Hungarian Algorithm

In [1]:
```
import opengm
import numpy


numObjects = 7
costs = numpy.random.rand(numObjects,numObjects)
print costs
```

```
[[ 0.22291014  0.86036321  0.30364001  0.62406093  0.27928    ]
 [ 0.27143579  0.66073554  0.74667637  0.07450558  0.48839155]
 [ 0.03011927  0.46634539  0.90776343  0.41867748  0.68479904]
 [ 0.12136733  0.95628486  0.15304729  0.4061785   0.75170059]
 [ 0.26976022  0.86853386  0.81226495  0.28000263  0.56060886]]
```

Soft Constraint GM

In [2]:
```
gm = opengm.gm([numObjects]*numObjects)

unaryIds = gm.addFunctions(costs)
gm.addFactors(unaryIds,numpy.arange(numObjects))

f1To1=opengm.pottsFunction([numObjects,numObjects],10000.0, 0.0)
f1To1Id=gm.addFunction(f1To1)

for x0 in range(numObjects):
    for x1 in range(x0+1,numObjects):
        gm.addFactor(f1To1Id,[x0,x1])
```

Inference with Lazy Flipper

In [3]:
```
Inf = opengm.inference.LazyFlipper
param = opengm.InfParam(maxSubgraphSize=1)
inf = Inf(gm=gm,parameter=param)
inf.infer()
arg = inf.arg()
print arg, gm.evaluate(arg)
print len(numpy.unique(arg))
```

```
[4 3 1 2 0] 1.2429384696
5
```

In [4]:

```
Inf = opengm.inference.LazyFlipper
param = opengm.InfParam(maxSubgraphSize=2)
inf = Inf(gm=gm,parameter=param)
# use warm start
inf.setStartingPoint(arg)
inf.infer()
arg = inf.arg()
print arg, gm.evaluate(arg)
print len(numpy.unique(arg))
```

```
[4 3 1 2 0] 1.2429384696
5
```

In [5]:

```
Inf = opengm.inference.LazyFlipper
param = opengm.InfParam(maxSubgraphSize=3)
inf = Inf(gm=gm,parameter=param)
inf.infer()
# use warm start
inf.setStartingPoint(arg)
arg = inf.arg()
print arg, gm.evaluate(arg)
print len(numpy.unique(arg))
```

```
[4 3 1 2 0] 1.2429384696
5
```

Hard Constraint GM

In [6]:

```
gmB = opengm.gm([numObjects]*numObjects)

unaryIds = gmB.addFunctions(costs)
gmB.addFactors(unaryIds,numpy.arange(numObjects))
```

Out[6]:

```
4L
```

Inference with LpCplex

In [7]:

```
cplexParam = opengm.InfParam(integerConstraint=True)
lpCplex = opengm.inference.LpCplex(gm=gm,parameter=cplexParam)


for x0 in range(numObjects):
    for x1 in range(x0+1,numObjects):

        for label in range(numObjects):
            #lpVarx0  = inf.lpNodeVariableIndex(x0,label)
            #lpVarx1  = inf.lpNodeVariableIndex(x1,label)
            constraintVars  = [1,2]
            constraintCoeff = [1.0,1.0]
            lowerBound = 0.0
            upperBound = 1.0
            lpCplex.addConstraint(constraintVars,constraintCoeff,lowerBound,upperBound)


lpCplex.infer()
arg = lpCplex.arg()
print arg, gm.evaluate(arg)
print len(numpy.unique(arg))
```

```
[4 3 1 2 0] 1.2429384696
5
```