Part III: Literature

Due to the fact my MATLAB broke down many times, I had to use different languages for one algorithm.

Problem 3.1

1. Product Approximation.

$$P(\mathbf{V}) \approx P_t(\mathbf{V}) = \prod_{i=2}^{N} P\left(V_i | V_{r(i)}\right)$$

2. Definition of Mutual Information. (It's not an assumption, but I'd like to list it for its significance.)

$$I(V_i, V_j) = \int_{V_i} \int_{V_j} p(v_i, v_j) ln(\frac{p(v_i, v_j)}{p(v_i)p(v_j)}) dv_i dv_j$$

3. Independence of the current injections.

$$I_i \perp I_j, 2 \leq i \leq n-1 \ and \ j = n, m$$

Assumption one **Product Approximation** makes finding the grid topology equivalent to finding the conditional distribution $P_t(\mathbf{V})$ that best approximate the joint distribution $P(\mathbf{V})$.

By introducing the idea of **MI**, we have a measurement for how similar the joint distribution of two random variables, $p\left(V_i, V_j\right)$, is to the products of the individual distributions, $p\left(V_i\right) p\left(V_j\right)$.

If we can prove the conditional independence of voltages, then Chow-Liu Algorithm can be applied. Now, the Assumption of **Independence of the current injections**, with the actual condition of (n+1)-bus system, we can prove the voltages are conditional independent.

Finally, we can use **Chow-Liu Algorithm** to fine the maximum weight spanning tree.

Problem 3.2

In this part, we first calculate for $v_i(t)$ and the mutual information.

3.2.1

This part is the main porgram for calculation.

```
clc;

v = zeros(1344, 26);

for i = 1: 1344
    for j = 1: 26
        v(i, j) = abs( v_vec(i,j) )+ exp(theta_vec(i,j)*sqrt(-1)); %* exp(
            sqrt(-1) * theta_vec(i,j));
    end
end

MI = zeros(26, 26);

for i = 2: 26
    for j = i: 26
        fprintf("---\n\n", j);
        MI(i,j) = MI_vol(v(:, i), v(:, j), 1344);
        fprintf("%d,%d,%d\n", i, j, MI(i,j));
        MI(j,i) = MI_vol(v(:, j), v(:, i), 1344);
        fprintf("%d,%d,%d\n", i, j, MI(j,i));
    end
end
```

main.m

3.2.2

This part is the MI function used in main.m.

```
function [MI] = MI_vol(u1,u2,n)% Calculate the mutual information of two
    vector

x = [u1, u2];
[xrow, xcol] = size(x);
bin = zeros(xrow,xcol);
pmf = zeros(n, 2);
for i = 1:2
    minx = min(x(:,i));
    maxx = max(x(:,i));
    binwidth = (maxx - minx) / n;
    edges = minx + binwidth*(0:n);
    histcEdges = [-Inf edges(2:end-1) Inf];
    [occur,bin(:,i)] = histc(x(:,i),histcEdges,1);
    pmf(:,i) = occur(1:n)./xrow;
end% Calculate the joint probability density of u1 and u2
jointOccur = accumarray(bin,1,[n,n]);
jointPmf = jointOccur./xrow;
Hx = -(pmf(:,1))'*log2(pmf(:,1)+eps);
Hy = -(pmf(:,2))'*log2(pmf(:,2)+eps);
Hxy = -(jointPmf(:))'*log2(jointPmf(:)+eps);
MI = Hx+Hy-Hxy;
```

MI_vol.m

3.2.3

After the two steps, we have the MI to further search for maximum weight spanning tree algorithm.

```
int findRoot(int x) // non-routine compress
{
    if (eHat[x] == -1)
        return x;
    else
        return findRoot(eHat[x]);
}

int main() {
    init();

    sort(edge + 1, edge + (N-2)*(N-3)/2 + 1); // non-increasing

    // INIT
    for (int i = 2; i <= N - 1; i++)
    {
        eHat[i] = -1;
    }
    int count = 1;
    double max_weight = 0;

    for (int i = 1; i <= (N-2)*(N-3)/2; i++)
    {
        int a = findRoot(edge[i].a);
        int b = findRoot(edge[i].b);
        if (a != b) // if not cycle, add it.
        {
            eHat[a] = b;
            tree[count].a = edge[i].a;
            tree[count].b = edge[i].b;
            max_weight += edge[i].MI;
            count++;
        }
        else
        if(count == N - 2)
            break;
    }
    return 0;
}
```

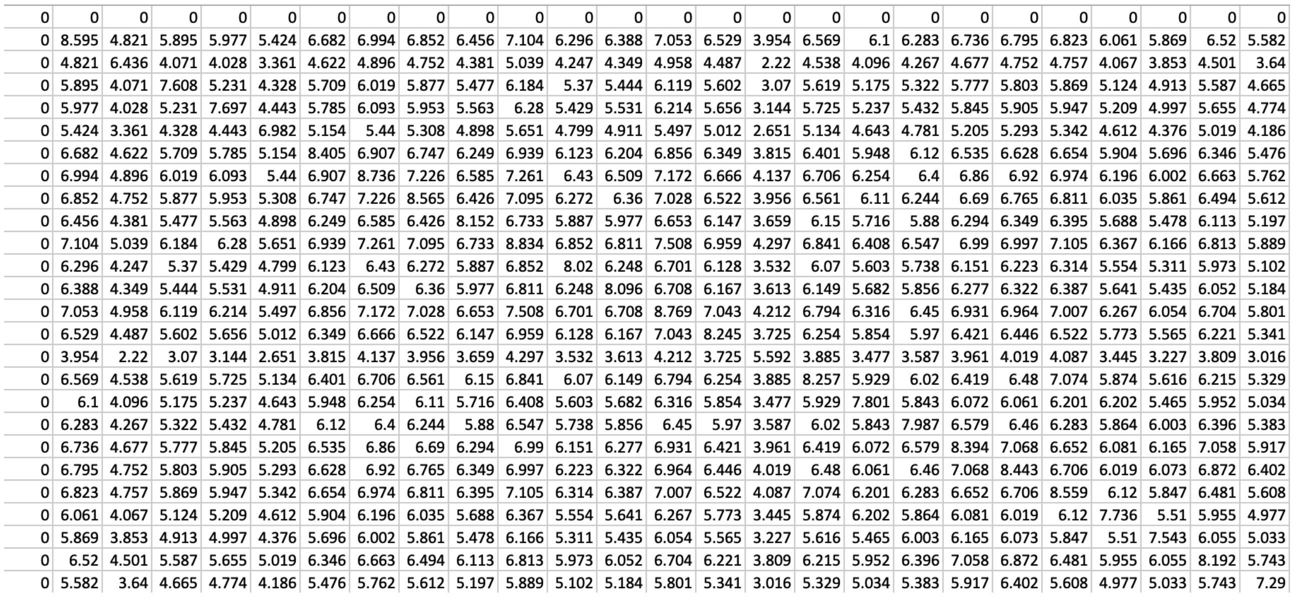
Main function for Kruskal Algorithm

Problem 3.3

3.3.1 Results of MI

I report my results of MI here, and due to my broken MATLAB, I have to export them as ”mat”, then use python transform them into the format of ”txt”.

In case wrong MI leads me to a wrong cost, I report them here to show the Program below is certainly correct.



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	8.595	4.821	5.895	5.977	5.424	6.682	6.994	6.852	6.456	7.104	6.296	6.388	7.053	6.529	3.954	6.569	6.1	6.283	6.736	6.795	6.823	6.061	5.869	6.52	5.582
0	4.821	6.436	4.071	4.028	3.361	4.622	4.896	4.752	4.381	5.039	4.247	4.349	4.958	4.487	2.22	4.538	4.096	4.267	4.677	4.752	4.757	4.067	3.853	4.501	3.64
0	5.895	4.071	7.608	5.231	4.328	5.709	6.019	5.877	5.477	6.184	5.37	5.444	6.119	5.602	3.07	5.619	5.175	5.322	5.777	5.803	5.869	5.124	4.913	5.587	4.665
0	5.977	4.028	5.231	7.697	4.443	5.785	6.093	5.953	5.563	6.28	5.429	5.531	6.214	5.656	3.144	5.725	5.237	5.432	5.845	5.905	5.947	5.209	4.997	5.655	4.774
0	5.424	3.361	4.328	4.443	6.982	5.154	5.44	5.308	4.898	5.651	4.799	4.911	5.497	5.012	2.651	5.134	4.643	4.781	5.205	5.293	5.342	4.612	4.376	5.019	4.186
0	6.682	4.622	5.709	5.785	5.154	8.405	6.907	6.747	6.249	6.939	6.123	6.204	6.856	6.349	3.815	6.401	5.948	6.12	6.535	6.628	6.654	5.904	5.696	6.346	5.476
0	6.994	4.896	6.019	6.093	5.44	6.907	8.736	7.226	6.585	7.261	6.43	6.509	7.172	6.666	4.137	6.706	6.254	6.4	6.86	6.92	6.974	6.196	6.002	6.663	5.762
0	6.852	4.752	5.877	5.953	5.308	6.747	7.226	8.565	6.426	7.095	6.272	6.36	7.028	6.522	3.956	6.561	6.11	6.244	6.69	6.765	6.811	6.035	5.861	6.494	5.612
0	6.456	4.381	5.477	5.563	4.898	6.249	6.585	6.426	8.152	6.733	5.887	5.977	6.653	6.147	3.659	6.15	5.716	5.88	6.294	6.349	6.395	5.688	5.478	6.113	5.197
0	7.104	5.039	6.184	6.28	5.651	6.939	7.261	7.095	6.733	8.834	6.852	6.811	7.508	6.959	4.297	6.841	6.408	6.547	6.99	6.997	7.105	6.367	6.166	6.813	5.889
0	6.296	4.247	5.37	5.429	4.799	6.123	6.43	6.272	5.887	6.852	8.02	6.248	6.701	6.128	3.532	6.07	5.603	5.738	6.151	6.223	6.314	5.554	5.311	5.973	5.102
0	6.388	4.349	5.444	5.531	4.911	6.204	6.509	6.36	5.977	6.811	6.248	8.096	6.708	6.167	3.613	6.149	5.682	5.856	6.277	6.322	6.387	5.641	5.435	6.052	5.184
0	7.053	4.958	6.119	6.214	5.497	6.856	7.172	7.028	6.653	7.508	6.701	6.708	8.769	7.043	4.212	6.794	6.316	6.45	6.931	6.964	7.007	6.267	6.054	6.704	5.801
0	6.529	4.487	5.602	5.656	5.012	6.349	6.666	6.522	6.147	6.959	6.128	6.167	7.043	8.245	3.725	6.254	5.854	5.97	6.421	6.446	6.522	5.773	5.565	6.221	5.341
0	3.954	2.22	3.07	3.144	2.651	3.815	4.137	3.956	3.659	4.297	3.532	3.613	4.212	3.725	5.592	3.885	3.477	3.587	3.961	4.019	4.087	3.445	3.227	3.809	3.016
0	6.569	4.538	5.619	5.725	5.134	6.401	6.706	6.561	6.15	6.841	6.07	6.149	6.794	6.254	3.885	8.257	5.929	6.02	6.419	6.48	7.074	5.874	5.616	6.215	5.329
0	6.1	4.096	5.175	5.237	4.643	5.948	6.254	6.11	5.716	6.408	5.603	5.682	6.316	5.854	3.477	5.929	7.801	5.843	6.072	6.061	6.201	6.202	5.465	5.952	5.034
0	6.283	4.267	5.322	5.432	4.781	6.12	6.4	6.244	5.88	6.547	5.738	5.856	6.45	5.97	3.587	6.02	5.843	7.987	6.579	6.46	6.283	5.864	6.003	6.396	5.383
0	6.736	4.677	5.777	5.845	5.205	6.535	6.86	6.69	6.294	6.99	6.151	6.277	6.931	6.421	3.961	6.419	6.072	6.579	8.394	7.068	6.652	6.081	6.165	7.058	5.917
0	6.795	4.752	5.803	5.905	5.293	6.628	6.92	6.765	6.349	6.997	6.223	6.322	6.964	6.446	4.019	6.48	6.061	6.46	7.068	8.443	6.706	6.019	6.073	6.872	6.402
0	6.823	4.757	5.869	5.947	5.342	6.654	6.974	6.811	6.395	7.105	6.314	6.387	7.007	6.522	4.087	7.074	6.201	6.283	6.652	6.706	8.559	6.12	5.847	6.481	5.608
0	6.061	4.067	5.124	5.209	4.612	5.904	6.196	6.035	5.688	6.367	5.554	5.641	6.267	5.773	3.445	5.874	6.202	5.864	6.081	6.019	6.12	7.736	5.51	5.955	4.977
0	5.869	3.853	4.913	4.997	4.376	5.696	6.002	5.861	5.478	6.166	5.311	5.435	6.054	5.565	3.227	5.616	5.465	6.003	6.165	6.073	5.847	5.51	7.543	6.055	5.033
0	6.52	4.501	5.587	5.655	5.019	6.346	6.663	6.494	6.113	6.813	5.973	6.052	6.704	6.221	3.809	6.215	5.952	6.396	7.058	6.872	6.481	5.955	6.055	8.192	5.743
0	5.582	3.64	4.665	4.774	4.186	5.476	5.762	5.612	5.197	5.889	5.102	5.184	5.801	5.341	3.016	5.329	5.034	5.383	5.917	6.402	5.608	4.977	5.033	5.743	7.29

3.3.2 Program

Combining the three parts in Section 3.2, use the MI data export from MATLAB.

Notice that f.open is using absolute routine, you should change it into the rountine in your computer.

```
#include <iostream>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <vector>
#define N 27
using namespace std;

struct Edge {
    int a;
    int b;
    double MI;
    bool operator < (const Edge &A) const
    {
        return MI > A.MI;
    }
}edge[N*(N-1)/2 + 2];

struct Tree {
    int a;
    int b;
    bool flag = true;
    bool operator < (const Edge &A) const
    {
        return a < A.a;
    }
}tree[N-2];

void init()
{
    ifstream f;
    f.open("/Users/Oasis/Desktop/exam/working_data/MI_c.txt");
    string str;
    vector<vector<double>> num;
    if(f.fail())
    {
        cout << "failed to read the file" << endl;
        return;
    }

    while(getline(f, str, ','))
    {
        istringstream input(str);
        vector<double> tmp;
        double a;
        while(input >> a)
            tmp.push_back(a);
        num.push_back(tmp);
    }

    int count = 1;
    for(int i = 2; i < N; i++)
    {
        for(int j = i + 1; j < N; j++)
        {
            edge[count].a = i;
            edge[count].b = j;
            edge[count].MI = num[i*N + j][0];
            // printf("%d, %d, %d, %lf\n", count, i, j, edge[count].MI);
            count++;
        }
        cout << endl;
    }

    f.close();
    return;
}
```

```
int eHat[N]; // 0, 1, meaningful points
int eHat1[N];

int findRoot(int x) // non-routine compress
{
    if (eHat[x] == -1)
        return x;
    else
        return findRoot(eHat[x]);
}

void display()
{
    for (int i = 2; i <= N - 1; i++)
    {
        int tmp1 = i;
        while(eHat1[i] != -1)
        {
            printf("%d-", i);
            int tmp2 = i;
            i = eHat1[i];
            if(eHat1[i] == -1)
            {
                printf("%d\n", i);
            }
        }
        i = tmp1;
    }
    return;
}


int main() {
    init();

    sort(edge + 1, edge + (N-2)*(N-3)/2 + 1); // non-increasing

    // INIT
    for (int i = 2; i <= N - 1; i++)
    {
        eHat[i] = -1;
    }
    int count = 1;
    double max_weight = 0;

    for (int i = 1; i <= (N-2)*(N-3)/2; i++)
    {
        int a = findRoot(edge[i].a);
        int b = findRoot(edge[i].b);
        if (a != b) // if not cycle, add it.
        {
            eHat[a] = b;
            tree[count].a = edge[i].a;
            tree[count].b = edge[i].b;
```

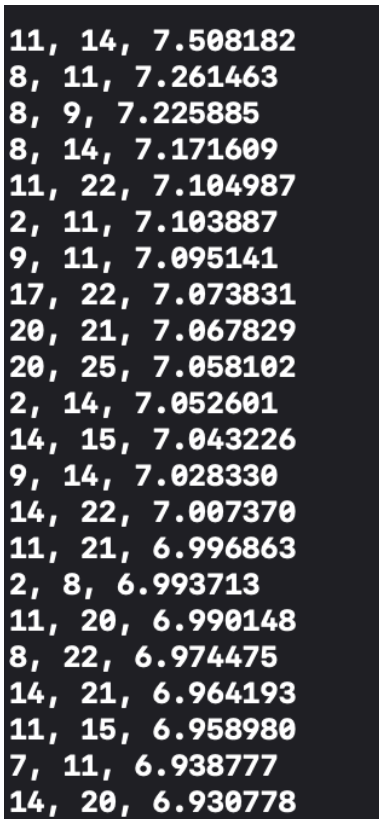
```
                max_weight += edge[i].MI;
                count++;
            }
            else
            if(count == N - 2)
                break;
    }

    printf("Max Weight is %f\n", max_weight);
//    display();
    for (int i = 1; i < N-2; i++)
    {
        printf("%d-%d\n", tree[i].a, tree[i].b);
    }
    return 0;
}
```

Kruskal.cpp

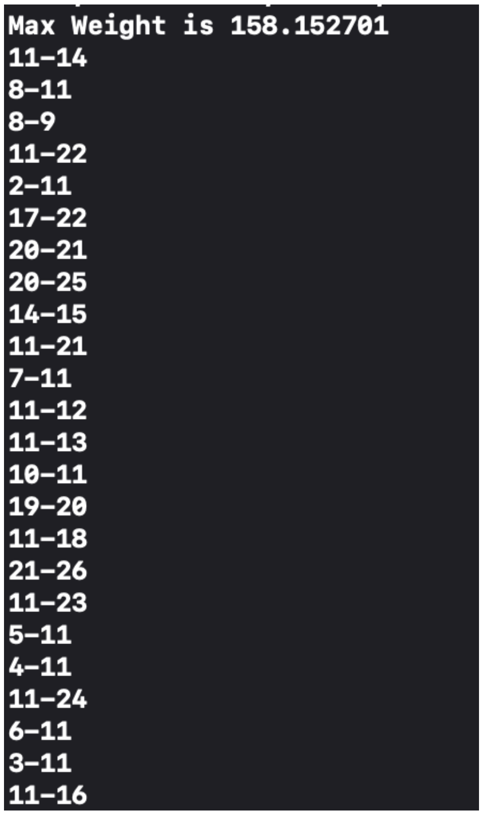
3.3.3 Results

This part is reporting the result of reconstruction.
Primarily, we report our max weight is 158.152701.



```
11, 14, 7.508182
8, 11, 7.261463
8, 9, 7.225885
8, 14, 7.171609
11, 22, 7.104987
2, 11, 7.103887
9, 11, 7.095141
17, 22, 7.073831
20, 21, 7.067829
20, 25, 7.058102
2, 14, 7.052601
14, 15, 7.043226
9, 14, 7.028330
14, 22, 7.007370
11, 21, 6.996863
2, 8, 6.993713
11, 20, 6.990148
8, 22, 6.974475
14, 21, 6.964193
11, 15, 6.958980
7, 11, 6.938777
14, 20, 6.930778
```

(a) Part of sorting results for MI.



```
Max Weight is 158.152701
11-14
8-11
8-9
11-22
2-11
17-22
20-21
20-25
14-15
11-21
7-11
11-12
11-13
10-11
19-20
11-18
21-26
11-23
5-11
4-11
11-24
6-11
3-11
11-16
```

(b) The edges of reconstruction.

Problem 3.4

I suppose there might be 2 places to be improved, one is related to details, and the other one is related to the algorithm.

1. **Details.**

   When we use **Union-find Sets** to solve this problem, such as the function I used.

```
int findRoot(int x) // non-routine compress
{
    if (eHat[x] == -1)
        return x;
    else
        return findRoot(eHat[x]);
}
```

   There will be a problem that the time required for this process is related to the distance between the node and root, that is, to the height of the tree. In the process of merging two trees, if we simply merge two trees like this, the tree height may increase. Furthermore, the time-consuming of finding root nodes will increase, and in extreme cases the tree may degenerate into a single linked list. Thus, finding the root node on it will become very time-consuming.

   To solve this problem, we can use routine compression and optimize the tree structure by having all points in the same set point to the same root node, which can be shown like this.

```
int findRoot(int x) // routine compress
{
    if (eHat[x] == -1)
        return x;
    else
    {
        int tmp = findRoot(eHat[x]);
        eHat[x] = tmp;
        return tmp;
    }
}
```

2. **Algorithm.**

   In the maximum weight spanning tree algorithm we use Kruskal Algorithm to achieve our goals. However, Kruskal Algorithm is with the complexity of $O(ElogE)$, and [1] mentioned a much faster argorithm which can be nearly viewed as linear time. It utilized a data structure **soft heap**. Inside the algorithm we only need to calculate for its function **meld** and **sift**. As is proved, it costs $O(m\alpha(m, n))$ where $\alpha(.)$is the classical functional inverse of Ackermann's function, which is usually no more than a constant. Thus, [1] Algorithm is nearly viewed as a linear time algorithm.

Due to the limited time, I didn't apply it into this paper; however, its codes are given in the paper, which is clearly to see and can be implemented.

Reference

[1] Chazelle, Bernard. (2000). The soft heap: An approximate priority queue with optimal error rate. J. ACM. 47. 1012-1027. 10.1145/355541.355554.