MATH 255	Graph Theory Spring '18	Lab 4

Max Cut: Random and Greedy Partitions

Pair Names: _____

At this point you are familiar with several problems related to finding small sets of edges or nodes whose removal disconnects a graph (small node cuts, or small edge cuts).

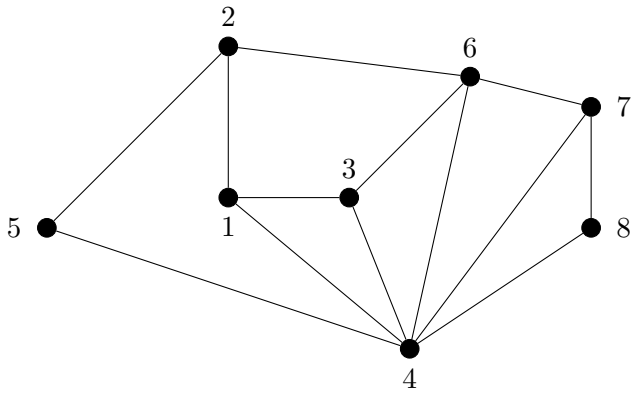
Graph theorists have also studied the problem of finding "large edge cuts" in graphs.

1 The Maximum Cut Problem

Maximum Cut Problem: Given input graph $G = (V, E)$,
Find a *Partition* of the nodes of G into two disjoint subsets V_1 and V_2 (with $V_1 \cup V_2 = V$)
so that the number of edges *across the cut* is maximized.

***An edge "crosses the cut" if it has one endpoint in V_1 and one endpoint in V_2.*
***Say that an edge e is "cut" by a partition if it if e crosses the cut.*

For example, **find the largest cut in the graph you worked with from Lab 3:**

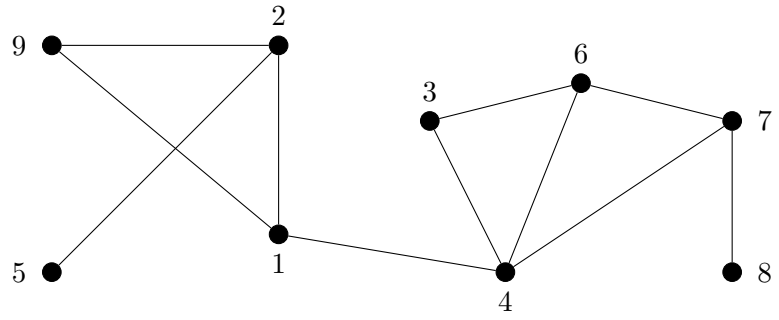


Which nodes will you put in V_1 ? V_2 ? Cut as many edges as possible!

How many edges are in the cut implied by your partition?

Is the best solution unique?
Is there a different partition just as good as the one you found?

Warm up by doing another example:



Which nodes will you put in V_1 ? V_2 ? Cut as many edges as possible!

How many edges are in the cut implied by your partition?

Is the best solution unique?
Is there a different partition just as good as the one you found?

2 A Mystery Graph

Download the file `mysterygraph.m` from Moodle.

Running this file will create an adjacency matrix for a 200-node graph G.
This function has no inputs, but gives the adjacency matrix as output.
You will explore this graph for the remainder of the lab.

To get started:

- Compute the average degree of G:
- Compute the total number of edges in G:
- Print a histogram of the degrees in G.

3 Counting Edges Across a Cut

Now you will build a function that takes as input an adjacency matrix `A`, and two vectors `V1` and `V2`.

`V1` will be a vector of length 200 that contains a 1 to indicate that a node is in V_1 and a 0 otherwise. `V2` is similar. There are other ways you could encode the members of V_1 and V_2, but this choice will make writing the function much easier.

Note: In your partition every node of V is in either V_1 or V_2, so notice that `V1(i)+V2(i)` should be 1 for all i.

Carefully comment on what the following lines of code are doing:

```
function[edgesacross]=countcut(A, V1, V2)

edgesacross=0;
for i=1:200
   for j=1:200
     if ((i>j)&& not(V1(i)==V1(j))&& A(i,j)==1)
       edgesacross=edgesacross+1;
     end
   end
end
```

So far your function isn't using `V2`, but this input may be useful to have around later.

First tests:

- Create inputs for a partition in which V_1 contains nodes $\{1, 2, 3\}$ and V_2 contains nodes all other nodes. Using the `countcut` function, how many edges of the mystery graph are cut? What does you answer mean in terms of G?

  Sample code: `V1=[ones(3,1); zeros(197,1)];`

- Create inputs for a partition in which V_1 contains nodes $\{1, 2, 3, ..., 100\}$ and V_2 contains nodes $\{101, 102, 103, ..., 200\}$. Using the `countcut` function, how many edges of the mystery graph are cut?

  Are you surprised by the change from the previous part?

4 A Random Partition

To find a partition that cuts a lot of edges, one thing you could try is guessing randomly.
(I know this doesn't seem terribly smart...)

Create a script that randomly flips a fair coin for each node
to decide whether it should be in V_1 or V_2,
and then runs `countcut` to find how many edges got cut.

Attach your code with comments about what each line does.

Run your script a couple of times until you feel confident that it is counting the correct quantity.

Now conduct 10 trials:

Sample	Number of Edges Cut
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
Avg. Num. Edges Cut:	

Write 3-4 sentences interpreting your results in terms of the mystery graph G.

5 Local Improvement of a Random Partition

Here is a slightly smarter idea:

1. Start by guessing a partition randomly a few times,
2. Take a moderately well-performing guess of V_1 and V_2 we observe,
3. Consider whether switching any single node to the other V_i increases the count of cut edges.

(If we find a node in V_1 where switching it to V_2 makes a big improvement, we'll switch it, etc).

The idea is to make a small-scale *local improvement*
to a random partition that is already pretty good.

Write a function that takes in `A`, `V1`, `V2`, and reports back on which single node being switched will give the largest improvement. For every node in V_1 your code should try out switching it to V_2 and check how much the number of cut edges increased. Similarly, for every node in V_2 your code should try out switching it to V_1 and check how much the number of cut edges increased. Whichever single node switch gives the largest increase in the number of edges cut should be reported as output.

Here are some ideas to get you started:

```
function[bestnodetoswitch]=localimprove(A, V1, V2)

bestnodetoswitch=0;
biggestimprove=0;

TrialV1=V1;
TrialV2=V2;

Currentcutsize=countcut(A, V1, V2)

for node=1:200
    if (V1(node)==1)
       TrialV1(node)=0;
       TrialV2(node)=1;
       TrialEdgescut= countcut(A,TrialV1,TrialV2);
       if ((TrialEdgescut-Currentcutsize)>biggestimprove)
          bestnodetoswitch=node;
          .............etc.....
```

Attach your code with comments about what each line does.

Now that you have a local improvement method,

1. Run your random guessing code until you get a partition to start from that is at least as good as the average you found in Section 4.

2. Run your local improvement method 10 times:
   After each time, switch the best node to switch as reported by the method, and report the current number of edges cut. Then run the local improvement method on the new partition to gain further improvements.

Partition	Current Num. Edges Cut	best node to switch
A good guess		
1 local improvement		
2 local improvements		
3 local improvements		
4 local improvements		
5 local improvements		
6 local improvements		
7 local improvements		
8 local improvements		
9 local improvements		
10 local improvements		stop!

Write 2-3 sentences interpreting your results.

6 A Greedy Partition

Last, we'll do something that sounds pretty smart.

1. Start from V_1 and V_2 empty.
2. Starting from node 1, for each node x,
greedily choose which of V_1 and V_2 x should be added to.
For each node, based on V_1 and V_2 so far, place x in the V_i that results in more edges crossing the cut.
3. Once all nodes have been assigned to V_1 or V_2, check how many edges cross the cut.

Create code that takes `A` as input and creates a greedy partition as described.

As you evaluate whether it is better to add a node x to V_1 or V_2 is likely that you'll want to consider how many edges are cut in some subgraph of G, composed of some initial run of nodes, e.g. $\{1, 2, 3, 4, 5\}$.

Try out the command `B=A(1:5,1:5)`

Print your code with comments about what each few lines is doing.

Write 2-3 sentences about how your Greedy Strategy performs (how many edges it cut) in comparison to your earlier random guessing and locally-improved random guess.

Finally, can you can make some local improvements
to the greedy strategy produced by your algorithm?

Partition	Current Num. Edges Cut	best node to switch
Greedy Strategy		
1 local improvement		
2 local improvements		
3 local improvements		that's enough...

7 Reflection

Please write **3 short thoughts, ideas, or reflections**
about this family of strategies to find a large cut.

What did you find most surprising? Do you have any conjectures? Other smart ways to find a large cut?