Homework 3, Zachary Shoults

1) Are there problems that are provably NP, not NP-complete, and not P?

Answer:

The ***graph isomorphism problem*** is one of few standard problems in computational complexity theory belonging to NP, but not known to belong to either of its well-known (and, if P ≠ NP, disjoint) subsets: P and NP-complete. It is one of only two, out of 12 total, problems listed in Garey & Johnson (1979) whose complexity remains unresolved, the other being integer factorization. It is however known that if the problem is NP-complete then the polynomial hierarchy collapses to a finite level.[6]

In November 2015, László Babai, a mathematician and computer scientist at the University of Chicago, claimed to have proven that the graph isomorphism problem is solvable in quasi-polynomial time. This work has not yet been vetted.[7][8]

Its generalization, the subgraph isomorphism problem, is known to be NP-complete. ***https://en.wikipedia.org/wiki/Graph_isomorphism***

2) Is it possible to partition the vertices of G into k disjoint cliques?

Answer:

This is a different take on the vertex coloring problem, which is NP-complete. That means this problem is also NP-complete. For any clique, they must contain vertices of all different colors (all adjacent to one another) and for all k cliques, they cannot share any of the same vertices. Another way to state this problem: "Is it possible to vertex-color a graph, G, and find k number of subsets of G where each subset has vertices of different colors, all of those vertices are connected to each other vertices in the subset, and no vertices appear more than once across all subsets (disjoint subsets)?"

3) Is it true that G contains a subgraph equal to H?

Answer:

This is a reduction of the k-sized clique problem, which is NP-complete. We are looking for a subgraph with k number of vertices. The only difference is the solution may or may not be a complete graph. Solving these problems are equally difficult.

Zachary Shoults

4) Is it possible to put all postcards in envelopes?

Answer:

This is like a 2-d bin-packing problem. If we sort all the envelopes and postcards by area, in descending order, we could try to pair envelopes to postcards with a sort-then-first-fit technique.

ASSUMPTION: Postcards and Envelopes are reasonable sizes. That is, the *domain of possible dimensions for postcards is the same as the domain for possible dimensions of envelopes*. The maximum size of postcard is also the maximum size of an envelope. Same for minimum.

We would want to optimize the pairings:

Find all PERFECT fits:

If an envelope and postcard have the same edge-dimensions pair them. Remove them both for future iterations (Perfect fit). Loop this test until no PERFECT matches are found.

Find all GOOD fits:

If an envelope has a greater area than the postcard AND they have at least one edge (width or height) equal [i.e. postcard_width == envelope_height] then they should get paired together. (Good fit) Remove them for future iterations. Loop this test until no GOOD matches are found.

Find all OVERSIZED fits:

If an envelope has a greater area than the postcard AND the width of the postcard is less than an edge of the envelope AND the length of the postcard is less than the other edge of the envelope, then they can be paired together. Because the envelopes and postcards are in order by area, the closest OVERSIZED fits will be found. (Of two postcards, one with area 200in^2 and one with 180in^2, the one with 200in^2 will be put into an envelope of area 300in^2 before the other, minimizing sub-optimal pairings)

At this point, all envelopes that could contain a postcard, have been paired. Empirically, this paired >97% of randomly generated postcards. The runtime is not great, but it's polynomial $O(n^2)$. Code (not cleaned up) is included. I believe it is NOT possible to match all postcards, for any possible set of P postcards and E envelopes.

Zachary Shoults

5) Minimum number of computers that should be infected to spy on all data?

Answer:

This is a vertex cover problem. Solve it using the approximation:

Find the edge where the its endpoint/vertices’ combined degree is maximum. These endpoints initially comprise the subset. Then, of all edges not adjacent to the current subset, find the one where its endpoints/vertices’ combined degree is highest. Add these 2 endpoints/vertices to the subset. Repeat this process until no edges remain that are not connected to the subset. Finding this subset can be done in polynomial time. It may not be the optimal solution, but it will be a viable solution.

Zachary Shoults ------- Code for envelope sorting

```
import random as r
postcards = []
envelopes = []

all_postcards = 10000       # Randomly generate postcards

for x in range (all_postcards):
    width = r.randrange(3,30)               #Domain of postcard sizes are 3inches to 30inches
    height = r.randrange(3,30)
    area = width*height
    postcard = [area, width, height]        #postcard[0]==Area, [1]==Width, [2]==Area
    postcards.append(postcard)

    width = r.randrange(3, 30)              #Domain of envelope sizes are 3inches to 30inches
    height = r.randrange(3, 30)
    area = width*height
    envelope = [area, width, height]        #envelope[0]==Area, [1]==Width, [2]==Area
    envelopes.append(envelope)

envelopes = sorted(envelopes)
postcards = sorted(postcards)   #sorted in decending order by Area
postcards_sent = 0              #every time a pairing is made, this is incremented

def foundPerfectMatches():
    global postcards_sent, envelopes, postcards
    found = False
    for p in postcards:
        for e in envelopes:
            if (p[1] == e[1] and p[2] == e[2]) or (
                    p[1] == e[2] and p[2] == e[1]):  # an arrangement of sides match perfectly
                envelopes.remove(e)
                postcards.remove(p)
                postcards_sent += 1
                # print('perfect  e: {},{}  p: {},{}'.format(e[1], e[2], p[1], p[2]))
                found = True;
                break
    return found

def foundGoodMatches():
    global postcards_sent, envelopes, postcards
    found = False
    for p in postcards:
        for e in envelopes:
            if p[0] < e[0]:
                if p[1] in e[1:] or p[2] in e[1:]:  # one side of postcard perfect fit for a side
of envelope
                    envelopes.remove(e)
                    postcards.remove(p)
                    # print('close  e: {},{}  p: {},{}'.format(e[1], e[2], p[1], p[2]))
                    postcards_sent += 1
                    found = True;
                    break
    return found
```

Zachary Shoults ------- Code for envelope sorting

```
def foundOversizedMatches():
    global postcards_sent, envelopes, postcards
    found = False
    for p in postcards:
        for e in envelopes:
            if p[0] < e[0]:
                if (p[1] <= e[1] and p[2] <= e[2]) or (p[1] <= e[2] and p[2] <= e[1]):  # postcard
can fit in envelope
                    envelopes.remove(e)
                    postcards.remove(p)
                    # print('oversized  e: {},{}  p: {},{}'.format(e[1], e[2], p[1], p[2]))
                    postcards_sent += 1
                    found = True;
                    break
    return found

#PERFECT FITS
perfect_loops = 0;
while(True):                        #find all PERFECTLY sized envelopes + postcard pairs (all
dimensions equal)
    perfect_loops+=1
    found_perfects = foundPerfectMatches();
    if(found_perfects == False):
        if(foundPerfectMatches()==False):   #one last check, overkill
            break
print("perfect loops:  "+str(perfect_loops+1))

#GOOD FITS
good_loops = 0;
while(True):                    #find all GOOD sized envelopes + postcard pairs (one dimension equal)
    good_loops+=1
    found_fits =foundGoodMatches()
    if(found_fits == False):
        if(foundGoodMatches()==False):  #one last check, overkill
            break
print("close loops:  " + str(good_loops+1))

#OVERSIZED FITS
oversize_loops = 0
while(True):                    #find all envelopes that CAN fit the postcard where areas are as close
as possible
    found_fits=False
    oversize_loops+=1
    found_fits=foundOversizedMatches()
    if(found_fits==False):
        if(foundOversizedMatches()==False): #one last check, overkill
            break

print("oversized loops:  "+str(oversize_loops+1))

print("size postcards: {}  size envelopes: {} sentSoFar: {}".format(len(postcards),
len(envelopes), postcards_sent))

print(postcards_sent/all_postcards) #Percentage of postcards sent


for e in envelopes:                 #find any mistakes on the remaining unpaired
    for p in postcards:
        if (p[1]<e[1] and p[2]<e[2]) or (p[2]<e[1] and p[1]<e[2]):
            print("Failure: p {},{}   e {},{}".format(p[1],p[2],e[1],e[2]))
```