EE 232E Graphs and Network Flows

HW4

Weikun Han 804774358
Duzhi Chen 004773782
2017/6/5

Introduction

In this assignment, we will study data from stock market. The goal of this assignment is to study correlation structures among fluctuation patterns of stock prices using tools from graph theory. A second goal is to utilize the underlying data to study approximation algorithms for solving the Δ-Traveling Salesman Problem (TSP) using Minimum Spanning Trees and Eulerian Cycles.

In first part, first, we understood why log return is a good measurement. And we understood use Pearson correlation coefficient to calculating correlations among time series data. Next, we used Pearson's distance to get adjacency matrix, and we used adjacency matrix to construct correlation graphs. Then, we used R to create undirected weight graph, and we used R to find Minimum Spanning Trees (MSTs) for the correlation graphs. Next, we predicted the market sector of an unknown stock based on the MSTs, and we evaluated sector clustering in MSTs.

In the second part, first, we used the MSTs to approximate tour for the TSP. Next, we constructed correlation graphs for weekly data, and we also evaluated sector clustering in new MSTs. In the end, we modified correlations to understand Pearson correlation coefficient based on the new MSTs.

Questions

1. Calculating Correlations among Time Series Data:
Can you explain why log return is a good measurement (do some Googling)?

From the online source, I can summary why log return is a good measurement. First, in general, while stock returns are not exactly normal, that is much closer to being the case than, say, assuming the prices themselves are normal. When testing for auto regression, I don't have to have the underlying process be normal to get a meaningful result, but it makes the residual more manageable to analyze further.

Second, the reason I want to look at returns when I test for auto regression is because I am looking for deviations from the random walk theory -- which enables me to be able to trade profitably. Therefore, an autoregressive coefficient with significance immediately suggests that there is some real effect there, potentially one I can trade on.

2. Constructing Correlation Graphs:
Plot the histogram of dij's.

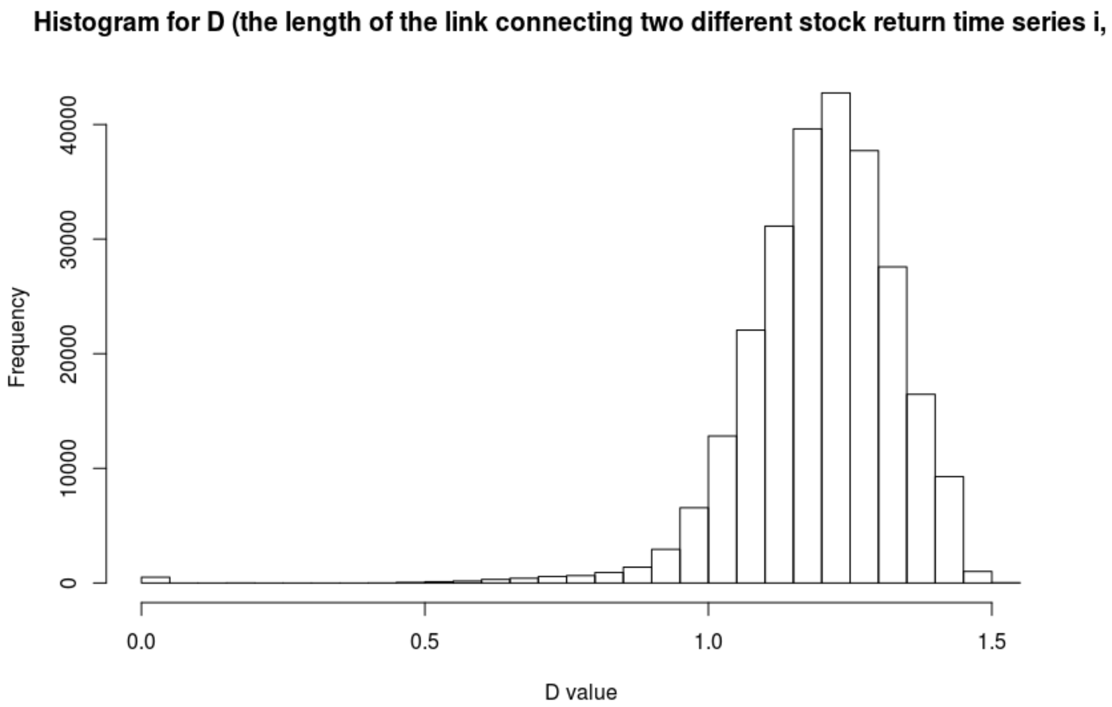

Figure 1, the histogram of the D

Here, I want explain what D is in histogram. But, first we need understand what P is here because D is come from P. In question 1, calculate correlations among time series data is use concept from Pearson correlation coefficient. Here, r is called the log return of the closing price, which are two variables in Pearson correlation coefficient. Therefore from the Pearson correlation concept, the range is from 1 to -1. The value at 1 means two variables have strong positive correlation, and the value -1 means two variables have strong negative correlation. From Figure 3, we can see our result is right based on the concept.

```
-------------------------Processing Finshed 1 -------------------------------
 Successful calculated all log return values in each file.
 The total number of files we processed:  505
 The total number of log return values in single file:  764
 ----------------------------------------------------------------------------
```

```
file_list                  Large list (505 elements, 2.9 Mb)
   : num [1:764] 0.001284 0.014381 -0.0058 0.000363 0.004894 ...
   : num [1:764] 0.00274 0.00301 0.00191 0.01489 0.0273 ...
   : num [1:764] 0.00585 -0.00544 -0.00447 -0.00867 0.02489 ...
   : num [1:764] 0.00186 0.01404 -0.01096 -0.00351 -0.00735 ...
   : num [1:764] -0.00837 0.00351 -0.0088 0.03626 -0.0122 ...
   : num [1:764] -0.00929 0.01283 -0.00508 -0.00356 -0.00217 ...
   : num [1:764] -0.00285 0.00724 -0.00672 0.00285 0.00103 ...
   : num [1:764] -0.00603 -0.00455 -0.00342 -0.00663 0.0079 ...
   : num [1:764] -0.01659 -0.00195 -0.03309 -0.01372 0.00765 ...
   : num [1:764] 0.01154 -0.00448 -0.00863 0.00118 0.00999 ...
   : num [1:764] 0.00714 -0.00322 -0.00624 0.02586 -0.00453 ...
   : num [1:764] -0.00103 0.00425 -0.01046 0.00518 0.00502 ...
```

Figure 2, each file have 764 log return value

```
------------------------Processing Finshed 2 ------------------------------
 Successful calculated the cross correlation coefficient of two different stock-return time series.
 The total number of p value we processed:  255025
 ---------------------------------------------------------------------------
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	1.000000000	0.29248048	0.19391830	0.0571720555	0.362243450	0.220241092	0.477966101	0.44318735	0.396831742	0.3865600805
2	0.292480478	1.00000000	0.25874995	0.0678734920	0.253767258	0.223827142	0.358405389	0.31913219	0.335831271	0.3318162487
3	0.193918297	0.25874995	1.00000000	0.0566843362	0.237723861	0.162557620	0.318131641	0.25991627	0.281453555	0.2512762213
4	0.057172056	0.06787349	0.05668434	1.0000000000	0.099870015	0.049542181	0.072912942	0.05864815	0.080518847	-0.0133238068
5	0.362243450	0.25376726	0.23772386	0.0998700152	1.000000000	0.376728402	0.462183412	0.26406465	0.313604221	0.2965578985
6	0.220241092	0.22382714	0.16255762	0.0495421812	0.376728402	1.000000000	0.387727620	0.24834295	0.213696119	0.1910042462
7	0.477966101	0.35840539	0.31813164	0.0729129421	0.462183412	0.387727620	1.000000000	0.49313090	0.445114299	0.4023270778
8	0.443187353	0.31913219	0.25991627	0.0586481484	0.264064650	0.248342951	0.493130900	1.00000000	0.495909929	0.4479051000
9	0.396831742	0.33583127	0.28145355	0.0805188475	0.313604221	0.213696119	0.445114299	0.49590993	1.000000000	0.4566362216
10	0.386560081	0.33181625	0.25127622	-0.0133238068	0.296557898	0.191004246	0.402327078	0.44790510	0.456636222	1.0000000000
11	0.307862376	0.24613845	0.27297252	0.0576160668	0.271724442	0.160173474	0.338196094	0.40090694	0.308250536	0.3359603838
12	0.374191726	0.39245556	0.30068141	0.0603901922	0.306336618	0.312425954	0.456404934	0.57255695	0.482727981	0.4495924512
13	0.352002228	0.37630017	0.27692221	0.0517015929	0.240643272	0.267046329	0.488654874	0.39562849	0.371422579	0.3446200443
14	0.408074374	0.42386066	0.20898415	0.0871083407	0.305371291	0.224167751	0.443460761	0.45305832	0.509767450	0.4102296148
15	0.123232676	0.10941170	0.16655696	0.0611790064	0.195378681	0.100642312	0.263795123	0.26168402	0.248862562	0.1473683325
16	0.139569748	0.11964231	0.17389619	0.0697679533	0.172221535	0.096084874	0.256961259	0.27680933	0.267278628	0.1654309001
17	0.273831359	0.20311249	0.23272852	0.0593845887	0.219454147	0.128895545	0.338103282	0.33714374	0.326979566	0.2975970188
18	0.352324401	0.28820470	0.24954277	0.0673337045	0.433553794	0.366765914	0.412777742	0.36980723	0.372370499	0.3122515145
19	0.431381043	0.33721258	0.32616995	0.0416676041	0.331058743	0.287131471	0.512507748	0.53837202	0.420994068	0.4203791072
20	0.294112217	0.27861678	0.22866978	0.0622264393	0.454718193	0.400653479	0.399453296	0.25216655	0.301991905	0.2880094625
21	0.412863790	0.34609857	0.31664603	0.0921058074	0.328040242	0.339818473	0.480008317	0.50792955	0.439697085	0.3857521933
22	0.207936874	0.19198593	0.18730006	0.0821827483	0.235945618	0.141559267	0.303295689	0.35675917	0.313715295	0.2356908295
23	0.282633876	0.23592591	0.27810631	0.0676964818	0.193626793	0.195201817	0.346237449	0.42106361	0.297726557	0.3176406807
24	0.420158927	0.37057843	0.31324131	0.0853486129	0.339420716	0.310224725	0.522864866	0.51065093	0.454540006	0.4353900489

Figure 3, the P value we calculate and store it into matrix

Understand P and finish calculate P value, we can next to figure out what is D and calculate D value. In assignment description, the D is the length of the link connecting two different stock return time series i j. However, D is come from Pearson's distance: a distance metric for two variables X and Y defined from their correlation coefficient. Therefore, the range of D is from 0 to 2. From Figure 4, we can see our result is right based on the concept.

```
------------------------Processing Finshed 3 ------------------------------
 Successful calculated the length of the link connecting two different stock return time series i, j.
 The total number of d value we processed:  255025
 ---------------------------------------------------------------------------
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
1	0.0000000	1.1895541	1.2697100	1.373192	1.1293862	1.2488066	1.0217964	1.0552845	1.0983335	1.1076461	1.1765523	1.1187567	1.1384180
2	1.1895541	0.0000000	1.2175796	1.365377	1.2216650	1.2459317	1.1327794	1.1669343	1.1525352	1.1560136	1.2278938	1.1023107	1.1168705
3	1.2697100	1.2175796	0.0000000	1.373547	1.2347276	1.2941734	1.1677914	1.2166213	1.1987881	1.2237024	1.2058420	1.1826399	1.2025621
4	1.3731919	1.3653765	1.3735470	0.000000	1.3417377	1.3787370	1.3616806	1.3721165	1.3560834	1.4236037	1.3728685	1.3708463	1.3771699
5	1.1293862	1.2216650	1.2347276	1.341738	0.0000000	1.1164870	1.0371274	1.2132068	1.1716619	1.1861215	1.2068766	1.1778484	1.2323609
6	1.2488066	1.2459317	1.2941734	1.378737	1.1164870	0.0000000	1.1065915	1.2260971	1.2540366	1.2720030	1.2960143	1.1726671	1.2107466
7	1.0217964	1.1327794	1.1677914	1.361681	1.0371274	1.1065915	0.0000000	1.0068457	1.0534569	1.0933187	1.1504816	1.0426841	1.0112815
8	1.0552845	1.1669343	1.2166213	1.372117	1.2132068	1.2260971	1.0068457	0.0000000	1.0040817	1.0508044	1.0946169	0.9246005	1.0994285
9	1.0983335	1.1525352	1.1987881	1.356083	1.1716619	1.2540366	1.0534569	1.0040817	0.0000000	1.0424623	1.1762223	1.0171254	1.1212292
10	1.1076461	1.1560136	1.2237024	1.423604	1.1861215	1.2720030	1.0933187	1.0508044	1.0424623	0.0000000	1.1524232	1.0491974	1.1448842
11	1.1765523	1.2278938	1.2058420	1.372868	1.2068766	1.2960143	1.1504816	1.0946169	1.1762223	1.1524232	0.0000000	1.1175989	1.2311164
12	1.1187567	1.1023107	1.1826399	1.370846	1.1778484	1.1726671	1.0426841	0.9246005	1.0171254	1.0491974	1.1175989	0.0000000	1.0994022
13	1.1384180	1.1168705	1.2025621	1.377170	1.2323609	1.2107466	1.0112815	1.0994285	1.1212292	1.1448842	1.2311164	1.0994022	0.0000000
14	1.0880493	1.0734424	1.2577884	1.351215	1.1786676	1.2456583	1.0550253	1.0458888	0.9901844	1.0860667	1.1579683	1.1095124	1.0980964
15	1.3242110	1.3346073	1.2910794	1.370271	1.2685593	1.3411619	1.2134289	1.2151675	1.2256732	1.3058573	1.2429716	1.2041442	1.3410500
16	1.3118157	1.3269195	1.2853823	1.363988	1.2866845	1.3445558	1.2190478	1.2026560	1.2105547	1.2919513	1.2443774	1.1869874	1.3264278
17	1.2051296	1.2624480	1.2387667	1.371580	1.2494366	1.3199276	1.1505622	1.1513959	1.1601900	1.1852451	1.0939353	1.1728312	1.2383221
18	1.1381350	1.1931432	1.2251181	1.365772	1.0643742	1.1253747	1.0837179	1.1226689	1.1203834	1.1728158	1.1813915	1.1170616	1.1763179
19	1.0664136	1.1513361	1.1608876	1.384437	1.1566687	1.1940423	0.9874130	0.9608621	1.0761096	1.0766809	1.1029203	0.9255067	1.1342130
20	1.1881816	1.2011521	1.2420388	1.369506	1.0443005	1.0948484	1.0959441	1.2229746	1.1815313	1.1933068	1.2482137	1.2000434	1.1941527
21	1.0836385	1.1435921	1.1690628	1.347512	1.1592754	1.1490705	1.0197957	0.9920388	1.0585867	1.1083752	1.0798734	1.0225872	1.1091920
22	1.2586208	1.2712310	1.2749117	1.354856	1.2361670	1.3102982	1.1804273	1.1342318	1.1715671	1.2363731	1.2117224	1.1278500	1.2650981
23	1.1978031	1.2361829	1.2015770	1.365506	1.2699395	1.2686987	1.1434706	1.0760450	1.1851358	1.1682117	1.2013561	1.1246941	1.2140993
24	1.0768854	1.1219818	1.1719716	1.352517	1.1494166	1.1745427	0.9768676	0.9892917	1.0444712	1.0626476	1.1068067	0.9575121	1.0602158

Figure 4, the D value we calculate and store it into matrix

Once we store D value into matrix, it can be adjacency matrix to get the correlation graph. In graph theory and computer science, an adjacency matrix is a square matrix used to represent a finite graph. Therefore, D value here can be consider as the weight between two nodes. In the end, we can use igraph package in R to construct graph and store it for later question.

Now construct a weighted graph G with adjacency matrix D = [d_{ij}].

```
-------------------------Processing Finshed 4 ------------------------------
 Successful store graph into .txt file.
 The file store at /home/weikun/Downloads/finance_data/graph.txt
 ----------------------------------------------------------------------------
```

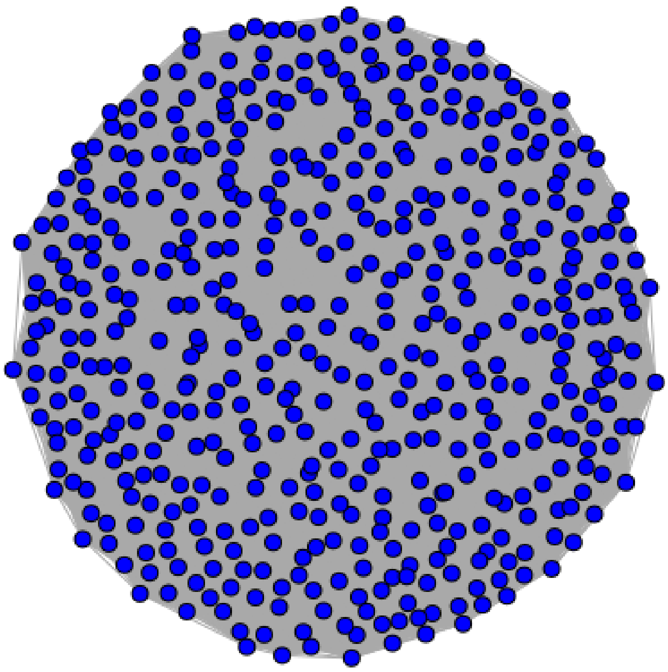

Figure 5, the plot for correlation graph

3. Finding Minimum Spanning Trees (MSTs) for the Correlation Graphs: Compute a minimum spanning tree (MST) for the correlation graph.

To get a MST for the correlation graph, we used the function in igraph package [1]. In igraph package, it has different algorithm to get a MST.

mst	*Minimum spanning tree*

Description

A subgraph of a connected graph is a *minimum spanning tree* if it is tree, and the sum of its edge weights are the minimal among all tree subgraphs of the graph. A minimum spanning forest of a graph is the graph consisting of the minimum spanning trees of its components.

Usage

```
mst(graph, weights = NULL, algorithm = NULL, ...)
```

Arguments

`graph`	The graph object to analyze.
`weights`	Numeric algorithm giving the weights of the edges in the graph. The order is determined by the edge ids. This is ignored if the `unweighted` algorithm is chosen
`algorithm`	The algorithm to use for calculation. `unweighted` can be used for unwieghted graphs, and `prim` runs Prim's algorithm for weighted graphs. If this is `NULL` then igraph tries to select the algorithm automatically: if the graph has an edge attribute called `weight` of the `weights` argument is not `NULL` then Prim's algorithm is chosen, otherwise the unwweighted algorithm is performed.
`...`	Additional arguments, unused.

Figure 6, the MST function description in the igraph package

Can you observe any particular pattern in the MST?

For the first question in this problem, if I do not color-code the nodes based on sectors, I can only see the MST have properties: the sum of the weights of the edges of the spanning trees is less than or equal to the sum of the weights of the edges of the other spanning trees. From this observation, particular pattern is same as the concept for MST. In the concept of MST: a minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

Each stock can be categorized into a sector, which can be found from Name_sector.csv file. Plot the MST and color-code the nodes based on sectors.

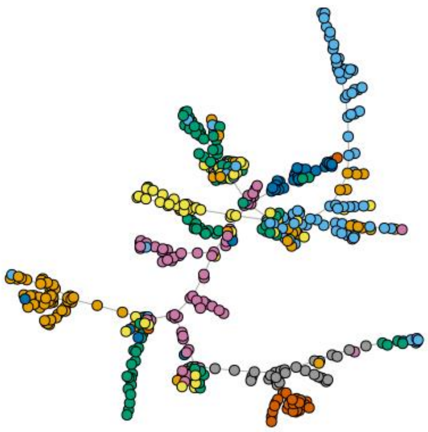

Figure 7, the plot of a MST and color-code the nodes based on sectors

Do you see any pattern in this colorful graph? Interpret your observations.

In Figure 7 above, we observe that sector clustering in the MST. Here, we color-code the nodes based on sectors. Then, we saw each branch in MST almost have same color nodes. On the other hand, each branch clustering each sectors.

The reason why each branch in MST clustering each sectors is that the MST for the correlation graph. In the correlation graph, the weight is calculated from Pearson's distance. Therefore, if two stocks have positive correlation base on the log turn for the price, Pearson's distance will be very small and so the weight between two nodes (two stocks) will be very small. From the MST concept, we know the MST is the minimum possible total edge weight. Thus, the MST will cluster each two nodes (two stocks) with positive correlation.

4. Evaluating Sector Clustering in MSTs:
Evaluate the performance of such a method as follows:

$$\alpha = \frac{1}{|V|} \sum_{v_i \in V} P(v_i \in S_i)|$$

where Si is the sector for the node i, and denoting neighbors of vi by Ni

$$P(v_i \in S_i) = \frac{|\{j|v_j \in N_i, S_j = S_i\}|}{|\{j|v_j \in N_i\}|}.$$

```
-------------------------Processing Finshed 1 -------------------------------
 Successful evaluating sector clustering in Minimum Spanning Trees (MSTs)
 The proformace is calculated as:  0.8140095
 ---------------------------------------------------------------------------
```

Figure 8, the result for evaluating sector clustering in MSTs

From Figure 8, we can see the performance is 81.4%. Here, I want explain how we evaluate sector clustering in MSTs. In question description, assume we want to predict the market sector of an unknown stock based on the MST you just found. One might be interested in doing so just based on the immediate neighbors of a stock in the MST.

`neighbors` *Neighboring (adjacent) vertices in a graph*

Description

A vertex is a neighbor of another one (in other words, the two vertices are adjacent), if they are incident to the same edge.

Usage

```
neighbors(graph, v, mode = c("out", "in", "all", "total"))
```

Arguments

graph	The input graph.
v	The vertex of which the adjacent vertices are queried.
mode	Whether to query outgoing ('out'), incoming ('in') edges, or both types ('all'). This is ignored for undirected graphs.

Figure 9, the neighbors function description in the igraph package

This process is spited into two parts. The first part is almost same as the question 3, which gives each sector a ID first. And use igraph packge to rename the nodes in graph base on the sector ID. Then find the MST, and each node in MST is named as sector ID (base the concept of MST, the total nodes in MST should same as the total nodes in original graph). Next, we use igraph packge to find each node's neighbors nodes. If the neighbor have same sector ID, we count it. Therefore, we can get one node probability. Base on the performance formula, we sum all nodes probability and we divide by totals nodes (here is 505 nodes).

5. ∆-Traveling Salesman Problem:

Determine if the triangle inequality holds for the fully connected graph G. Show your methodology.

I will use the triangle inequality provide in lecture to determine if it holds for the fully connected graph G:

$$d_{ij} + d_{jk} \geq d_{ik} \qquad \text{for all } 1 \leq i, j, k \leq n$$

where d is the length of the link connecting two different stock return time series i. j, and it can convert to the weight between two nodes (two stocks).

Moreover, I would proof triangle inequality using a proof by contradiction. Suppose d(x, y) + d(y, z) < d(x, z) d(x, y) + d(y, z) < d(x, z). By definition, d(x, y) d(x, y) is the length of the shortest path from x to y, and d(y, z) d(y, z) is the length of the shortest path from y to z. However, d(x, z) d(x, z) is the length of the shortest path from x to z, so d(x, y) + d(y, z) d(x, y) + d(y, z) can't be less than d(x, z) d(x, z). So we have a contradiction.

Now, we want to find an approximation algorithm for the traveling salesman problem (TSP) on G

1. Find the minimum spanning tree T under $[d_{ij}]$.
2. Create a multigraph G by using *two copies* of each edge of T.
3. Find an Eulerian walk of G and an embedded tour.

Then find an approximate tour for the TSP

```
-------------------------Processing Finshed 4 -------------------------------
 Successful find an approximation tour for the traveling salesman problem (TSP) on G.
 The number of nodes in an apporximate the traveling salesman problem (TSP):  505
 -------------------------------------------------------------------------------
```

```
> tsp_tour
  [1] 204 205  37 431 359 267 101 413 185 352  11 281 375 302 438 220 283 454  48 166 168 364 250 225 450  55 374 324 309 211  73 139
 [33] 388 194 415 199 292 145 296 184 207 439 458  35 251  33 432 274 303 179 294  57 285  63 442 338 118 459   3 449 297 322 478 422
 [65] 237 189 254 124 496 311 368 229 391 106  59 420 390 503  98  44 119 308  89 410  74 470 301 427  62 417  58  38 266 445 305 201
 [97] 259 325 269 366 440 476 321 361  75  56   6 357 337  16 105 419 146 103 493 479 154 414 385 252 407  68 287 120 482 275 133 159
[129] 384 150 181 460 173  92 486 112  96 264  97 386 144 501 307  40 350  54 126 423  29  34 389  24 434 424 411 110  61  87 132 276
[161] 451 418 466 378 395 472  71 462 369 353 257 480 155 290 161 253 272 111 221 360 315 138 299 416 500 271 396 196 235  17  39  94
[193]  46 288 158 461 152 425  49   9 495  93 135 192 193 455 217  20 238 239  82 100 327 255 428  76 499 228 153 380  23 162 339 349
[225]   7 243   8 115 448  83 412 136 137 246 491 265 347 149 316 323 233 293 490 170 365 351 160 492 125 381 485 429 114 282  52 130
[257] 446 468 377  41  13 471  28  53 109 469 306 367 346 328 258 289 270 208 452 397  32 206 437 465 224 240   0 273  60 379 190 142
[289]  79 102 354 227 483 203 176 474 218 219 356 174 441 234 300 113 157 362  64 403 164 236 165 298  78   5 481 231 504 248 178 291
[321]  21 262 484 400 447 209 319 117 182  15 371  51 213 409 226  42 326  45  90  85 387 127   1  99 401 335 280 487 247 202 463 295
[353] 214 406  10 488 332 336 195 393 330 223 343 171 261 382 212 177 197 278 230  22 175 376 140  86 279 200   4 342 256  25 453 286
[385] 370 464 341 198 456 444 241  66 312  43 383  31 163 399 107 329 180 129 249   2 134 498 191 372 310  77 475 430 284  30 426 147
[417] 169  91 304  95 443  80 494 268 355 156 313  88 172 358  14  26  27 363 398 187 333  84  36 489  81 244  50 405 188 320 215  19
[449] 123 477 502  72 334 260 186 408 314 373 167 404 394 317 210 402  18 222 104 457 148 151  47 497 232 108 436 141 131 433 122  12
[481] 331  65 143 128 340  67 421 116 473  70 121 392 435 467  69 348 245 216 277 345 344 183 263 242 318 204
```

```
> duplicated(tsp_tour)
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [22] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [43] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [64] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[106] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[127] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[148] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[190] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[211] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[232] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[253] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[274] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[295] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[316] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[337] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[358] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[379] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[400] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[421] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[442] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[463] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[484] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[505] FALSE  TRUE
> 
```

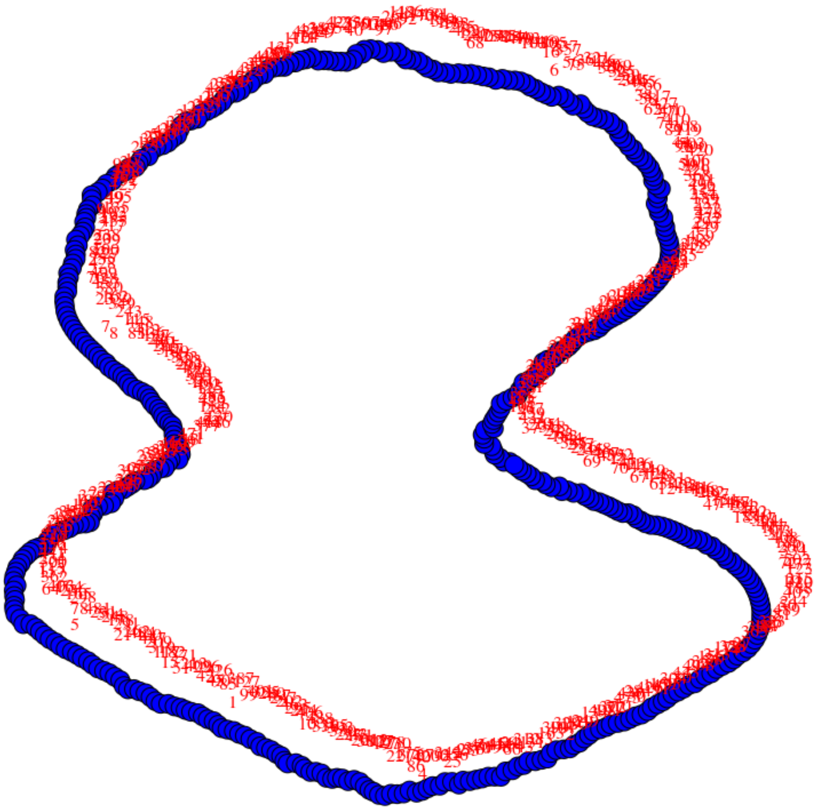

Figure 10, the results and the plot for a tour for TSP using the igraph package

Here, I want to explain how I can get the a tour for TSP. The overview process is followed by the instruction in lecture, which also have detail explain on this reference [2]:

Approximate Traveling Salesperson (TSP) Tour Construction (Doubling MST)

This is a handout for another version of the Approximate TSP Tour Construction Algorithm given on page 119 of the textbook (4th ed.). Given a TSP instance on a graph G with an associated cost matrix C, the algorithm goes through three main steps:

1. Find a minimal spanning tree (MST) on G by using Prim's or Kruskal's algorithm. Let T be the resulting MST.
2. (Doubling) For each edge $(i, j) \in T$, add another edge between i and j with the same cost c_{ij}. Note that the subgraph consisting only of the edges in T and these duplicate edges has an Euler cycle (why?).
3. (Rounding) Pick any vertex i. Find an Euler cycle P that starts and ends at i. Let $P = (x_1, x_2, \ldots, x_n)$, where $x_1 = x_n = i$. Trace the Euler cycle P and delete the repeated vertices until you are left with a TSP tour.

However, in igraph packge, first, I hard to double edge a MST, and, second, no Euler circuit function available. Therefore, I take time to do some research on it. I try to write this kind of function, but the result is not good (the bonus points for this question).

Later, I found one R package have Euler circuit function, which is the PairViz package [3]. However, this package need graph base on the graph package in R [4]. Therefore, for first part in this question, we need use igraph package do MST, and store it into .txt file.

```
-------------------------Processing Finshed 1 -------------------------------
 Successful store graph_mst into .txt file.
 The file store at /home/weikun/Downloads/finance_data/graph_mst.txt
 ---------------------------------------------------------------------------
```

```
graph_mst                       List of 10
    attr: name (v/c), weight (e/n)
    edges (vertex names):
    [1] 0 --440 1 --127 2 --350 3 --459 4 --34 5 --78 6 --224 6 --301 6 --440 6 --465 7 --121 7 --185 7 --349 7 --3…
    15] 8 --115 8 --243 9 --449 9 --495 10 --361 11 --352 12 --130 13 --299 14 --493 15 --147 15 --169 15 --182 15 …
    29] 16 --105 16 --337 17 --39 17 --104 17 --235 17 --457 18 --375 19 --34 19 --324 20 --257 21 --388 21 --454 2…
    43] 23 --307 24 --434 25 --158 26 --127 27 --95 28 --53 28 --185 29 --34 30 --284 31 --297 32 --250 32 --397 33…
    57] 34 --66 34 --70 34 --78 34 --86 34 --200 34 --241 34 --360 34 --389 34 --440 34 --473 35 --251 35 --281 35 …
    71] 36 --367 37 --205 38 --58 38 --73 38 --266 39 --94 40 --307 41 --316 42 --149 42 --326 43 --380 44 --297 45…
    85] 47 --151 47 --185 48 --162 48 --166 48 --454 49 --425 49 --449 50 --244 50 --405 51 --371 52 --130 54 --350…
    99] 57 --294 58 --60 58 --258 58 --402 58 --417 59 --106 59 --215 59 --320 59 --420 61 --427 62 --427 63 --251 …
    13] 65 --213 67 --340 67 --421 68 --209 68 --251 68 --287 68 --319 69 --250 71 --135 71 --236 71 --255 71 --307…
    27] 72 --427 73 --139 73 --176 73 --211 74 --262 74 --410 74 --470 75 --257 75 --317 75 --340 75 --375 76 --250…
    41] 77 --359 78 --123 78 --298 79 --102 79 --129 79 --180 79 --364 80 --443 80 --494 81 --251 81 --489 82 --100…
    55] 83 --435 83 --448 85 --90 85 --127 85 --293 85 --387 87 --132 88 --313 89 --410 91 --149 92 --173 93 --135 …
```

Figure 11, do MST in igraph package and store data it into .txt file

In the second part, we need write a code to covert graph in igraph package to graphNEL in graph package. After that, we can get result in Figure 12.

```
-------------------------Processing Finshed 1 -------------------------------
 Successful convert the graph in 'igraph' package into graphNEL 'graph' package.
 Dose it create a multigraph G by using two copies of each edge of MST:  FALSE
 ---------------------------------------------------------------------------
```

```
> graph
A graphNEL graph with undirected edges
Number of Nodes = 505
Number of Edges = 504
> 
```

Figure 12, convert MST in igraph package to graph package

Next, we can use the function in PairViz package to double edges in MST in

Figure 13.

mk_even_graph	*Constructs an even graph*

Description

~~ Methods for function mk_even_graph. Each of these return an instance of even_graph, where all nodes are of even degree. The result satisfies is_even_graph. The resulting graph yields an euler tour.

Methods

self = "graphNEL",use_weights=TRUE,add_edges=TRUE This is the workhorse method. If self does not satisfy is_even_graph, the graph is forced to be even by one of the folowing. If add_edges is TRUE, the odd nodes are paired off and a new edge added between each pair, possibly duplicating an existing edge. If add_edges is a vector of the odd nodes, they are paired off in this order. If add_edges is FALSE a new dummy node is added with edges going to all odd nodes.

self = "matrix",use_weights=TRUE,add_edges=TRUE first constructs a complete graph using mk_complete_graph, which is then augmented to be even.

self = "numeric",use_weights=FALSE,add_edges=TRUE first constructs a complete graph using mk_complete_graph, which is then augmented to be even.

self = "ANY",use_weights=TRUE,add_edges=TRUE first constructs a complete graph using mk_complete_graph, which is then augmented to be even.

self = "even_graph",add_edges=TRUE returns self.

Figure 13, the double edges function description in the PairViz package

Here, we can see the result use make even graph function in Figure 14. From Figure 14, we see that make graph function is better than the function I write for bonus points. Because make even not necessary directly double edges. If you directly double the edge, you will need to cut it with the same starting point as one leaf in MST.

```
-------------------------Processing Finshed 2 ------------------------------
 Successful create a multigraph G by using two copies of each edge of MST.
 Dose it create a multigraph G by using two copies of each edge of MST:  TRUE
 ---------------------------------------------------------------------------

                      A even_graph graph with undirected edges
                      Number of Nodes = 505
                      Number of Edges = 680
```

Figure 14, make graph to even nodes

Then, we can use the function in PairViz package to do Eulerian Cycle in MST with double edges in Figure 15.

etour	*Constructs eulerian tours on a graph.*

Description

etour– Constructs an eulerian tour on a graph using Hierholzer's algorithm. Returns a vector of node labels. If weighted is TRUE constructs a weight-decreasing eulerian using the modified Hierholzer's algorithm. Usually etour is not called directly, rather the generic function eulerian is used.

Usage

```
etour(g, start=NULL,weighted=TRUE)
```

Arguments

`g`	a graph satisfying `is_even_graph`
`start`	an optional starting node for the tour.
`weighted`	whether tour uses weights

Figure 15, the Eulerian tours function description in the PairViz package

After use the above function, we can get the nicely tour for Eulerian circuit. An Euler circuit is a circuit that uses every edge of a graph exactly once. And an Euler circuit starts and ends at the same vertex. Here, I post the node I visit in Euler circuit in Figure 16. Also, because of the concept of Euler circuit, it will have some repeat visit nodes.

```
-------------------------Processing Finshed 3 ------------------------------
 Successful get the Eulerian circuit for double edges of MST.
 The number of tour nodes in Eulerian circuit:  681
 ----------------------------------------------------------------------------
```

```
> eulerian_cycle
  [1] "204" "205" "37"  "431" "359" "267" "101" "413" "185" "352" "11"  "281" "375" "302" "438" "220" "283" "454" "48"  "166" "168"
 [22] "364" "250" "225" "450" "55"  "374" "324" "309" "211" "73"  "139" "388" "194" "415" "199" "292" "145" "296" "185" "184" "207"
 [43] "439" "458" "281" "35"  "251" "33"  "432" "274" "303" "179" "294" "57"  "285" "251" "63"  "442" "338" "118" "459" "3"   "449"
 [64] "297" "322" "478" "422" "237" "189" "254" "124" "496" "311" "368" "229" "391" "106" "59"  "420" "390" "503" "98"  "297" "44"
 [85] "119" "250" "308" "89"  "410" "74"  "470" "301" "427" "62"  "417" "58"  "38"  "266" "445" "305" "201" "259" "35"  "325" "269"
[106] "366" "440" "476" "321" "361" "375" "75"  "56"  "440" "6"   "301" "357" "337" "16"  "105" "419" "146" "103" "493" "479" "154"
[127] "414" "385" "252" "407" "251" "68"  "287" "120" "482" "303" "275" "133" "364" "159" "384" "150" "181" "460" "173" "92"  "486"
[148] "112" "359" "96"  "264" "97"  "386" "311" "359" "144" "501" "307" "40"  "350" "54"  "146" "126" "305" "359" "423" "29"  "34"
[169] "389" "24"  "434" "250" "424" "411" "110" "427" "61"  "87"  "132" "276" "451" "418" "466" "378" "395" "472" "185" "307" "71"
[190] "462" "369" "420" "353" "75"  "257" "462" "480" "155" "185" "290" "161" "253" "361" "272" "111" "221" "360" "315" "138" "251"
[211] "299" "416" "500" "271" "338" "168" "396" "196" "235" "17"  "39"  "94"  "46"  "288" "158" "168" "461" "152" "425" "49"  "449"
[232] "9"   "495" "93"  "135" "192" "193" "34"  "440" "132" "455" "217" "257" "20"  "238" "155" "239" "82"  "100" "327" "71"  "255"
[253] "428" "76"  "250" "499" "228" "153" "380" "250" "307" "23"  "48"  "162" "478" "339" "349" "7"   "185" "243" "8"   "115" "448"
[274] "83"  "412" "136" "137" "250" "246" "491" "265" "347" "149" "316" "323" "410" "233" "293" "490" "170" "455" "365" "275" "351"
[295] "160" "492" "125" "381" "347" "485" "429" "201" "114" "282" "52"  "130" "106" "446" "468" "377" "124" "111" "316" "41"  "299"
[316] "13"  "471" "185" "28"  "53"  "109" "225" "469" "306" "367" "194" "346" "103" "328" "258" "289" "270" "208" "452" "397" "32"
[337] "250" "206" "437" "465" "6"   "224" "240" "440" "0"   "273" "258" "58"  "60"  "379" "189" "190" "142" "364" "79"  "102" "354"
[358] "385" "227" "483" "203" "176" "73"  "38"  "474" "218" "219" "290" "356" "174" "441" "234" "201" "300" "113" "157" "96"  "362"
[379] "64"  "157" "403" "493" "164" "71"  "236" "165" "298" "78"  "5"   "481" "176" "361" "231" "504" "440" "248" "178" "205" "291"
[400] "454" "21"  "388" "262" "74"  "484" "400" "185" "447" "209" "68"  "319" "117" "182" "15"  "493" "371" "51"  "213" "409" "226"
[421] "160" "149" "42"  "326" "218" "194" "45"  "90"  "85"  "387" "293" "85"  "127" "1"   "206" "99"  "401" "335" "280" "487" "262"
[442] "247" "202" "250" "463" "295" "214" "185" "220" "303" "406" "10"  "361" "488" "332" "336" "195" "226" "393" "330" "223" "361"
[463] "343" "171" "261" "382" "212" "177" "250" "197" "335" "278" "230" "22"  "71"  "135" "175" "376" "194" "140" "86"  "34"  "360"
[484] "279" "200" "34"  "4"   "342" "449" "434" "256" "25"  "158" "453" "127" "286" "190" "370" "464" "341" "250" "198" "456" "220"
[505] "444" "241" "34"  "66"  "312" "380" "43"  "383" "297" "31"  "163" "399" "107" "329" "180" "79"  "129" "249" "185" "350" "2"
[526] "134" "361" "498" "191" "380" "372" "310" "77"  "359" "475" "430" "284" "30"  "426" "391" "147" "15"  "169" "399" "149" "91"
[547] "304" "96"  "307" "95"  "443" "80"  "494" "268" "378" "355" "493" "156" "180" "313" "88"  "172" "358" "493" "14"  "127" "26"
[568] "27"  "95"  "363" "398" "438" "187" "333" "84"  "36"  "367" "489" "81"  "251" "244" "50"  "405" "188" "462" "320" "59"  "215"
[589] "324" "19"  "34"  "78"  "123" "477" "502" "427" "72"  "334" "307" "260" "390" "186" "201" "408" "314" "468" "373" "15"  "167"
[610] "404" "394" "317" "210" "58"  "402" "18"  "375" "222" "104" "17"  "457" "132" "148" "151" "47"  "185" "497" "232" "108" "466"
[631] "438" "436" "141" "131" "433" "185" "122" "12"  "130" "331" "492" "65"  "213" "288" "143" "128" "317" "75"  "340" "67"  "421"
[652] "181" "250" "116" "473" "34"  "70"  "121" "7"   "392" "435" "83"  "192" "467" "69"  "250" "348" "245" "216" "277" "251" "345"
[673] "344" "361" "183" "263" "284" "449" "242" "318" "204"
```

```
> duplicated(eulerian_cycle)
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [22] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
 [43] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [64] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
 [85] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
[106] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[127] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[148] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
[169] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
[190] FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[211] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
[232] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
[253] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
[274] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
[295] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
[316] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[337]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE
[358]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
[379] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
[400]  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[421]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
[442] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
[463] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
[484] FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
[505] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE
[526] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
[547] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
[568] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
[589]  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE
[610] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
[631]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
[652]  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
[673] FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
> |
```

Figure 16, the Eulerian tour result

From the concept of TSP: TSP should visit every node only once. Next, we want remove the repeat nodes in Euler circuit in order to find an approximate tour for the TSP.

$$1\ w(MST) < w(TSP) < 2\ w(MST)$$

Therefore, we can get result in Figure 10.

Can you give a guarantee on the globally optimality of your solution?

Since the TSP is an NP-Complete problem, it indicates that we cannot use the computer in the polynomial time to find the global optimal solution.

Because of this, most decision tree algorithms use heuristic algorithms to solve TSP, such as greedy algorithms, to guide the search for hypothetical spaces. It can be said that the final result of the decision tree is the best choice at each step, every node. The result of the decision tree is that it is not guaranteed to be globally optimal.

6. Constructing Correlation Graphs for weekly data:
In the first part, we used daily closing prices for stocks to compute returns. Now, sample the stock data weekly on Mondays.

```
--------------------------Processing Finshed 1 --------------------------------
 Successful calculated all log return values in each file.
 The total number of files we processed:  505
 The total number of log return values in single file:  142
 --------------------------------------------------------------------------------
```

```
file_list                    Large list (505 elements, 576.6 Kb)
  : num [1:142] 0.0193 -0.0259 0.0343 0.0412 -0.0163 ...
  : num [1:142] 0.0735 -0.021 0.0664 0.0559 -0.0598 ...
  : num [1:142] 0.0343 -0.0198 0.0135 0.0257 -0.0214 ...
  : num [1:142] -0.0136 0.0196 0.039 -1.9035 -0.0161 ...
  : num [1:142] 0.019474 0.032685 0.000739 -0.005741 0.002967 ...
  : num [1:142] 0.0238 0.0404 0.0532 -0.012 -0.0114 ...
  : num [1:142] 0.01101 0.00913 0.00428 0.00826 -0.00575 ...
  : num [1:142] 0.0024 0.00466 0.02127 0.03613 -0.01724 ...
  : num [1:142] -0.0121 0.0244 0.0385 0.0348 0.0103 ...
  : num [1:142] 0.02031 -0.0048 0.00747 0.05462 0.00343 ...
  : num [1:142] 0.02861 -0.01762 0.02164 0.00645 -0.02059 ...
  : num [1:142] 0.01301 -0.0107 0.0169 0.00778 -0.01702 ...
  : num [1:142] 0.0101 -0.0118 0.0842 0.0243 0.0237 ...
```

Figure 17, each file have 142 log return value

And then calculate pij based on weekly data.

```
-------------------------Processing Finshed 2 -------------------------------
 Successful calculated the cross correlation coefficient of two different stock-return time series.
 The total number of p value we processed:  255025
 ------------------------------------------------------------------------------
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	1.000000000	0.28049998	0.20560568	-0.0169117998	0.383900005	0.242637908	0.49465790	0.45315775	0.446951468	0.369857394	0.365750
2	0.280499981	1.00000000	0.40779673	-0.0176108549	0.232668799	0.375195815	0.39647841	0.44009771	0.481106724	0.494662654	0.395713
3	0.205605679	0.40779673	1.00000000	-0.0211011476	0.265592188	0.166747389	0.28888975	0.31825995	0.372301669	0.437698925	0.330117
4	-0.016911800	-0.01761085	-0.02110115	1.0000000000	0.084846794	0.119712041	0.09971504	-0.02131846	0.015460916	-0.014432181	0.044280
5	0.383900005	0.23266880	0.26559219	0.0848467937	1.000000000	0.481706103	0.55058891	0.26354906	0.326913384	0.290861698	0.270687
6	0.242637908	0.37519581	0.16674739	0.1197120411	0.481706103	1.000000000	0.48768040	0.19150869	0.344640320	0.338502608	0.202470
7	0.494657903	0.39647841	0.28888975	0.0997150357	0.550588912	0.487680397	1.00000000	0.46401909	0.512451071	0.502988691	0.431337
8	0.453157753	0.44009771	0.31825995	-0.0213184612	0.263549056	0.191508687	0.46401909	1.00000000	0.618209902	0.547636878	0.505155
9	0.446951468	0.48110672	0.37230167	0.0154609156	0.326913384	0.344640320	0.51245107	0.61820990	1.000000000	0.565271555	0.479616
10	0.369857394	0.49466265	0.43769893	-0.0144321812	0.290861698	0.338502608	0.50298869	0.54763688	0.565271555	1.000000000	0.491905
11	0.365750992	0.39571351	0.33011767	0.0442802267	0.270686996	0.202470591	0.43133704	0.50515565	0.479616982	0.491905897	1.000000
12	0.279529423	0.46921643	0.31616077	0.0386430586	0.322529508	0.348564603	0.44358783	0.58025242	0.566643917	0.553708074	0.508146
13	0.334239313	0.51363797	0.40847783	0.0328800498	0.411477866	0.359778655	0.47297303	0.43661948	0.494039159	0.502373687	0.330547
14	0.434356086	0.48875635	0.27843208	0.0090905357	0.326187678	0.364559746	0.44278560	0.46092849	0.538693223	0.443494504	0.414073
15	0.047311003	0.21929330	0.18177637	0.0468922189	0.133372202	0.029882190	0.20501082	0.29617041	0.281385196	0.168733679	0.281379
16	0.023369825	0.26717358	0.16288934	0.0469478871	0.122048839	0.021782957	0.19917235	0.26873583	0.276367300	0.140369493	0.250392
17	0.303361447	0.32172144	0.32064347	0.0103935022	0.306599778	0.175591338	0.36302588	0.45108957	0.437335861	0.424254512	0.620733
18	0.348938358	0.35484544	0.35196646	0.0265469774	0.525013891	0.540480158	0.47391305	0.43364121	0.492495051	0.434550371	0.350995
19	0.468242760	0.41543294	0.30836874	-0.0099391191	0.325855786	0.296628730	0.52166817	0.56591363	0.465422820	0.507634682	0.436917
20	0.391462541	0.32310910	0.23741413	0.1204621322	0.540338973	0.527505458	0.50028328	0.24549606	0.383245537	0.316143825	0.240259
21	0.432237080	0.42894029	0.35310683	0.0370151467	0.373049685	0.408629421	0.47129356	0.49707806	0.484799280	0.513706578	0.506612
22	0.170753845	0.28407785	0.19837304	0.0545981935	0.214875137	0.259096395	0.31063571	0.39750908	0.395970547	0.274572375	0.309989
23	0.235405620	0.30672122	0.31385710	0.0152711302	0.175191113	0.213566725	0.22185296	0.44357274	0.267187302	0.284720751	0.267632
24	0.375467535	0.46775686	0.31254728	0.1023743650	0.426283245	0.424959314	0.56501164	0.52238389	0.480458383	0.545842406	0.425035

Figure 17, the P value we calculate and store it into matrix

The way to get the D value is same as the in the problem 2. And the result is in the Figure 18.

```
-------------------------Processing Finshed 3 -------------------------------
 Successful calculated the length of the link connecting two different stock return time series i, j.
 The total number of d value we processed:  255025
 ------------------------------------------------------------------------------
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
1	0.0000000	1.1995833	1.2604716	1.426122	1.1100450	1.2307413	1.0053279	1.0457937	1.0517115	1.1226243	1.1262762	1.2003921	1.1539157	1.06
2	1.1995833	0.0000000	1.0883044	1.426612	1.2388149	1.1178588	1.0986552	1.0582082	1.0187181	1.0053232	1.0993512	1.0303238	0.9862677	1.01
3	1.2604716	1.0883044	0.0000000	1.429056	1.2119470	1.2909319	1.1925689	1.1676815	1.1204448	1.0604726	1.1574820	1.1694779	1.0876784	1.20
4	1.4261219	1.4266120	1.4290564	0.000000	1.3528882	1.3268670	1.3418532	1.4292085	1.4032385	1.4243821	1.3825482	1.3866196	1.3907695	1.40
5	1.1100450	1.2388149	1.2119470	1.352888	0.0000000	1.0181296	0.9480623	1.2136317	1.1602471	1.1909142	1.2077359	1.1640193	1.0849167	1.16
6	1.2307413	1.1178588	1.2909319	1.326867	1.0181296	0.0000000	1.0122446	1.2716063	1.1448665	1.1502151	1.2629564	1.1414337	1.1315665	1.12
7	1.0053279	1.0986552	1.1925689	1.341853	0.9480623	1.0122446	0.0000000	1.0353559	0.9874704	0.9970068	1.0664548	1.0549049	1.0266713	1.05
8	1.0457937	1.0582082	1.1676815	1.429208	1.2136317	1.2716063	1.0353559	0.0000000	0.8738308	0.9511710	0.9948310	0.9162397	1.0614900	1.03
9	1.0517115	1.0187181	1.1204448	1.403238	1.1602471	1.1448665	0.9874704	0.8738308	0.0000000	0.9324467	1.0201794	0.9309738	1.0059432	0.96
10	1.1226243	1.0053232	1.0604726	1.424382	1.1909142	1.1502151	0.9970068	0.9511710	0.9324467	0.0000000	1.0080616	0.9447666	0.9976235	1.05
11	1.1262762	1.0993512	1.1574820	1.382548	1.2077359	1.2629564	1.0664548	0.9948310	1.0201794	1.0080616	0.0000000	0.9918203	1.1571109	1.08
12	1.2003921	1.0303238	1.1694779	1.386620	1.1640193	1.1414337	1.0549049	0.9162397	0.9309738	0.9447666	0.9918203	0.0000000	1.0603909	1.07
13	1.1539157	0.9862677	1.0876784	1.390770	1.0849167	1.1315665	1.0266713	1.0614900	1.0059432	0.9976235	1.1571109	1.0603909	0.0000000	1.05
14	1.0636202	1.0111811	1.2013059	1.407771	1.1608724	1.1273334	1.0556651	1.0383367	0.9605277	1.0549934	1.0825213	1.0747899	1.0532873	0.00
15	1.3803543	1.2495653	1.2792370	1.380658	1.3165317	1.3929234	1.2609434	1.1864481	1.1988451	1.2893924	1.1988498	1.1772086	1.2952006	1.27
16	1.3975909	1.2106415	1.2939170	1.380617	1.3251046	1.3987259	1.2655652	1.2093504	1.2030234	1.3112059	1.2244241	1.1865429	1.2602184	1.33
17	1.1803716	1.1647133	1.1656385	1.406845	1.1776249	1.2840628	1.1286932	1.0477695	1.0608149	1.0730755	0.8709380	1.0649216	1.1697578	1.13
18	1.1411062	1.1359177	1.1384494	1.395316	0.9746652	0.9586656	1.0257553	1.0642921	1.0074770	1.0634375	1.1393018	1.0860745	1.0755507	1.16
19	1.0312684	1.0812651	1.1761218	1.421224	1.1611582	1.1860618	0.9780918	0.9317579	1.0339992	0.9923359	1.0612097	0.8710696	1.0942626	1.03
20	1.1032112	1.1635213	1.2349784	1.326302	0.9588128	0.9721055	0.9997167	1.2284168	1.1106345	1.1694923	1.2326721	1.2453391	1.0851123	1.16
21	1.0656105	1.0686999	1.1374473	1.387793	1.1197770	1.0875390	1.0283058	1.0029177	1.0150869	0.9861982	0.9933651	1.0270356	1.0892505	1.03
22	1.2878246	1.1965970	1.2661966	1.375065	1.2530961	1.2172950	1.1741927	1.0977166	1.0991173	1.2045145	1.1747428	1.1462235	1.2595277	1.26
23	1.2366037	1.1775218	1.1714460	1.403374	1.2843745	1.2541398	1.2475152	1.0549192	1.2106302	1.1960596	1.2102625	1.1143170	1.2327070	1.21
24	1.1176157	1.0317394	1.1725636	1.339870	1.0711832	1.0724185	0.9327254	0.9773598	1.0193543	0.9530557	1.0723478	0.9153999	1.0196492	0.95

Figure 18, the D value we calculate and store it into matrix

And we use same way to construct the graph, which is adjacency matrix to get the correlation graph in the problem 2. And the result is in the Figure 19.

```
-------------------------Processing Finshed 4 ------------------------------
 Successful store graph into .txt file.
 The file store at /home/weikun/Downloads/finance_data/graph2.txt
 ----------------------------------------------------------------------------
```

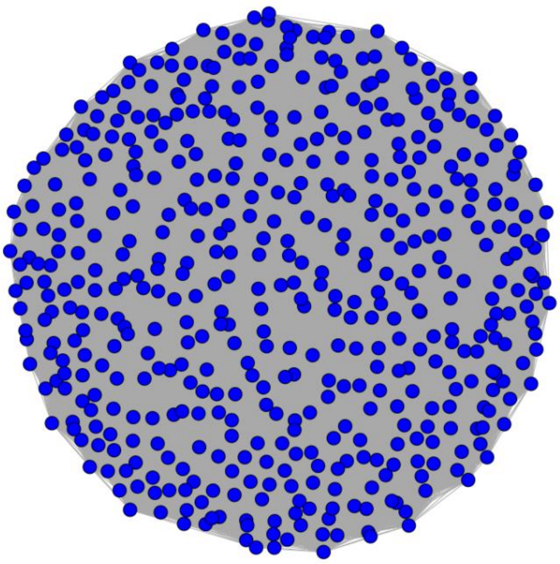

Figure 19, the plot for correlation graph 2

Determine the related MST and compare the two results: based on daily and weekly stock prices.

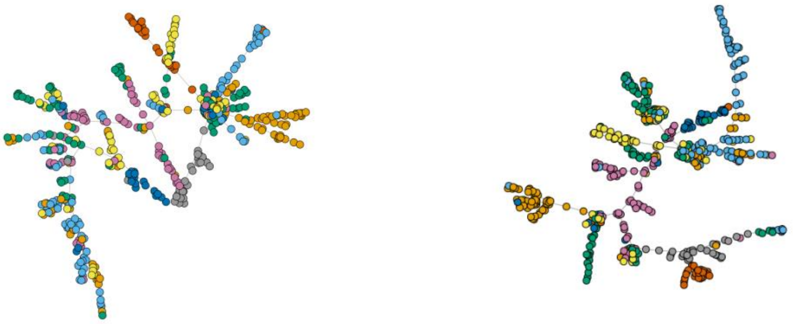

Figure 20, left graph is based on weekly stock prices and right graph is based on daily stock prices in problem 2

From the two plots, we can see the right graph have more clearly cluster in the branch of the Stock Minimum Spanning Tree (MST). Also, we can calculate the performance in order to see the difference. The performance based on daily stock prices is 81.4%. However, the performance based on the weekly stock prices is just 68.91%. Therefore, we can see the Pearson correlation coefficient is not accuracy base on the weekly stock prices.

```
-------------------------Processing Finshed 1 ------------------------------
 Successful evaluating sector clustering in Minimum Spanning Trees (MSTs)
 The proformace is calculated as:  0.6891179
 ---------------------------------------------------------------------------
```

Figure 21, the result for evaluating sector clustering in MSTs based on weekly stock prices

7. Modifying Correlations:

Plot the histogram of pij ’s from daily data.

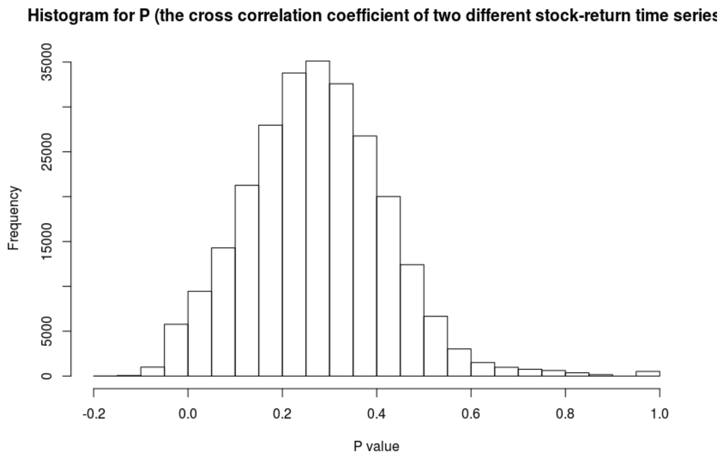

Figure 22, the histogram of the P

Here, the P value is same as problem 2 and the stack log return value is same way to calculate in problem 2, which is showed in Figure 23.

```
-------------------------Processing Finshed 1 ------------------------------
 Successful calculated all log return values in each file.
 The total number of files we processed:  505
 The total number of log return values in single file:  764
 ---------------------------------------------------------------------------

file_list                       Large list (505 elements, 2.9 Mb)
  : num [1:764] 0.001284 0.014381 -0.0058 0.000363 0.004894 ...
  : num [1:764] 0.00274 0.00301 0.00191 0.01489 0.0273 ...
  : num [1:764] 0.00585 -0.00544 -0.00447 -0.00867 0.02489 ...
  : num [1:764] 0.00186 0.01404 -0.01096 -0.00351 -0.00735 ...
  : num [1:764] -0.00837 0.00351 -0.0088 0.03626 -0.0122 ...
  : num [1:764] -0.00929 0.01283 -0.00508 -0.00356 -0.00217 ...
  : num [1:764] -0.00285 0.00724 -0.00672 0.00285 0.00103 ...
  : num [1:764] -0.00603 -0.00455 -0.00342 -0.00663 0.0079 ...
  : num [1:764] -0.01659 -0.00195 -0.03309 -0.01372 0.00765 ...
  : num [1:764] 0.01154 -0.00448 -0.00863 0.00118 0.00999 ...
  : num [1:764] 0.00714 -0.00322 -0.00624 0.02586 -0.00453 ...
  : num [1:764] -0.00103 0.00425 -0.01046 0.00518 0.00502 ...
  : num [1:764] -0.0033 0.00534 -0.00484 -0.0226 0.00784 ...
```

Figure 23, each file have 764 log return value

Then, set all pij 's larger than 0.3 as -1; for pij <= 0:3, keep the original value.

```
-------------------------Processing Finshed 2 ------------------------------
 Successful calculated the cross correlation coefficient of two different stock-return time series.
 The total number of p value we processed:  255025
 ---------------------------------------------------------------------------
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	-1.000000000	0.29248048	0.19391830	0.0571720555	-1.000000000	0.220241092	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
2	0.292480478	-1.00000000	0.25874995	0.0678734920	0.253767258	0.223827142	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	0.2461
3	0.193918297	0.25874995	-1.00000000	0.0566843362	0.237723861	0.162557620	-1.000000000	0.25991627	0.281453555	0.2512762213	0.2729
4	0.057172056	0.06787349	0.05668434	-1.0000000000	0.099870015	0.049542181	0.072912942	0.05864815	0.080518847	-0.0133238068	0.0576
5	-1.000000000	0.25376726	0.23772386	0.0998700152	-1.000000000	-1.000000000	-1.000000000	0.26406465	-1.000000000	0.2965578985	0.2717
6	0.220241092	0.22382714	0.16255762	0.0495421812	-1.000000000	-1.000000000	-1.000000000	0.24834295	0.213696119	0.1910042462	0.1601
7	-1.000000000	-1.00000000	-1.00000000	0.0729129421	-1.000000000	-1.000000000	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
8	-1.000000000	-1.00000000	0.25991627	0.0586481484	0.264064650	0.248342951	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
9	-1.000000000	-1.00000000	0.28145355	0.0805188475	-1.000000000	0.213696119	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
10	-1.000000000	-1.00000000	0.25127622	-0.0133238068	0.296557898	0.191004246	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
11	-1.000000000	0.24613845	0.27297252	0.0576160668	0.271724442	0.160173474	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
12	-1.000000000	-1.00000000	-1.00000000	0.0603901922	-1.000000000	-1.000000000	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
13	-1.000000000	-1.00000000	0.27692221	0.0517015929	0.240643272	0.267046329	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	0.2421
14	-1.000000000	-1.00000000	0.20898415	0.0871083407	-1.000000000	0.224167751	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
15	0.123232676	0.10941170	0.16655696	0.0611790064	0.195378681	0.100642312	0.263795123	0.26168402	0.248862562	0.1473683325	0.2275
16	0.139569748	0.11964231	0.17389619	0.0697679533	0.172221535	0.096084874	0.256961259	0.27680933	0.267278628	0.1654309001	0.2257
17	0.273831359	0.20311249	0.23272852	0.0593845887	0.219454147	0.128895545	-1.000000000	-1.00000000	-1.000000000	0.2975970188	-1.0000
18	-1.000000000	0.28820470	0.24954277	0.0673337045	-1.000000000	-1.000000000	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
19	-1.000000000	-1.00000000	-1.00000000	0.0416676041	-1.000000000	0.287131471	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
20	0.294112217	0.27861678	0.22866978	0.0622264393	-1.000000000	-1.000000000	-1.000000000	0.25216655	-1.000000000	0.2880094625	0.2209
21	-1.000000000	-1.00000000	-1.00000000	0.0921058074	-1.000000000	-1.000000000	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000
22	0.207936874	0.19198593	0.18730006	0.0821827483	0.235945618	0.141559267	-1.000000000	-1.00000000	-1.000000000	0.2356908295	0.2658
23	0.282633876	0.23592591	0.27810631	0.0676964818	0.193626793	0.195201817	-1.000000000	-1.00000000	0.297726557	-1.0000000000	0.2783
24	-1.000000000	-1.00000000	-1.00000000	0.0853486129	-1.000000000	-1.000000000	-1.000000000	-1.00000000	-1.000000000	-1.0000000000	-1.0000

Figure 24, the modify P value we calculate and store it into matrix

Calculate $d_{ij} = (2 * (1 - p_{ij}))$^(-2)

```
-------------------------Processing Finshed 3 ------------------------------
 Successful calculated the length of the link connecting two different stock return time series i, j.
 The total number of d value we processed:  255025
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
1	2.000000	1.189554	1.269710	1.373192	2.000000	1.248807	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.32421
2	1.189554	2.000000	1.217580	1.365377	1.221665	1.245932	2.000000	2.000000	2.000000	2.000000	1.227894	2.000000	2.000000	2.000000	1.33460
3	1.269710	1.217580	2.000000	1.373547	1.234728	1.294173	2.000000	1.216621	1.198788	1.223702	1.205842	2.000000	1.202562	1.257788	1.29107
4	1.373192	1.365377	1.373547	2.000000	1.341738	1.378737	1.361681	1.372117	1.356083	1.423604	1.372868	1.370846	1.377170	1.351215	1.37027
5	2.000000	1.221665	1.234728	1.341738	2.000000	2.000000	2.000000	1.213207	2.000000	1.186121	1.206877	2.000000	1.232361	2.000000	1.26855
6	1.248807	1.245932	1.294173	1.378737	2.000000	2.000000	2.000000	1.226097	1.254037	1.272003	1.296014	2.000000	1.210747	1.245658	1.34116
7	2.000000	2.000000	2.000000	1.361681	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.21342
8	2.000000	2.000000	1.216621	1.372117	1.213207	1.226097	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.21516
9	2.000000	2.000000	1.198788	1.356083	2.000000	1.254037	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.22567
10	2.000000	2.000000	1.223702	1.423604	1.186121	1.272003	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.30585
11	2.000000	1.227894	1.205842	1.372868	1.206877	1.296014	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.231116	2.000000	1.24297
12	2.000000	2.000000	2.000000	1.370846	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.20414
13	2.000000	2.000000	1.202562	1.377170	1.232361	1.210747	2.000000	2.000000	2.000000	2.000000	1.231116	2.000000	2.000000	2.000000	1.34105
14	2.000000	2.000000	1.257788	1.351215	2.000000	1.245658	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.30497
15	1.324211	1.334607	1.291079	1.370271	1.268559	1.341162	1.213429	1.215167	1.225673	1.305857	1.242972	1.204144	1.341050	1.304978	2.00000
16	1.311816	1.326920	1.285382	1.363988	1.286684	1.344556	1.219048	1.202656	1.210555	1.291951	1.244377	1.186987	1.326428	1.300963	2.00000
17	1.205130	1.262448	1.238767	1.371580	1.249437	1.319928	2.000000	2.000000	2.000000	1.185245	2.000000	2.000000	1.238322	1.211348	2.00000
18	2.000000	1.193143	1.225118	1.365772	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.27755
19	2.000000	2.000000	2.000000	1.384437	2.000000	1.194042	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.23351
20	1.188182	1.201152	1.242039	1.369506	2.000000	2.000000	2.000000	1.222975	2.000000	1.193307	1.248214	1.200043	1.194153	1.219472	1.31787
21	2.000000	2.000000	2.000000	1.347512	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.30611
22	1.258621	1.271231	1.274912	1.354856	1.236167	1.310298	2.000000	2.000000	2.000000	1.236373	1.211722	2.000000	1.265098	1.241750	2.00000
23	1.197803	1.236183	1.201577	1.365506	1.269940	1.268699	2.000000	2.000000	1.185136	2.000000	1.201356	2.000000	1.214099	1.185179	1.26041
24	2.000000	2.000000	2.000000	1.352517	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.20663
25	2.000000	1.188769	1.275478	1.396957	1.245483	1.258906	2.000000	2.000000	2.000000	2.000000	1.185832	2.000000	2.000000	2.000000	1.31881
26	2.000000	1.255124	1.203702	1.369332	1.208178	1.306060	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.195474	2.000000	1.29006
27	1.276794	2.000000	1.287900	1.376965	1.305016	1.317414	1.251226	1.253845	1.260745	1.273433	1.360337	1.226867	1.237594	1.250419	1.36816
28	2.000000	1.213359	1.236359	1.354954	1.244755	1.264976	2.000000	2.000000	1.184789	2.000000	2.000000	2.000000	1.205609	1.200907	1.18667
29	2.000000	2.000000	2.000000	1.377713	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.27087
30	2.000000	2.000000	1.281516	1.352105	2.000000	1.197336	2.000000	2.000000	2.000000	2.000000	1.207637	2.000000	2.000000	2.000000	1.35479

Figure 25, the modify D value we calculate and store it into matrix

Construct the graph and run MST.

```
--------------------------Processing Finshed 4 -------------------------------
 Successful store graph 3 into .txt file.
 The file store at /home/weikun/Downloads/finance_data/graph3.txt
 ----------------------------------------------------------------------------
```

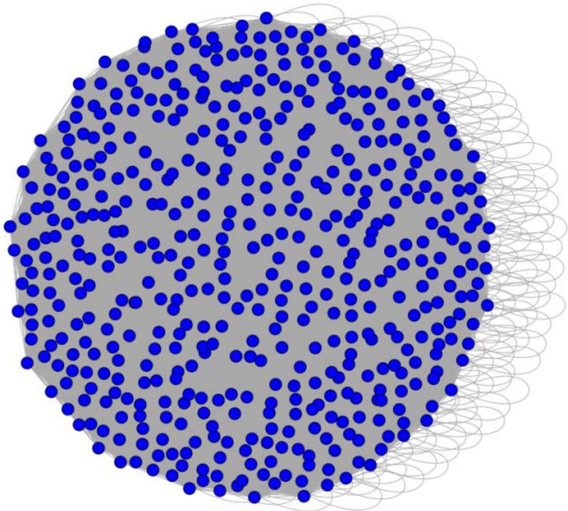

Figure 26, the plot for correlation graph 3

Do you see vine cluster patterns anymore with modified correlations? Can you explain why?

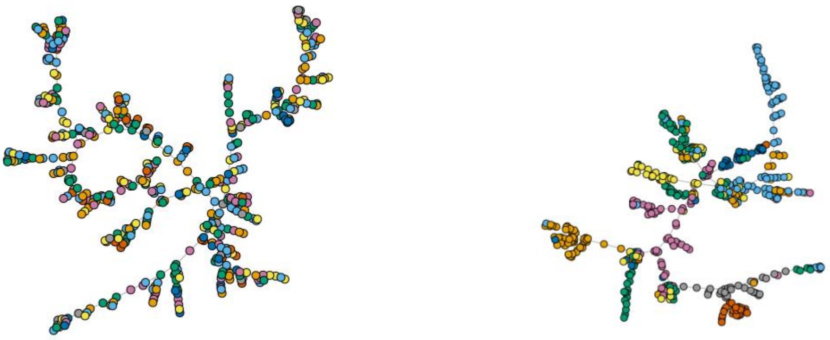

Figure 27, left graph is MST base on modifying correlations the and right graph is Vine Clusters

Here, I want to explain why cluster patterns with modified. As I explain in the problem 2, the concept of Pearson correlation coefficient and Pearson's distance. From the Pearson correlation concept, the range is from 1 to -1. The value at 1 means two variables have strong positive correlation, and the value -1 means two variables have strong negative correlation. We modify all P larger than 0.3 to be -1 means that we change all strong positive correlation to be strong negative correlation. D is come from Pearson's distance: a distance metric for two variables X and Y defined from their correlation coefficient. Therefore, the range of D is from 0 to 2 and the original D close to the 0 becomes 2 right now. Also, we know D can be converting as weight in the correlation graph. Recall the MST concept is that: A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. Therefore, the original edge in the MST will not appear in the modify MST because the original edge weight change from small to maximum.

Also we can post the performance is only 16.06% for the modify MST, showed in the Figure 28.

```
-------------------------Processing Finshed 1 -------------------------------
 Successful evaluating sector clustering in Minimum Spanning Trees (MSTs)
 The proformace is calculated as:  0.1606318
 ----------------------------------------------------------------------------
```

Figure 28, the result for evaluating sector clustering in MSTs based modify P value

Conclusion

In general, the results meet our expectations.

References

[1] ftp://cran.r-project.org/pub/R/web/packages/igraph/igraph.pdf
[2] http://www.ams.sunysb.edu/~estie/courses/301/app_tsp.pdf
[3] https://cran.r-project.org/web/packages/PairViz/PairViz.pdf
[4] http://bioconductor.org/packages/release/bioc/manuals/graph/man/graph.pdf