Instructions. Collaboration is permitted in groups of size at most three. You must write the names of your collaborators on your solutions and must write your solutions alone.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>95</td>
<td></td>
</tr>
</tbody>
</table>

1. All Pairs Resistances

Throughout this problem let $G = (V, E, c)$ be an undirected connected graph with edge conductances being $c_e$ with $n$ edges and $m$ vertices. Let $B$ be the $n$ by $m$ boundary matrix and $C$ the diagonal matrix of conductances. Thus the graph Laplacian is $L = BCB^T$. Let $\chi_u$, where $u \in V$, be the characteristic column of $u$.

1. In class we introduced the Spectral theorem that said that $L$ could be written as $L = U^T \Lambda U$ where each row of $U$ is an eigenvector, $UU^T = I$ and $\Lambda$ is a diagonal matrix of eigenvalues of $L$. Define the pseudoinverse of $\Lambda$

$$\Lambda_i^+ = \begin{cases} \Lambda_i^{-1} & \text{if } \Lambda_i \neq 0 \\ 0 & \text{Otherwise} \end{cases}$$

We define $L^+ = U^T \Lambda^+ U$

Prove the following Moore-Penrose pseudoinverse properties of our inverse:

(a) $LL^+L = L$

(b) $L^+LL^+ = L^+$

(c) $LL^+ = L^+L$

2. Explain why the effective resistance is $R_{uv} = (\chi_u - \chi_v)^T L^+ (\chi_u - \chi_v)$.

3. Consider the linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$ given by $C^{1/2}B^TL^+$. Show how this map takes vectors $\chi_u$ to points $(C^{1/2}B^TL^+)\chi_u$ in $\mathbb{R}^m$ such that, for any two vertices $u$ and $v$, their new Euclidean pairwise square distance equals their effective resistance in $G$. 

(30)
4. Explain how Johnson Lindenstrauss can be used to further map the points into $d = O(\log n)$ dimensions so that pairwise squared distances are approximately preserved.

- Note that we cannot check whether every pair of points has their distance preserved within an $\epsilon$ factor, since doing so already takes $\Omega(n^2)$ time! Therefore, for this problem, assume that we got lucky and all distances are indeed preserved.
- It turns out that, while the lecture only proves that all distances are preserved with probability $1/2$, this probability can be increased to $1 - 1/n^c$ for any constant $c$ by setting $d := C\epsilon^2 \log n = O(\log n)$ for large enough constant $C$, making the failure probability virtually zero. It may be helpful, although not required for the problem, to convince yourself that this is true.

5. Let $\Phi$ be the $d$-by-$n$ Johnson Lindenstrauss matrix from part 4.
Suppose we are given a subprocedure that, given any vector $u \in \mathbb{R}^n$, computes the vector $L^+u$ (or, equivalently, $u^T L^+$). Explain why one needs only $d$ calls to this subprocedure to determine the $d$ by $n$ matrix $M := \Phi C\epsilon^2 B^T L^+$ such that for any two vertices $u$ and $v$, we have $(1 - \epsilon)\|M\chi_u - M\chi_v\|^2 \leq R_{uv} \leq (1 + \epsilon)\|M\chi_u - M\chi_v\|^2$.
In other words, $M$ maps the vectors $\chi_u$ into vectors $M\chi_u$ in $d$ dimensions such that $R_{uv}$ is approximately the squared distance of $M\chi_u$ and $M\chi_v$.

- Note that we don’t know $L^+$ explicitly: computing $L^+$ explicitly requires at least $\Omega(n^2)$ to even write down the matrix $L^+$, so we cannot afford to do that.
- It is known how to compute $L^+u$ fast (i.e. near-linear time) by solving the Laplacian system $Lx = u$ for $x$. Of course, for this problem, assume that such a subprocedure is given.

(30) 2. Planted Cut Problem

In this problem we will construct a graph with a small cut and we will see if the approximate random walks with resets algorithm can find it.

Let $G$ consist of two disjoint cliques: a clique $K_t$ where each edge has weight $1/t$ and a clique $K_n$ where each edge has weight $1/n$. Connecting these two cliques is the complete bipartite graph $K_{t,n}$ where each edge has weight $\delta/n$ for some $0 < \delta < 1$.

Here we will run a random walk with resets with the reset probability $\alpha$.

1. What is the conductance of the cut $(K_t, G \setminus K_t)$ (i.e. the cut separating $K_t$ from the rest of the graph)?

2. Let $u$ be a vertex in $K_t$. Compute the stationary distribution $P_u$ for a random walk with reset probability $0 < \alpha < 1$ resetting to $u$. You may pick $\delta = 1/4$ throughout this problem. You may use a linear algebra solver such as Wolfram-Alpha to compute the stationary distribution. You may need to substantially simplify the calculation before running a solver.
Hints:
(a) How many distinct values will \( P_u \) have?
(b) You may assume that \( t \) is big and that \( n \) is much larger than \( t \). In other words, you can assume things like \( \frac{t}{T} \approx 1 \) and \( t/n \approx 0 \), in order to simplify your calculation. Your values of \( P_u \) should only depend on \( \alpha \) and \( t \).

3. Approximately what value should you set \( \alpha \) to maximize the difference between values of \( P_u \) in \( K_t \) and \( K_n \)? In other words, find the value of \( \alpha \) that maximizes \( P_v - P_w \) for any \( v \in K_t \setminus u \) and \( w \in K_n \). As a sanity check, convince yourself that \( P_v = P_w = 0 \) when \( \alpha = 0 \) or \( \alpha = 1 \). You may use Wolfram-Alpha here.

4. Suppose we call the subroutine \texttt{ApproxPainting}(u, \alpha, \epsilon) and obtain a vector \( \tilde{P} \) that approximates the stationary distribution \( P \) within an additive error \( \epsilon \) (i.e. \( |\tilde{P}_v - P_v| \leq \epsilon \) for each vertex \( v \)). For the optimal alpha from part 3, find a small enough \( \epsilon \) such that it is possible to distinguish vertices in \( K_t \) from those in \( K_n \) using the values in \( \tilde{P} \).

3. (30) Parallel Closest Pair

In this problem we will construct a parallel version of the randomized closest pair algorithm using hashing given in class.

Recall at each round of the algorithm it constructs a hash table \( H_\alpha \) for a grid \( G_\alpha \) knowing that there is a pair whose distance is at most \( \alpha \).

1. Show that the expected number of grids and hash tables constructed throughout the life of the algorithm is \( O(\log n) \).

2. Let’s now consider one round of the algorithm, for a fixed \( \alpha \). Suppose that we have already randomly permuted the vertices (in parallel). Our goal is to find the first point in the random ordering whose insertion brings the minimum distance below \( \alpha \) (or conclude that such a point doesn’t exist). Observe that finding the first is important, since that is what allows the backwards analysis to go through. However, finding the first such point in parallel is non-trivial, since first is an inherently sequential concept. For now, let’s solve a slightly easier problem: given a set of points, determine if any pair of points has distance less than \( \alpha \).

To do this, we need to hash the points in \( H_\alpha \) in parallel. You may assume a parallel hashing algorithm that does the following:

• Input: A series of key-value pairs. The keys are the boxes which form the hash table, and the values are the points themselves.

• Output: If there exists a key (box) with 5 or more input key-value pairs with this key, output “failure”. Otherwise, outputs a list of at most 4 points for each key (box).

• Time: Expected \( O(\log n) \) time in parallel using \( O(n/\log n) \) processors.

Assuming such a parallel hashing algorithm, find out whether there exists a pair of points less than \( \alpha \) apart, and output such a pair if one exists. Your algorithm should run in \( O(\log n) \) time using \( O(n) \) processors.
3. Using the algorithm from part 2, find the first point whose insertion decreases the minimum distance to below $\alpha$. Your algorithm should run in $O(\log^2 n)$ time.
   Hint: run multiple calls of the algorithm from part 2 using a different set of points each time.

4. Back to the original problem: give an algorithm to find the closest pair of points in $O(\log^3 n)$ time.

5. [BONUS] Define the work of an algorithm to be the sum of running times over all processors. Note that processors may not be running at all times, so total work can be much less than the upper bound of (number of processors) $\times$ (parallel time). A closer analysis shows that the algorithm from part 2 is $O(n)$ work, and the entire algorithm from part 4 is $O(n \log^2 n)$ work. The goal of this problem is to improve the total work by another $\log n$ factor, to $O(n \log n)$. You may need to modify your algorithm to achieve this.
   Hint: one way to achieve this is to improve the $O(n \log n)$-work algorithm in part 3.

(5) 4. **Time Spent**

How much time did you spend on this problem set? If you can remember the breakdown, please report this per problem. A break down into time spent solving each problem and typing up your solution would also be useful.