CSE 521 Problem Set 1

Tyler Chen

Problem 1

In this problem you are supposed to implement Kargers min-cut algorithm. I have uploaded three input files to the course website. Each file contains the list of edge of a graph; note that the graphs may also have parallel edges.

For each input file you should output the size and the number of min cuts. Please upload your code to Canvas. You should also write the output of your program for each input in the designated "text box" of Problem 2 in Canvas. There are four inputs uploaded to the course website. The last one, "b3.in" is very large but it has only 10 percent of the grade of this problem.

Solution

We construct a weighted adjacency matrix G and keep track of vertex degrees at each step.

In Karger's algorithm a edge is selected at random uniformly from the edge set. To do this we first select a vertex v_0 out of all vertices, with probability proportional to the vertex degree. We then select a vertex v_1 from the set of vertices attached to v_0 with probability proportional to the edge weight.

We then merge v_1 into v_0 by updating G and deleting any loops which may have formed.

To keep track of which vertices belong to the same supernode we keep a list where the i-th entry tells us which supernode vertex i belongs to. Every time we contract an edge we update this list.

The size of the max cut will be the maximum value of vertex-degrees, and the cut itself can be obtained from vertex-labels.

We implement this in python with a few modifications. First, the reading and writing to G are vectorized. As a result numpy will use its compiled functions to compute these changes. We also update any of the entries of vertex-label which were previously v_1 to v_0. This way the final output of vertex-label will have just two values. We also note that G is symmetric so that we could avoid about half the read/writes to G by using only the upper triangle.

This implementation is fast enough that for the first three data sets we are able to run it $\mathcal{O}(n^2)$ in under a minute to find all min cuts. However, to find a single cut on the final data set takes about 2-3 seconds, so to find $n^2 = 10000^2$ cuts is not tractable.

We therefore implement the Karger-Stein algorithm. At each stage we take the submatrix generated by the nonzero entries of the vertex degree vector as input to the recursive call. We also keep track of all cuts corresponding to the smallest cut found in a given round

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  6 20:17:09 2018

@author: tyler
"""

import numpy as np
import sys

#%%

def karger(G,vertex_label,vertex_degree,iteration_schedule):

    if len(iteration_schedule) > 14:
        print('='*len(iteration_schedule),len(iteration_schedule))

    for n in range(iteration_schedule[0]):
#           if n%1000==0: print('iteration:',n)

        # uniformly at random pick e = (v0,v1)
        cs0 = np.cumsum(vertex_degree)
        rand_idx0 = np.random.randint(cs0[-1])
        v0 = np.searchsorted(cs0,rand_idx0,side='right')

        #cs1 = np.cumsum(np.append(G[e0,e0:],G[:e0,e0]))
        cs1 = np.cumsum(G[v0])
        rand_idx1 = np.random.randint(cs1[-1])
        v1 = np.searchsorted(cs1,rand_idx1,side='right')


        # bring edges from v1 into v0

        # add new edges to v0
        G[v0] += G[v1]
        G[:,v0] += G[v1]
        new_edge_count = vertex_degree[v1] - G[v0,v0] #- G[v1,v1]

        # delete old edges from v1
        G[v1] = 0
        G[:,v1] = 0

        # delete any created loops
        G[v0,v0] = 0

        # update degrees
        vertex_degree[v0] += new_edge_count
        vertex_degree[v1] = 0

        # update supernodes
        np.putmask(vertex_label,vertex_label==v1,v0)

    nz = np.nonzero(vertex_degree)[0]

    if(len(nz) == 2):
        SN = [set(np.where(vertex_label == nz[0])[0]),set(np.where(
            vertex_label == nz[1])[0])]
        if vertex_degree[nz[0]] < 100: # assume min cut is of size  <100
            return [SN],vertex_degree[nz[0]]
        else:
            return [],100

    else:
        cuts_H = []
        size_V_H = len(nz)
        H = G[np.ix_(nz,nz)]
        vertex_label_H = np.arange(size_V_H,dtype='int')

        SN0,v_d_0 = karger(np.copy(H),np.copy(vertex_label_H),
            vertex_degree[nz],iteration_schedule[1:])
        SN1,v_d_1 = karger(np.copy(H),np.copy(vertex_label_H),
            vertex_degree[nz],iteration_schedule[1:])

        # keep all min cuts
        cut_size = min(v_d_0,v_d_1)
        if v_d_0 == cut_size:
            cuts_H += SN0
        if v_d_1 == cut_size:
            cuts_H += SN1

        cuts = []
        for cut in cuts_H:
            # build updated list of supernodes
            SN = [set([]),set([])]
            for i,n in enumerate(nz):
                eq_nodes = np.where(vertex_label == n)[0]
                if i in cut[0]:
                    for j in eq_nodes:
                        SN[0].add(j)
                else:
                    for j in eq_nodes:
                        SN[1].add(j)

            cuts.append(SN)

    return cuts,cut_size

#%%

#python p1.py z N ID
z = sys.argv[1] # 0,1,2,3
N = int(sys.argv[2]) # integer number of runs
ID = sys.argv[3] # output file id

#%%
E_raw = np.loadtxt('b'+str(z)+'.in',dtype='int')

min_E = np.min(E_raw)
E = E_raw - min_E
size_V = np.max(E)+1

G = np.zeros((size_V,size_V),dtype='uint16')
```

```
vertex_degree = np.zeros(size_V,dtype='int')
for e0,e1 in E:
    vertex_degree[e0] += 1;
    vertex_degree[e1] += 1;
    G[min(e0,e1),max(e0,e1)] += 1;
    G[max(e0,e1),min(e0,e1)] += 1;

del(E)
del(E_raw)

vertex_label = np.arange(size_V,dtype='int') # gives index of supervertex
     containg vertex
#%%
c = np.sqrt(2)
iter_schedule = [int(size_V/c**i) for i in range(int(np.floor(np.log(
    size_V/6)/np.log(c))))]
iter_schedule.append(2)
iter_schedule.reverse()
iter_schedule = np.diff(iter_schedule)[::-1]
iter_schedule[-1] += size_V-2 - np.sum(iter_schedule)

#%%
#cuts,cut_size = karger(np.copy(G),np.copy(vertex_label),np.copy(
    vertex_degree),iter_schedule)

#%%
f=open('b'+z+'/cuts_'+ID+'.dat','ab')
g=open('b'+z+'/cut_sizes_'+ID+'.dat','ab')
#
for n in range(N):
    if n%1 == 0:
        print(ID+'_trial :', n+1,' of ',N)
    cuts,cut_size = karger(np.copy(G),np.copy(vertex_label),np.copy(
        vertex_degree),iter_schedule)
    for cut in cuts:
        if len(cut[0]) < len(cut[1]):
            vl = list(cut[0])
        else:
            vl = list(cut[1])
        np.savetxt(f,[np.sort(vl)],fmt='%d',delimiter=',')
        np.savetxt(g,[cut_size],fmt='%d',delimiter=',')

f.close()
g.close()
```

We then post process the output, finding the size of the min cut and then creating a set of all of the min cuts.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 12 07:33:37 2018
```

dataset	min cut size	# of unique min cuts found	total # of min cuts found
b0	8	50	27277
b1	2	74	89101
b2	1	3	68334
b3	100		

Table 1: Summary of Results

```
@author: tyler
"""

import numpy as np

#%%
for z in []:
    cut_count = 0
    last_new_set_index = 0 # ideally this is low compared to the number
        of min cuts meaing we have lots of duplicate cuts
    print('dataset :',z)
    cut_sizes = np.loadtxt('b'+str(z)+'/cut_sizes_t1.dat',dtype='int')
    min_cut_size = np.min(cut_sizes)
    min_cut_locations = np.where(cut_sizes == min_cut_size)[0]

    cuts = set([])
    with open('b'+str(z)+'/cuts_t1.dat') as fd:
        for n, line in enumerate(fd):
            if n%5000==0:
                print(n)
            if n in min_cut_locations:
                cut_count += 1
                cut = frozenset(map(int,line.rstrip().split(',')))
                if cut not in cuts:
                    cuts.add(cut)
                    last_new_set_index = n

    np.savetxt('b'+str(z)+'/cut_stats.txt',[min_cut_size,len(cuts),
        cut_count],fmt='%d')
    print('minimum cut size found: ', min_cut_size)
    print('number of distinct min cuts found: ',len(cuts))
    print('number of min cut found: ',cut_count)
```

Figure 1 shows the result of running our algorithm on the data sets provided.

The total number of min cuts found is provided to give some sense of how likely we are to have found all the cuts.

We note that we did not find a cut of size smaller than 100 on b3 despite running over 100 iterations of Karger-Stein. We also note that we did not save any cuts of size 100 since this required a lot of IO and we did not expect 100 to be the min cut.

Problem 2

Consider the following process for executing n jobs on n processors. In each round, every (remaining) job picks a processor uniformly and independently at random. The jobs that have no contention on the processors they picked get executed, and all other jobs back off and then try again. Jobs only take one round of time to execute, so in every round all the processors are available.

For example, suppose we want to run 3 jobs on 3 processors. Suppose in round 1, jobs 1 and 2 choose the first processor and job 3 chooses the second processor. Then job 3 will be executed and jobs 1 and 2 back off. Suppose in round 2, job 1 chooses the third processor and job 2 chooses the first processor. Then both of them are executed and the process ends in 2 rounds.

In this problem we almost show that the number of rounds until all jobs are finished is $\mathcal{O}(\log\log n)$ with high probability.

(a) Suppose less than $\sqrt{n}$ jobs are left at the beginning of some round. Show that for some constant $c > 0$, with probability at least c no job remains after this round.
(b) In the first round we have n jobs. Show that the expected number of processors that are picked by no jobs is $n(1-1/n)^n$.
(c) Suppose there are r jobs left at the beginning of some round. What is the expected number of processors that are matched to exactly one job? What is the expected number of jobs remaining to be completed after that round?
(d) Suppose in each round the number of jobs completed is exactly equal to its expectation. Show that (under this false assumption) the number of rounds until all jobs are finished is $\mathcal{O}(\log\log n)$.
(e) In this part we almost justify the false assumption. Suppose there are r jobs left at the beginning of some round. Let E be the expected number of processors that are matched to exactly one job. Show that for any $k > 1$, the number of processors with exactly one matched job is in the interval $[E-k\sqrt{r}, E+k\sqrt{r}]$ with probability at least $1-\exp(-\Omega(k^2))$. You can use the McDiarmids inequality to prove the claim.

> **Theorem.** (McDiarmids inequality)
>
> Let $X_1, \ldots, X_n \in \mathcal{X}$ be independent random variables. Let $f : \mathcal{X}^n \to \mathbb{R}$. If for all $1 \leq i \leq n$ and for all $x_1, \ldots, x_n$ and $\tilde{x}_i$,
>
> $$|f(x_1, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, \tilde{x}_i, x_{i+1}, \ldots, x_n)| \leq c_i,$$
>
> then,
>
> $$\mathbb{P}\left[|f(X_1, \ldots, X_n) - \mathbb{E}[f]| \geq \epsilon\right] \leq 2\exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}\right)$$

Solution

(a) Let $X_i = \in \{1, \ldots, n\}$ denote the node that processor i picks. That is, X_i is uniformly distributed with range $\{1, \ldots, n\}$.

It is then clear that this is the birthday paradox. It follows immediately from Lemma 3.1 that the probability of no collisions when less than $\sqrt{n}$ jobs remain is at least one half. If there are no collisions then all jobs terminate in this round.

(b) For a fixed processor j, define,

$$\mathcal{A}_{i\to j} = \text{event that process } i \text{ picks processor } j$$

Then,

$$\mathbb{P}[\mathcal{A}_{i\to j}] = 1/n$$

Note that $\mathcal{A}_{i\to j}$ and $\mathcal{A}_{i'\to j}$ are independent for $i \neq i'$. Therefore,

$$\mathbb{P}[\text{processor } j \text{ never picked}] = \prod_{i=1}^{n} \mathbb{P}[\mathcal{A}_{i\to j}^c] = (1-1/n)^n$$

Let $X_j = \mathbb{1}[\text{processor } j \text{ never picked}]$ so that $X = \sum_j X_j$ gives the number of processors never picked. Then,

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = n\mathbb{P}[\text{processor } j \text{ never picked}] = n(1-1/n)^n$$

(c) For a fixed processor j define,

$$\mathcal{B}_{i\to j} = \text{event that process } i \text{ is the only process picking processor } j$$

Let R be the set of r nodes. Then,

$$\mathcal{B}_{i\to j} = \mathcal{A}_{i\to j} \cap \left(\bigcap_{i' \in R\setminus\{i\}} \mathcal{A}_{i'\to j}^c \right)$$

Again by the independence of $\mathcal{A}_{i\to j}$ and $\mathcal{A}_{i'\to j}$ for $i \neq i'$,

$$\mathbb{P}[\mathcal{B}_{i\to j}] = \mathbb{P}[\mathcal{A}_{i\to j}] \left(\prod_{i' \in R\setminus\{i\}} \mathbb{P}[\mathcal{A}_{i'\to j}^c] \right) = (1/n)(1-1/n)^{r-1}$$

Therefore, since $\mathcal{B}_{i\to j}$ and $\mathcal{B}_{i'\to j}$ are independent for $i \neq i'$,

$$\mathbb{P}[\text{processor } j \text{ has exactly one process}] = \sum_{i\in R} \mathbb{P}[\mathcal{B}_{i\to j}] = \frac{r}{n}(1-1/n)^{r-1}$$

Let $Y_j = \mathbb{1}[\text{processor } j \text{ has exactly one process}]$ so that $Y = \sum_j Y_j$ gives the number of processors with exactly one process.
Then,

$$\mathbb{E}[Y] = \sum_{j=1}^{n} \mathbb{E}[Y_j] = n\mathbb{P}[\text{processor } j \text{ has exactly one process}] = r(1-1/n)^{r-1}$$

Therefore, since a job is run only if it is the only job on a processor,

$$\mathbb{E}[\text{number of jobs remaining}] = r - r(1-1/n)^{r-1}$$

(d) Let r_k be the number of steps at step k. We seek k such that $r_k = 0$ starting with $r_0 = n$. By the previous result, r_k is defined recursively as,

$$r_{k+1} = r_k - r_k(1 - 1/n)^{r_k - 1}$$

It is well known that for $b > 1$ and $a > 0$ sufficiently small,

$$(1-a)^b \geq 1 - ab$$

Therefore, for sufficiently large n and since $r_k \leq r_k^2$,

$$r_{k+1} \leq r_k - r_k\left(\frac{1 - r_k/n}{1 - 1/n}\right) = \frac{r_k(r_k - 1)}{n-1} \leq \frac{r_k^2 - r_k}{n} \leq \frac{2}{n} r_k^2$$

Using Mathematica we can solve recursive relationship $a_{k+1} = 2a_k^2/n$ with $a_0 = n/4$. This has explicit solution,

$$a_k = n \cdot 2^{-(1+2^k)}$$

We find k such that $a_k = 1$. We have,

$$-(1+2^k)\log(2) = \log\left(2^{-(1+2^k)}\right) = \log(1/n) = -\log(n)$$

Therefore,

$$2^k = \log(n)/\log(2) - 1$$

Finally,

$$k = \log\left(\frac{\log(n)}{\log(2)} - 1\right) = \mathcal{O}(\log\log(n))$$

We now justify that the convergence of r_k is at least as fast as a_k.

Obviously once $r_k = 1$, the algorithm will terminate in one step (constant time). More importantly, for sufficiently large n, $r_k \leq n/4$ in $\mathcal{O}(1)$ operations.

Note that $\lim_{n\to\infty}(1 - 1/n)^{n-1} = 1/e \geq 1/4$. Therefore,

$$\lim_{n\to\infty} \frac{n - n(1-1/n)^{n-1}}{(3/4)n} = \frac{4}{3}\left(1 - \frac{1}{e}\right) < 1$$

This shows that for sufficiently large n less than $3/4$ of the jobs remain after the first step.

As r decreases the fraction of the jobs terminating each step increases, since $r - r(1 - 1/n)^{r-1}$ is a increasing convex function of r. Therefore, in the first 5 iterations, at least $(3/4)^5 < 1/4$ of the jobs terminate.

In summary, r_k will reach $n/4$ is constant time. From this point r_k converges at least as fast as a_k, which converges like $\log\log n$. This convergence continues until $r_k = 1$, in which case the algorithm terminates in the next step. Therefore, the overall convergence is $\log\log n$.

(e) Define f by,

$$f(x_1,\ldots,x_n)=\sum_{i=1}^{n}\mathbb{1}[\forall j\in\{1,\ldots,n\}\setminus\{i\}:x_i\neq x_j]$$

That is, f counts how many of the x_i are unique.

Then for any $x_1,\ldots,x_n$ and $\tilde{x_i}$, f satisfies,

$$|f(x_1,\ldots,x_n)-f(x_1,\ldots,x_{i-1},\tilde{x}_i,x_{i+1},\ldots,x_n)|\leq 2$$

Let X_i be the index of the processor process i picks. Therefore the X_i are independent and $X=f(X_1,\ldots,X_N)$ is the total number of processors with exactly one job. Finally,

$$E=\mathbb{E}[f(X_1,\ldots,X_N)]$$

By McDiarmids inequality,

$$\mathbb{P}[|X-\mathbb{E}[X]|\geq k\sqrt{r}]\leq 2\exp\left(\frac{-2rk^2}{\sum_{i=1}^{n}2}\right)=2\exp\left(\frac{-2rk^2}{2n}\right)=\exp\left(\frac{-2rk^2}{2n}+\ln(2)\right)$$

Therefore, for fixed r and n,

$$\mathbb{P}[X\in[E-k\sqrt{r},E+k\sqrt{r}]]\geq 1-\exp\left(\frac{-2rk^2}{2n}+\ln(2)\right)=1-\exp(-\Omega(k^2))$$

Problem 3

Given a graph $G = (V, E)$ with $n = |V|$ vertices, let k be the size of the minimum cut of G. In this problem we show that if k is large enough we can down sample G and preserve the size of the min-cut.

Let H be a subgraph of G defined as follows: For every edge e of G include e in H with probability $p = (12 \ln n)/(k\epsilon^2)$. If we include e in H we weight it by $1/p$.

We show that the (weighted) min-cut of H is at least $k(1-\epsilon)$ with probability at least $1 - 1/n$. Note that H has only $p|E|$ many edges (in expectation). So, H has significantly less edges than G if $k \gg \ln(n)/\epsilon^2$.

(a) For a cut $(S, \overline{S})$, let

$$w_H(S, \overline{S}) = \frac{|H(S, \overline{S})|}{p}$$

be the sum of the weights of all edges of H across the cut. Show that for any set $S \subset V$,

$$\mathbb{E}\left[w_H(S, \overline{S})\right] = |E(S, \overline{S})|$$

(b) Use Chernoff bound to show that for any set $S \subset V$,

$$\mathbb{P}\left[w_H(S, \overline{S}) < (1-\epsilon)|E(S, \overline{S})|\right] \leq e^{-6 \ln n |E(S,\overline{S})|/k} = n^{-6|E(S,\overline{S})|/k}$$

(c) Recall that in class we proved that each graph has at most $\binom{n}{2}$ many min-cuts. We say a cut of G is an α-min-cut if it is of size at most α times the size of the min cut, i.e. at most αk many edges. It follows by an extension of the Kargers algorithm that any graph G has at most $n^{2\alpha}$ α-min cuts. Use union bound (and the latter fact) to show that for any integer $\alpha \geq 1$, with probability at least $1 - 1/n^2$, for all sets S where $\alpha k \leq |E(S, \overline{S})| \leq 2\alpha k$,

$$w_H(S, \overline{S}) \geq (1-\epsilon)|E(S, \overline{S})|.$$

(d) Use another application of union bound to show that with probability at least $1 - 1/n$, for all sets $S \subset V$,

$$w_H(S, \overline{S}) \geq (1-\epsilon)|E(S, \overline{S})|.$$

Solution

For notational convenience we define $E_S = E(S, \overline{S})$. Note also that when we say $S \subset V$ we generally mean $S \subset V$ for which $S \neq \emptyset$ and $S \neq V$. That is, $S \in 2^V \setminus \{\emptyset, V\}$.

(a) Let $S \subset V$. Not that $|H(S, \overline{S})|$ is the number of edges of G contained in the edge set of H after (randomly) down sampling. We can think of $|H(S, \overline{S})|$ as a random variable defined by,

$$|H(S, \overline{S})| = \sum_{e \in E_S} X_e$$

where the X_e are iid Bernouli random variables with success probability p.

Clearly each such variable indicates whether or not edge e is contained in the edge set of H.

Then,

$$\mathbb{E}\left[w_H(S,\overline{S})\right] = \frac{1}{p}\mathbb{E}\left[|H(S,\overline{S})|\right] = \frac{1}{p}\left(\sum_{e\in E_S}\mathbb{P}[e\in H]\right) = \frac{1}{p}|E(S,\overline{S})| = |E(S,\overline{S})|$$

(b) Suppose Y_i are independent and Bernouli distributed (not necessarily identically). Let $Y=\sum_{i=1}^{N} Y_i$. Then the Chernoff bound states,

$$\mathbb{P}(Y \leq (1-\epsilon)\mathbb{E}[Y]) \leq \exp\left(\frac{-\mathbb{E}[Y]\epsilon^2}{2}\right)$$

Fix some $S\subset V$. We can take $Y_i = X_e$, implicitly using an abritrary bijection between $i\in\{1,\ldots,N\}$ and $e\in E(S,\overline{S})$. Then $Y = p\, w_H(S,\overline{S})$ so,

$$\mathbb{E}[Y] = p\,\mathbb{E}[w_H(S,\overline{S})] = p\,|E(S,\overline{S})|$$

Applying this bound gives,

$$\begin{aligned}\mathbb{P}[w_H(S,\overline{S}) \leq (1-\epsilon)|E(S,\overline{S})|] &= \mathbb{P}[p\, w_H(S,\overline{S}) \leq (1-\epsilon)p\,|E(S,\overline{S})|]\\ &\leq \exp\left(\frac{-p\,|E(S,\overline{S})|\,\epsilon^2}{2}\right)\\ &= \exp\left(\frac{-6\ln n\,|E(S,\overline{S})|}{k}\right)\end{aligned}$$

(c) Fix an integer $\alpha \geq 1$ and for notational convenience define,

$$2^V_\alpha = \{S\subset V : |E(S,\overline{S})| \in [\alpha k, 2\alpha k]\}$$

Note that by the extension of Karger's algorithm, the maximum possible number of 2α min cuts is $n^{4\alpha}$ so $|2^V_\alpha| \leq n^{4\alpha}$.

Let $\mathcal{E}_S$ be the event that for a fixed $S\in 2^V$,

$$\{w_H(S,\overline{S}) \leq (1-\epsilon)|E(S,\overline{S})|\}$$

For $S\in 2^V_\alpha$ we have $|E(S,\overline{S})|\in[\alpha k, 2\alpha k]$ so,

$$\mathbb{P}[\mathcal{E}_S] \leq n^{\frac{-6|E(S,\overline{S})|}{k}} \leq n^{\frac{-6(\alpha k)}{k}} = n^{-6\alpha}$$

Now note that $\mathcal{E}^\alpha = \cup_{S\in 2^V_\alpha}\mathcal{E}_S$ is the event,

$$\left\{\exists S\in 2^V_\alpha : w_H(S,\overline{S}) \leq (1-\epsilon)|E(S,\overline{S})|\right\}$$

Therefore, using union bound,

$$\mathbb{P}[\mathcal{E}^\alpha] = \mathbb{P}\left[\bigcup_{S\in 2^V_\alpha} \mathcal{E}_S\right] \leq \sum_{S\in 2^V_\alpha} \mathbb{P}[\mathcal{E}_S] \leq \sum_{S\in 2^V_\alpha} n^{-6\alpha} = |2^V_\alpha|\, n^{-6\alpha}$$

Using the fact that $|2^V_\alpha| \leq n^{4\alpha}$,

$$\mathbb{P}\left[\mathcal{E}^\alpha\right] \leq |2^V_\alpha|\, n^{-6\alpha} \leq n^{4\alpha}\, n^{-6\alpha} = n^{-2\alpha}$$

Therefore, since $\alpha \geq 1$,

$$\mathbb{P}\left[\forall S \subset 2^V_\alpha : w_H(S,\overline{S}) \geq (1-\epsilon)|E(S,\overline{S})|\right] = \mathbb{P}\left[\,\overline{\mathcal{E}^\alpha}\,\right] \geq 1 - \frac{1}{n^{2\alpha}} \geq 1 - \frac{1}{n^2}$$

(d) *Intuition*: We will partition $S \subset V$ into sets $[\alpha k, 2\alpha k]$ for varying α. Applying union bound to the union of the events that S is in a given set and $w_H(S,\overline{S})$ is the right size will give the result.

Note that we must bound the size of this union, and therefore the number of intervals needed. In particular, we must bound $|E(S,\overline{S})|$. If G has parallel edges, there is no bound based on just n and k for $|E(S,\overline{S})|$. To see this consider an example graph where there are three vertices $1,2,3$ and one edge $\{1,2\}$ and m parallel edges $\{2,3\}$. Then $n=3$, $k=1$, but the max cut is m which is arbitrary. We need that the number of intervals required is $\mathcal{O}(n)$.

We therefore make the assumption that the max cut is $2^{(n+1)k}$ so that,

$$k \leq |E(S,\overline{S})| \leq 2^{(n+1)k}$$

Now observe,

$$\bigcup_{i=0}^{n}\left[2^{ik}, 2^{(i+1)k}\right] = [2^0, 2^{(n+1)k}]$$

so that for any $S \subset V$,

$$\exists\, i \in \{1,\ldots,n\} \text{ such that } |E(S,\overline{S})| \in [2^{ik}, 2^{(i+1)k}]$$

Again let,

$$\mathcal{E}^\alpha = \left\{\exists S \in 2^V_\alpha : w_H(S,\overline{S}) \leq (1-\epsilon)|E(S,\overline{S})|\right\}$$

Now note that,

$$\mathcal{E} = \bigcup_{i=0}^{n} \mathcal{E}^{2^i} = \left\{\exists S \subset V : w_H(S,\overline{S}) \leq (1-\epsilon)|E(S,\overline{S})|\right\}$$

Therefore, using union bound and the result from (c),

$$\mathbb{P}[\mathcal{E}] = \mathbb{P}\left[\bigcup_{i=0}^{n} \mathcal{E}^{2^i}\right] \leq \sum_{\alpha=1}^{n} \mathbb{P}[\mathcal{E}^{2^i}] \leq \sum_{\alpha=1}^{n} \frac{1}{n^2} = \frac{n}{n^2} \leq \frac{1}{n}$$

Therefore,

$$\mathbb{P}\left[\forall S \subset V : w_H(S,\overline{S}) \geq (1-\epsilon)|E(S,\overline{S})|\right] = \mathbb{P}[\overline{\mathcal{E}}] \geq 1 - \frac{1}{n}$$

Remark: If we have a better bound for the max cut we can improve the bound. For instance, if there are no parallel edges we can lazily bound the max cut by n^2 and find a bound of nearly $1-1/n^2$. Moreover, if use the bound of $1/n^{2\alpha}$ for $\mathbb{P}[\mathcal{E}^\alpha]$ then a better bound can be obtained.

Problem 4

In lecture 4 we discussed the pairwise independent hash functions. We say that for a prime p we can generate a pairwise independent hash functions by choosing a, b independently from the interval $\{1,\ldots,p-1\}$ and using $ax+b$ as a random number. Suppose we generate t pseudo random numbers this way, $r_1,\ldots,r_t$ where $r_i = ai+b$. We want to say this set is far from being mutually independent. Consider the set $S=\{p/2,\ldots,p-1\}$ which has half of all elements. Prove that with probability at least $\Omega(1/t)$ none of the pseudo-random-numbers are in S. Note that if we had mutual independence this would have been $1/2^t$.

Solution

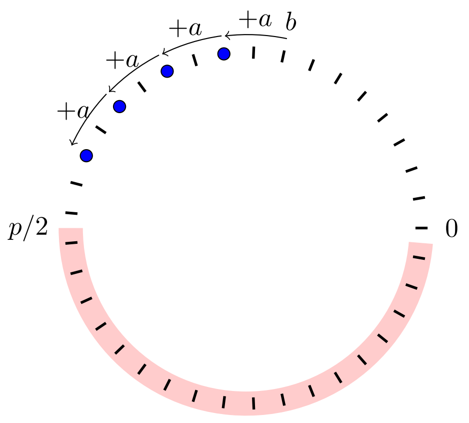
First note that r_i are elements of the finite field $\mathbb{Z}_p$ and $a\neq 0$ so that a is (uniquely) invertible. Then,

$$r_i - r_j = a(i-j) = 0 \qquad\iff\qquad i=j$$

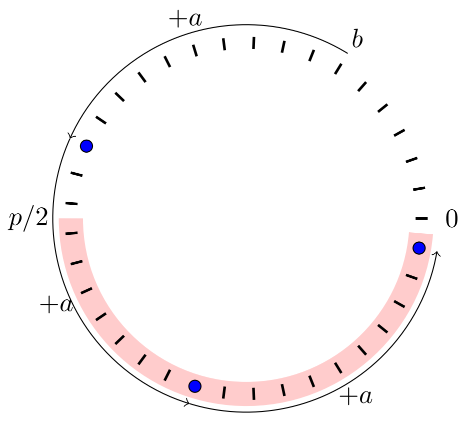
Therefore, by the pigeon hole principle, if $t > p/2$, then there is some r_i contained in S. That is to say, the problem only makes sense when t is much smaller than p. We therefore consider the case where $t\to T$ and $T \ll p$.

Intuition: If a is near 0 (where by near we mean that $p-1$ and 1 are equidistant to zero) then the r_i will be closely clustered. Therefore, if b is far from S (i.e. near $p/4$) we do not expect any of the r_i to be in S.

Figure 1a shows an example of a choice of a and b for which $t=4$ and in fact $t=5$ produce "random numbers" not in S (highlighted in red). Similarly, Figure 1b shows a choice of a and b for which even $t=2$ produces some "random numbers" in s.



(a) $p=37, a=2, b=8, t=4$



(b) $p=37, a=10, b=6, t=3$

We now prove this formally. Given that $a,b\in\{1,\ldots,p-1\}$ there are $(p-1)^2$ values of a and b. For a fixed $t<T\ll p$ we give a bound on how many of these pairs of a and b will produce numbers $r_1,\ldots,r_t$ not in S.

Define $\|\cdot\| : \mathbb{Z}_p \to \mathbb{N}_{\geq 0}$ to be the absolute distance to 0 in $\mathbb{Z}_p$. That is,

$$\|x\| = \min_{k\in\mathbb{Z}} |i(x) - pk|$$

where $i : \mathbb{Z}_p \to \mathbb{R}$ is the natural inclusion map and $|\cdot|$ is taken over $\mathbb{R}$.

With this notation, $\|1\| = \|p-1\| = 1$, $\|2\| - \|p-2\| = 2$ etc.

Fix $t \ll p$ and define $A = \{a : \|a\| < p/(2(t-1))\}$. Then all of the a which do not produce a collision with S are contained in A.

To see this observe that if $\|a\| > p/(2(t-1))$ that one of the r_i will be in S regardless of b since to get from r_1 to r_t, passing in order through each of the other points $r_2, \ldots, rt-1$ we must pass over about $\|a\|(t-1) > p/2$ points. This means that at least some of them are in S.

Suppose that $a \in A$. The range spanned by the r_i is of size $\|a\|(t-1)$. There are about $(p+1)/2$ elements not in S, so there are $(p+1)/2 - \|a\|(t-1)$ possible values of b for which none of the r_i are in S.

Therefore, the total number of pairs a, b for which none of the r_i are in S, denoted N satisfies,

$$N \geq \sum_{a\in A} \left(\frac{p}{2} - \|a\|(t-1)\right) = 2\sum_{a=1}^{\lceil p/(2(t-1))\rceil} \left(\frac{p}{2} - a(t-1)\right) \geq 2\left\lceil \frac{p}{2(t-1)}\right\rceil \frac{p}{2}$$

The probability of picking such a and b is,

$$\mathbb{P}\left[\frac{\text{number of } a, b \text{ that do not produce collision with } S}{\text{number of possible choices of } a, b}\right] = \frac{N}{(p-1)^2} = \Omega\left(\frac{1}{t}\right)$$

Extra Credit

Say we have a plane with n seats and we have a sequence of n passengers $1, 2, \ldots, n$ who are going to board the plane in this order and suppose passenger i is supposed to sit at seat i. Say when 1 comes they chooses to sit at some arbitrary other seat different from 1. From now on, when passenger i boards, if their seat i is available they sit at i, otherwise they choose to sit at a uniformly random seat that is still available. What is the probability that passenger n sits at their seat n?

Solution

When passenger i boards, all the seats of index j, $1 < j < i$ must be filled by the boarding rules (if seat j were empty, then person j did not take their seat when they could have). There are $i-2$ such seats, and $i-1$ passengers already seated. Therefore, if someone is in seat one, seat i must be empty and person i (and, by induction, everyone behind them) can sit in their own seat.

We therefore make the key observation that the success or failure of the process is fixed once a person sits in seat 1 or n (by success we mean person n sits in seat n, and by failure we mean they do not). Moreover, before the n-th passenger boards, someone must have sat in at least of of seat 1 or seat n. Therefore we need only consider the boarding process until seat 1 or seat n is taken.

If person 1 sits in seat n, then person n will sit in seat n with probability zero.

Suppose person 1 does not sit in seat n. By construction, if seats 1 and n are available (i.e. the success or failure of the process has not been determined), the probability that any person sits in seat 1 is the same as the probability that they sit in seat n.

Therefore, given that person one does not sit in seat n, the probability that seat 1 is filled before seat n is the same as the probability that seat n is filled before seat 1. As mentioned above, these events correspond exactly to person n sitting or not sitting in seat n, and the union of the events is the entire probability space.

If person 1 does not sit in seat n, then person n will sit in seat n with probability 1/2.