E-402-STFO PROBLEMS FOR MODULE 5

CREATED BY HENNING ULFARSSON

This module is concerned with linear programming and Monte-Carlo integration You get a perfect score for this module by getting 47 points or more.

1. Linear Programming (30 points)

Linear programming is a method for stating some optimization problems and solving them efficiently. Consider for example a company that needs to transport some goods with a truck. The volume in the truck is 10 (cubic meters). The company has the following goods: 4 minks, worth 200 kr. and taking up 3 volume each; 7 hamsters, worth 50 kr. and taking up 1 volume each; 2 cows worth 300 kr. taking up 5 volume each; 9 orcs, worth 250 kr. and taking up 4 volume each; and 11 goats, worth 200 kr. and taking up 3.5 volume each. We create variables for each of the goods: x_m for how many minks we put in the truck, x_h for how many hamsters, etc. We can now state the problem in the following way:
Maximize

$$200 \cdot x_m + 50 \cdot x_h + 300 \cdot x_c + 250 \cdot x_o + 200 \cdot x_g$$

subject to (how much of each we have)

$$0 \leq x_m \leq 4,\ 0 \leq x_h \leq 7,\ 0 \leq x_c \leq 2,\ \ldots$$

and subject to (how much we can fit in the truck)

$$3 \cdot x_m + 1 \cdot x_h + 5 \cdot x_c + \cdots \leq 10$$

Sage can solve these kinds of problems for us:

```
p = MixedIntegerLinearProgram()
v = p.new_variable()

goods  = ['m','h','c','o','g']
worth  = [200,50,300,250,200]
vol    = [3,1,5,4,3.5]
amount = [4,7,2,9,11]

# We want to maximize this:
p.set_objective(sum(v[goods[i]]*worth[i] for i in range(len(goods))))

# We are constrained by how much we have of each:
for i in range(len(goods)):
    p.add_constraint(v[goods[i]] <= amount[i])

# And we are constrained by the space in the truck:
p.add_constraint(sum(v[goods[i]]*vol[i] for i in range(len(goods))) <= 10)

print "What is the worth of the optimal solution?"
print p.solve()
print "------------------------------"
```

Date: Updated October 21, 2013.

```
print "How much of each"
for i in range(len(goods)):
    print goods[i], p.get_values(v[goods[i]])
```

Running the code produces

```
What is the worth of the optimal solution?
666.666666667
-------------------------------
How much of each
m 3.33333333333
h 0.0
c 0.0
o 0.0
g 0.0
```

We can view this as an approximate solution, or we can ask for a solution that uses only integers, at the cost of more time. Set

```
v = p.new_variable(integer = True)
```

and rerun the above code. This produces

```
What is the worth of the optimal solution?
650.0
-------------------------------
How much of each
m 2.0
h 0.0
c 0.0
o 1.0
g 0.0
```

Problem 1 (10 points)**.** Write a function `m5p1(G)` that given a graph[1] `G` uses integer linear programming to find the size of the maximum clique.

```
Input: G = graphs.PetersenGraph()
       type(G)
       <class 'sage.graphs.graph.Graph'>
Run: m5p1(G)
Output: 2
```

Problem 2 (10 points)**.** Write a function `m5p2(G)` that given a graph[2] `G` uses integer linear programming to find the size of the largest independent set.

```
Input: G = graphs.PetersenGraph()
       type(G)
       <class 'sage.graphs.graph.Graph'>
Run: m5p2(G)
Output: 4
```

[1]Undirected, with no double edges and no loops

[2]Still undirected, with no double edges and no loops

Problem 3 (10 points). Write a function `m5p3(U,S)` that given a set `U` and a set `S` of subsets of `U` uses integer linear programming to find lowest number of subsets from `S` that can be used to cover `U`.

```
Input: U = Set([1,2,3,4,5])
       S = Set([Set([1,2,3]), Set([2,4]), Set([3,4]), Set([4,5])])
Run: m5p3(U,S)
Output: 2
```

2. Monte-Carlo Integration (32 points)

Here we experiment with regions of the plane, such as squares and circles. We define regions using predicates. For example, the rectangle with bottom left corner in (x_{min}, y_{min}) and top right corner in (x_{max}, y_{max}) is defined by the predicate

$$P(x, y) = (x_{min} \leq x \leq x_{max}) \text{ and } (y_{min} \leq y \leq y_{max}).$$

Note that $P(x, y)$ is true if and only if (x, y) is a point in that rectangle.

Problem 4 (3 points). Write a function `m5p4(xmin,xmax,ymin,ymax)` that given the integers `xmin,xmax,ymin,ymax` returns a predicate as above

```
Input: xmin,xmax,ymin,ymax = 1,2,3,4
Run: m5p4(xmin,xmax,ymin,ymax)
Output: <function <lambda> at 0x115596c80>
```

Note that running `m5p4(xmin,xmax,ymin,ymax)(2,3)` returns True

Problem 5 (1 points). Write a function `m5p5(x0,y0,r)` that given the integers `x0,y0,r` returns a predicate for a square with center `(x0,y0)` and side-length `2r`.

```
Input: x0,y0,r = 2,3,4
Run: m5p5(x0,y0,r)
Output: <function <lambda> at 0x11559a500>
```

Note that running `m5p5(x0,y0,r)(2,2)` returns `True`.

Problem 6 (3 points). Write a function `m5p6(x0,y0,r)` that given the integers `x0,y0,r` returns a predicate for a disc with center `(x0,y0)` and radius `r`.

```
Input: x0,y0,r = 2,3,4
Run: m5p6(x0,y0,r)
Output: <function <lambda> at 0x11559ac80>
```

Note that running `m5p6(x0,y0,r)(9,9)` returns `False`.

One way of illustrating a region R defined by a predicate, as above, is to randomly "throw darts" at a rectangle containing R and plot the "darts" that hit R. This is achieved by the following function:

```
def hits(P, n=10000, xmin=-1, xmax=1, ymin=-1, ymax=1):
out = []

for i in xrange(n):
    x = random() * (xmax - xmin) + xmin
    y = random() * (ymax - ymin) + ymin

    if P(x, y):
        out.append((x, y))
    return out

D = m5p6(0,0,1) # The unit disk
R = points(hits(D))
R.show()
```

You can use it to plot some of the regions you defined above.

Problem 7 (10 points)**.** Write a function `m5p7(P,n=10000,xmin=-1,xmax=1,ymin=-1,ymax=1)` that given the predicate `P`, as well as the integers `xmin,xmax,ymin,ymax`, returns an approximation for the area covered by `P` in the rectangle `xmin <= x <= xmax`, `ymin <= y <= ymax`.

```
Input: P = m5p6(0,0,1)
Run: m5p7(P)
Output: 3.14000
```

Note that you will get a slightly different answer when you run your own code.

Problem 8 (15 points)**.** Write a function

```
m5p8(f,n=10000,xmin=-1,xmax=1,ymin=-1,ymax=1,zmin=-1,zmax=1)
```

that given the function `f`, as well as the integers `xmin,xmax,ymin,ymax,zmin,zmax`, returns an approximation for the volume between the xy-plane and the surface determined by `f`, inside the box `xmin <= x <= xmax`, `ymin <= y <= ymax`, `zmin <= z <= zmax`.

```
Input: f = x^2 + y^2
Run: m5p7(f)
Output: 2.67144
```

Note that you will get a slightly different answer when you run your own code.

School of Computer Science, Reykjavik University, Menntavegi 1, 101 Reykjavík, Iceland

E-mail address: `henningu@ru.is`