3, 3, 6, 8]
graphs: 62
failed by rank: 62
failed by posdef: 0
type  [3, 3, 7, 7]
graphs: 42
failed by rank: 42
failed by posdef: 0
type  [3, 3, 3, 3, 8]
graphs: 58
failed by rank: 58
failed by posdef: 0
type  [4, 4, 4, 4, 4]
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 0, 1, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 0, 1, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 0, 1, 1], [0, 0, 1, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 0, 1, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 0, 1, 1], [0, 1, 0, 1], [0, 0, 1, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 0, 1, 1], [0, 0, 1, 1],
[0, 1, 0, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 0, 1, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 0, 1, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 0, 1, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 0, 1, 1], [0, 1, 0, 1],
[0, 0, 1, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 0, 1, 1],
[0, 0, 1, 1], [0, 1, 0, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 0, 1, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 0, 1, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 0, 1, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 0, 1, 1], [0, 1, 0, 1],
[0, 1, 0, 1], [0, 0, 1, 1]] 16 0
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 0, 1, 1],
[0, 1, 0, 1], [0, 0, 1, 1]] 16 0
```

```
through:  [4, 4, 4, 4, 4] [[0, 1, 0, 1], [0, 1, 0, 1], [0, 1, 0, 1],
[0, 0, 1, 1], [0, 0, 1, 1]] 16 0
graphs: 106
failed by rank: 80
failed by posdef: 10
type  [3, 4, 4, 4, 5]
graphs: 50
failed by rank: 50
failed by posdef: 0
type  [3, 3, 4, 4, 6]
graphs: 66
failed by rank: 66
failed by posdef: 0
type  [3, 3, 3, 4, 7]
graphs: 48
failed by rank: 48
failed by posdef: 0
type  [3, 3, 4, 5, 5]
graphs: 28
failed by rank: 28
failed by posdef: 0
type  [3, 3, 3, 5, 6]
graphs: 42
failed by rank: 42
failed by posdef: 0
type  [3, 3, 3, 3, 3, 5]
graphs: 20
failed by rank: 0
failed by posdef: 20
type  [3, 3, 3, 3, 4, 4]
graphs: 58
failed by rank: 0
failed by posdef: 58
Graphs (matrices) checked: 5526
(16526, 17)
(16526, 17)
(16526, 17)
(16526, 17)
```