OPEN SHORTEST PATH FIRST

PROJECT REPORT FOR CS254

Presented by

Kunal Gupta (150001015)

Keshav Goyal (150001014)

Bhor Verma (150001005)

Computer Science and Engineering

2^{nd} Year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2017

CONTENTS

INTRODUCTION

In computer networks, the main aim is to share resources or data among the networked nodes. This becomes fairly complicated when networks are large and data needs to be transferred from any node to any other because if the node cannot directly connect to the destination node, it has to send it via other nodes along a proper route to the destination node. And since most nodes do not try to figure out which route(s) might work, we use routers as special nodes in the network.

Routers help to select a path for sending data packets in the network by a process called routing. Routing protocols help routers to determine which path should be chosen and how routers should communicate with each other, disseminating information that enables them to select the required route.

Although there are many types of routing protocols, we will be limiting our research to OSPF, an interior gateway protocol.

OSPF or Open Shortest Path First [1] is the routing protocol used for Internet Protocol (IP) networks, using a link state routing algorithm [2].

The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a *graph*, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path[1] from it to every possible destination in the network. Each collection of best paths will then form each node's routing table, which contains information about the topology of the network immediately around it.

In addition to just selecting the optimal path, the protocol also involves updating the routing table whenever one or more routers go to sleep and/or new routers are added.

[1] The best logical path is the path through which data packet takes the shortest time to reach from source to the destination.

MOTIVATION

Computer networking not only involves creating networks by interconnecting nodes, its heart lies in sending data along the network to and from the nodes. Minimizing the time taken by the packet to reach the destination results in faster connectivity, low ping times and stable connections.

The biggest application of OSPF or any other routing protocol for that matter is that it is used in Internet Protocol (IP) which enables internetworking, and essentially establishes the internet.

OSPF was the first widely deployed routing protocol that could converge a network in the low seconds, and guarantee loop-free paths. This historical significance of OSPF was also one of the factors that motivated us to choose it as a topic for our project.

OBJECTIVES

In our project we will be concerned with the creation of routing table in OSPF.

Thus our objectives will be to:

- Analyze the different algorithms which can be used for OSPF and select the best out of them.
- Analyze the algorithm for finding the shortest path and filling in the routing table.
- Further optimize the algorithm and discuss and remove its limitations if possible.

POSSIBLE METHODOLGY

We will make some assumptions for simplifying our analysis.

- The network will be treated as undirected, connected, edge-weighted graph.
- Each router in the network will be a node in the graph.
- The weighted edges represent the connections between routers. That is:
    - If two routers are directly connected, there will be an edge between them.
    - The round trip time between two directly connected as the weight of the edge connecting their corresponding nodes in the graph.
- Now, a path would represent the nodes (routers) a message would have to travel through and the edge it will traverse would represent the time taken to reach the end of the edge

Thus the problem reduces to finding the shortest-path in an undirected connected edge-weighted graph.

We have a number of options available.

BELLMAN FORD

Bellman Ford computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. Our graph is an undirected one, since the time taken for a packet to travel between two directly connected routers is independent of the direction of travel. Thus, necessary conversion of our graph from an undirected graph to a directed one to use the algorithm is required.

It can calculate shortest path in a graph with negative edges as well, but this is of no use since all the edge are non-negative.

We can use Bellman Ford for positive edges with a complexity of $O(|E||V|)$ per node.

Thus total complexity would be $O(|E||V|^2)$, since we need to compute all source to all destinations paths.

FLOYD-WARSHALL

Floyd-Warshall finds shortest path between all pairs of vertices which our objective. Thus it eliminates the need to apply the algorithm to each node.

It is generally a good choice for dense graphs. But ours will generally be a sparse one due to each router getting connected to a significantly low number of routers as compared to the square of total number of routers, i.e., the probability of each router being connected to every other router is *extremely* low.

Overall complexity for the algorithm would be $O(|V|^3)$

JOHNSON'S ALGORITHM

Johnson's Algorithm finds the shortest paths between all pairs of vertices in a sparse, edge weighted, directed graph. Again, we are dealing with undirected graph and thus it is necessary to perform a conversion like Bellman Ford.

Overall complexity for Johnson's Algorithm is $O(|V|^2\ log|V| + |V||E|)$

DIJKSTRA'S ALGORITHM

For a given source node in the graph, Dijkstra's Algorithm finds the shortest path between that node and every other.

Our requirement is to calculate shortest path for each node to all nodes. Thus, a possible solution is to run Dijkstra on every node.

By using Fibonacci Heap, the complexity per node comes out to be $O(|E| + |V|\ log|V|)$ [3].

Thus overall complexity is $O(|E||V| + |V|^2\ log|V|)$

SELECTING THE BEST

We can easily observe that Dijkstra's algorithm seems to be best way for finding the shortest path from all vertices to all vertices. It seems fit because:

- We have a graph with positive weights
- We have an undirected Graph
- It is more efficient even when applied on all nodes.
- We have a sparse graph; thus it works even better than others like Floyd-Warshall.

Thus we will now analyze this algorithm in reference to our problem.

ALGORITHM DESIGN

Given a Graph $G(V, E)$, Dijkstra's algorithm will require 3 inputs:

- The graph G containing the vertices and edges
- The weights w
- The source vertex s

DESIGN

The algorithm is outlined as follows:

$dijkstra\ (G, w, s)$:

$\quad d[s] = 0$

$\quad for\ each\ v \in V - \{s\}$

$\quad\quad do\ d[v] = \infty$

$\quad S = \emptyset$

$\quad Q = s$

$\quad while\ Q \neq \emptyset$

$\quad\quad do\ u = ExtractMin(Q)$

$\quad\quad S = S \cup \{u\}$

$\quad\quad for\ each\ v \in adj\ \{u\}$

$\quad\quad\quad do\ if\ d[v] > d[u] + w(u, v)\ then$

$\quad\quad\quad\quad d[v] = d[u] + w(u, v)$

$\quad\quad\quad\quad Q = Q \cup \{v\}$

INITIALISATION

Dijkstra's algorithm must first initialize its three important arrays. First, the array S contains the vertices that have already been examined or relaxed. It first starts as the empty set, but as the algorithm progresses, it will fill it with each vertex until all are examined. Then, the distance array d[x] is defined to be an array of the shortest paths from s to x, or also denoted $\delta(s, x)\ when\ x \in S$. Finally, Q is simply the data type used to form the list of vertices.

SHORTEST PATH CALCULATION

After the initialization portion of the algorithm is complete, we then move into the shortest path calculation. The function will have to run as long as it takes to relax each edge for each vertex. Next, we chose to use "ExtractMin" because we needed a method to choose a new vertex to examine. Extracting the vertex corresponding to the shortest path so far, will guarantee choosing a new unique vertex. We must compare every edge that connects to this newly chosen vertex u. If the adjacent vertex v currently has a distance to the source that is greater than the distance to u plus the cost of the distance between u and v, then we must update the distance to v.

After completion of this step, we now have an array d[x] that holds the value for the shortest distance for from the source to each of the vertices in the graph.

ALGORITHM ANALYSIS

The following analysis is for the built in `priority_queue` (binary heap) in C++.

Algorithm	Analysis
$dijkstra\ (G, w, s)$:	
$d[s] = 0$	
$for\ each\ v \in V - \{s\}$	V times
$do\ d[v] = \infty$	
$S = \emptyset$	
$Q = \mathrm{s}$	
a) $while\ Q \neq \emptyset$	Max $(\lvert E\rvert + \lvert V\rvert)$ times
$do\ u = ExtractMin(Q)$	Time $= O(log\lvert V\rvert)$
$S = S \cup \{u\}$	
b) $for\ each\ v \in adj\ \{u\}$	$\lvert E\rvert$ times
$do\ if\ d[v] > d[u] + w(u, v)\ then$	
$d[v] = d[u] + w(u, v)$	
$Q = Q \cup \{v\}$	Time $= O(log\lvert V\rvert)$

The time complexity of the above code/algorithm looks $O(|V|^2)$ as there are two nested loops. If we take a closer look, we can observe that the statements in inner loop are executed $O(|V| + |E|)$ times (similar to BFS).

So overall time complexity is:

$$O(|E| + |V|) \cdot O(log|V|)$$

which is

$$O((|E| + |V|) \cdot log|V|) \ = \ O(|E|log|V|)$$

However in our analysis we have optimized our algorithm a bit further (see code) so as to have only the required vertices so as to improve average running time.

IMPLEMENTATION

We have implemented the algorithm in C++ using the built-in priority queue.

Here's the code for the same:

```
#include <iostream>
#include <queue>
#include <vector>
#include <climits>
#include <cmath>
#include <ctime>
using namespace std;
#define INF INT_MAX //Infinity

const int sz=10001;
int parent[10001];
vector<pair<int,int> > a[sz]; //Adjacency list

int dis[sz][sz]; //Stores shortest distance
 void Dijkstra(int source, int n) //Algorithm
{
    bool vis[sz]={0};
    for(int i=0;i<sz;i++)
        dis[source][i]=INF;
    for(int i=1;i<=n;i++)
    {
        parent[i]=source;
    }
    priority_queue< pair<int,int>,vector<pair<int,int> >,greater<
pair<int,int> > > pq;
    pq.push(make_pair(dis[source][source]=0,source));

    while(!pq.empty())
    {

        pair<int, int> curr=pq.top(); //Current vertex. The shortest distance
for this has been found
        pq.pop();
        int cv=curr.second,cw=curr.first; //'cw' the final shortest distance
for this vertex
    if(vis[cv])
            continue;

        vis[cv]=true;


        for(int i=0;i<a[cv].size();i++)          {
            if(a[cv][i].second+cw<dis[source][a[cv][i].first])
            {
                dis[source][a[cv][i].first]=a[cv][i].second+cw;
                dis[a[cv][i].first][source]=a[cv][i].second+cw;
                if(!vis[a[cv][i].first] )

                        pq.push(make_pair(dis[source][a[cv][i].first],
```

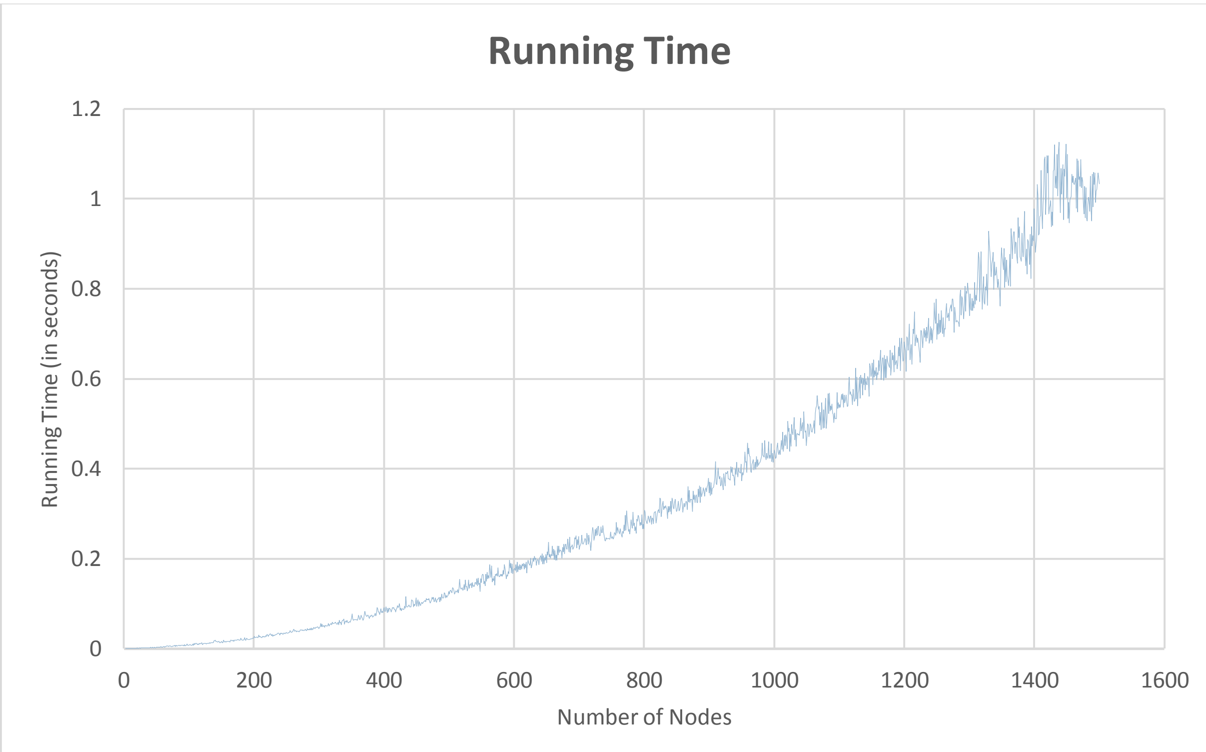
```
                        a[cv][i].first)); //Set the new distance and add to
                        priority queue
                parent[a[cv][i].first]=curr.second;
            }
        }
    }
}

int main() {
    int n,m,x,y,w,j,i,k;//Number of vertices and edges
        clock_t start, end;
        double msecs;
    cout<<"Enter number of vertices and edges in the graph\n";
    cin>>n>>m;
    for(i=0;i<m;i++) //Building Graph
    {
        cin>>x>>y>>w; //Vertex1, Vertex2, weight of edge
        a[x].push_back(make_pair(y,w));
        a[y].push_back(make_pair(x,w));
    }
    //cout<<"Enter source for Dijkstra's algorithm\n";
start = clock();
    for(j=1;j<=n;j++)
    {
        Dijkstra(j,n);
    }
    end = clock();
    msecs = ((double) (end - start))/ CLOCKS_PER_SEC;
    cout <<msecs<<endl;
    return 0;
}
```

RESULTS

We ran 1500 randomly generated test cases (all Test Cases were self-generated and Running Time for Each Test Case can be found in Appendix).

The number was edges was 1 less than the number of vertices in each case.

With the Data we got the following plot:



CONCLUSION AND FUTURE WORK

We observe that the algorithm takes small times for number of nodes even upto 700 routers. This shows that Dijkstra, when used in large networks is still very efficient. This point is further strengthened by the fact that OSPF generally uses Dijkstra based implementation in many companies [4]: Cisco IOS, D-Link, Juniper Junos, NetWare, OpenBSD to name a few.

But we also see that over 1000 nodes the running time is large, going up to 0.5 seconds which is not very practical. Hence it is not advisable to use it when number of routers is greater than 1000 (1000 is a still quite a large number for number of routers).

But nowadays, when the latency time goes over 0.2 seconds, the ping isn't satisfactory enough. Thus the algorithmic calculation along with the practical ping time of the router will be too large (~0.4-0.5 seconds) for practical usage. Thus it would be advisable to stay under 600 routers for Dijkstra to work well.

ANALYSIS AND USAGE OF FIBONACCI HEAP

Fibonacci heap is a data structure we can use for priority queue operations, instead of the C++'s standard built-in `priority_queue` and has a better amortized (considers both the costly and less costly operations together over the whole series of operations of the algorithm) running time than many other priority queue data structures including the binary heap and binomial heap.

It consists of a collection minimum-heap-ordered trees, implying that the minimum key is always at the root of one of the trees.

Comparing them with binary heaps, the Fibonacci heaps have a more flexible structure. They do not have a prescribed shape and in the extreme case the heap can have every element in a separate tree. This flexibility allows some of the operations to be executed in a lazy [5] manner (postponing the work for later operations).

To allow fast deletion and concatenation, the roots of all trees are linked using a circular, doubly linked list. The children of each node are also linked using such a list. For each node, we maintain its number of children and whether the node is marked. Moreover, we maintain a pointer to the root containing the minimum key.

Thus, the operation to find the minimum value is now trivial because we keep the pointer to the node containing it.

The deletion operation starts by removing the minimum node from the root list and adding its children to the root list. If the minimum was the only node in the root list, the pointer to the minimum node is set to the smallest node in the root list and the operation is completed.

If not, all trees of the same order are merged together until there are no two trees of the same order. The minimum is then set to the smallest node in the root list.

For the Fibonacci heap, the find-minimum operation takes constant ($O(1)$) amortized time [6] and the deletion operation takes $O(log\ n)$ amortized time, where n is the size of the heap [7].

The running time for Dijkstra [8] can be written as:

$$O(|E| \cdot T_{dk} + |V| \ \cdot \ T_{em})$$

T_{dk} and T_{em} are the complexities of the decrease-key and extract-minimum operations in Q, respectively.

For Fibonacci Heap we have:

$T_{dk} = \ \Theta(1)$ and

$T_{em} = \Theta(\log(n))$

Thus, the overall complexity for Dijkstra improves to:

$$O(|E| + |V| log\ |V|)$$

BIBLIOGRAPHY

[1] J. Moy, “OSPF Version 2,” April 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328. [Accessed 28 09 2007].

[2] I. R. a. E. C. R. John M. McQuillan, “ARPANet Routing Algorithm Improvements,” in *BBN Report No. 3803*, Cambridge, April 1978.

[3] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms.,” in *25th Annual Symposium on Foundations of Computer Science. IEEE.*, 1984.

[4] Wikipedia, “Open Shortest Path First - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Open_Shortest_Path_First#Applications. [Accessed 19 April 2017].

[5] P. Hudak, “ Conception, Evolution, and Application of Functional Programming Languages,” *ACM Computing Surveys.,* vol. 21, no. 3, p. 384, 1989.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, “Chapter 20: Fibonacci Heaps,” in *Introduction to Algorithms (2nd ed.). ISBN 0-262-03293-7.*, MIT Press and McGraw-Hill, 1990, pp. 476-497.

[7] M. L. Fredman and R. E. Tarjan, “ Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the Association for Computing Machinery,* vol. 34, no. 3, pp. 596-615, 1987.

[8] Wikipedia, “Dijkstra's algorithm - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Running_time. [Accessed 19 April 2017].

APPENDIX

THE RUNTIME TABLE FOR THE GRAPH IN RESULT

The whole test data can be viewed at https://www.github.com/iamKunal/OSPF

Nodes	Running Time	40	0.002018	78	0.005958	115	0.009746	153	0.018866	191	0.021038	229	0.028387
3	0.000181	41	0.003381	79	0.007422	116	0.011495	154	0.015931	192	0.020365	230	0.030626
4	0.000235	42	0.002376			117	0.010784	155	0.015234	193	0.020321	231	0.028915
5	0.000251	43	0.002269	80	0.005513	118	0.012156	156	0.016411	194	0.0211	232	0.031148
6	0.000298	44	0.002993	81	0.007415	119	0.011797	157	0.014695	195	0.021553	233	0.031022
7	0.000342	45	0.002477	82	0.005802	120	0.01327	158	0.016127	196	0.02137	234	0.031775
8	0.000368	46	0.00272	83	0.006394	121	0.009216	159	0.01699	197	0.021293	235	0.031476
9	0.000419	47	0.003623	84	0.007777	122	0.010046	160	0.016416	198	0.023806	236	0.033797
10	0.000512	48	0.002761	85	0.007622	123	0.012352	161	0.016547	199	0.023209	237	0.034469
11	0.000497	49	0.002602	86	0.006831	124	0.011509	162	0.015869	200	0.023307	238	0.034829
12	0.000563	50	0.00379	87	0.007552	125	0.012096	163	0.015539	201	0.02734	239	0.03393
13	0.000578	51	0.003086	88	0.007302	126	0.012143	164	0.018903	202	0.024728	240	0.030094
14	0.000805	52	0.002847	89	0.00791	127	0.010522	165	0.019641	203	0.024491	241	0.034456
15	0.000899	53	0.004149	90	0.007907	128	0.012209	166	0.018235	204	0.025734	242	0.033786
16	0.00078	54	0.003267	91	0.008085	129	0.012615	167	0.01786	205	0.024528	243	0.03429
17	0.000926	55	0.003445	92	0.006797	130	0.011892	168	0.01915	206	0.024124	244	0.033373
18	0.001179	56	0.003809	93	0.00866	131	0.01133	169	0.016941	207	0.030144	245	0.031432
19	0.000941	57	0.004528	94	0.006795	132	0.013513	170	0.019154	208	0.026802	246	0.035249
20	0.000924	58	0.003287	95	0.008468	133	0.012088	171	0.019735	209	0.027839	247	0.03482
21	0.001496	59	0.003523	96	0.007568	134	0.013067	172	0.019561	210	0.026173	248	0.0338
22	0.001122	60	0.003656	97	0.009947	135	0.012743	173	0.019424	211	0.024442	249	0.034163
23	0.001167	61	0.005599	98	0.00753	136	0.013917	174	0.018993	212	0.024549	250	0.03572
24	0.001426	62	0.004388	99	0.007092	137	0.013574	175	0.020405	213	0.028082	251	0.035216
25	0.001302	63	0.004075	100	0.007079	138	0.012753	176	0.020115	214	0.0269	252	0.034255
26	0.001498	64	0.004858	101	0.009304	139	0.017063	177	0.019479	215	0.027693	253	0.034383
27	0.001601	65	0.004969	102	0.008777	140	0.017475	178	0.019034	216	0.026612	254	0.036094
28	0.002543	66	0.005963	103	0.008065	141	0.019134	179	0.018843	217	0.026162	255	0.036591
29	0.001581	67	0.005648	104	0.008333	142	0.016549	180	0.022755	218	0.02917	256	0.040238
30	0.00167	68	0.006535	105	0.008369	143	0.016559	181	0.020492	219	0.027973	257	0.037659
31	0.002618	69	0.004869	106	0.010092	144	0.014272	182	0.019802	220	0.031772	258	0.035836
32	0.001659	70	0.006527	107	0.008379	145	0.015988	183	0.019199	221	0.029829	259	0.037349
33	0.001609	71	0.006203	108	0.011638	146	0.015679	184	0.019818	222	0.026981	260	0.038236
34	0.001666	72	0.005658	109	0.009659	147	0.013748	185	0.021155	223	0.029948	261	0.041784
35	0.001867	73	0.004901	110	0.01143	148	0.014051	186	0.024291	224	0.031499	262	0.043627
36	0.002239	74	0.004643	111	0.011487	149	0.015499	187	0.019658	225	0.032704	263	0.039536
37	0.00184	75	0.00479	112	0.011679	150	0.015923	188	0.019184	226	0.032179	264	0.041
38	0.002075	76	0.00604	113	0.009441	151	0.013941	189	0.023073	227	0.02941	265	0.037839
39	0.002304	77	0.005791	114	0.011083	152	0.013791	190	0.021579	228	0.031861	266	0.038356

267	0.038974	312	0.050985	357	0.06411	402	0.083734	447	0.09652	492	0.112578	537	0.138126
268	0.039328	313	0.05006	358	0.063998	403	0.092524	448	0.100289	493	0.12464	538	0.139974
269	0.038195	314	0.052026	359	0.063172	404	0.086469	449	0.112512	494	0.122713	539	0.138718
270	0.039711	315	0.056463	360	0.066447	405	0.079982	450	0.094693	495	0.11346	540	0.14779
271	0.041586	316	0.050318	361	0.075063	406	0.089767	451	0.102541	496	0.118309	541	0.147204
272	0.041434	317	0.05317	362	0.070783	407	0.081094	452	0.108999	497	0.115085	542	0.142568
273	0.041407	318	0.055184	363	0.06521	408	0.090429	453	0.094554	498	0.116783	543	0.155801
274	0.041172	319	0.056003	364	0.067702	409	0.089067	454	0.104304	499	0.118328	544	0.140517
275	0.042514	320	0.053022	365	0.074078	410	0.087839	455	0.09905	500	0.126768	545	0.157197
276	0.040265	321	0.059605	366	0.070189	411	0.081595	456	0.095832	501	0.121514	546	0.143436
277	0.040637	322	0.056511	367	0.06814	412	0.085888	457	0.100982	502	0.128049	547	0.145532
278	0.041969	323	0.055517	368	0.062836	413	0.091023	458	0.103729	503	0.129486	548	0.127345
279	0.042652	324	0.057451	369	0.066618	414	0.089748	459	0.100682	504	0.124893	549	0.155036
280	0.04031	325	0.058556	370	0.076678	415	0.091363	460	0.102623	505	0.136235	550	0.14695
281	0.039811	326	0.055856	371	0.082738	416	0.08674	461	0.103914	506	0.134146	551	0.163301
282	0.044883	327	0.054166	372	0.075376	417	0.087969	462	0.105883	507	0.129837	552	0.165713
283	0.042594	328	0.054625	373	0.073227	418	0.082137	463	0.108037	508	0.121284	553	0.163716
284	0.043007	329	0.059427	374	0.0716	419	0.086733	464	0.104505	509	0.124238	554	0.144286
285	0.041905	330	0.053792	375	0.074239	420	0.089644	465	0.111418	510	0.129357	555	0.150102
286	0.044184	331	0.063261	376	0.075103	421	0.096055	466	0.108403	511	0.127094	556	0.140542
287	0.042355	332	0.057001	377	0.070062	422	0.092109	467	0.110509	512	0.125826	557	0.165693
288	0.045809	333	0.059639	378	0.067411	423	0.094565	468	0.109247	513	0.131182	558	0.150645
289	0.041504	334	0.053576	379	0.073366	424	0.100169	469	0.111519	514	0.123791	559	0.169535
290	0.044441	335	0.059773	380	0.072061	425	0.085414	470	0.105606	515	0.132295	560	0.155069
291	0.043049	336	0.06393	381	0.07425	426	0.090755	471	0.102183	516	0.153703	561	0.162386
292	0.047033	337	0.063837	382	0.073956	427	0.084927	472	0.104373	517	0.136963	562	0.156816
293	0.045819	338	0.065497	383	0.070781	428	0.088513	473	0.108543	518	0.132403	563	0.186341
294	0.044708	339	0.054612	384	0.072993	429	0.088445	474	0.114331	519	0.132817	564	0.157992
295	0.048083	340	0.060656	385	0.075044	430	0.087924	475	0.110337	520	0.135418	565	0.184645
296	0.047425	341	0.061821	386	0.079431	431	0.088068	476	0.114031	521	0.129472	566	0.148343
297	0.048282	342	0.059336	387	0.073106	432	0.094912	477	0.105025	522	0.13211	567	0.15582
298	0.049282	343	0.063533	388	0.077485	433	0.089701	478	0.106005	523	0.143203	568	0.151245
299	0.048296	344	0.059317	389	0.091664	434	0.115192	479	0.113201	524	0.130935	569	0.161374
300	0.044818	345	0.057593	390	0.088751	435	0.093207	480	0.106475	525	0.153173	570	0.14196
301	0.0478	346	0.059498	391	0.08156	436	0.093486	481	0.102629	526	0.137112	571	0.165424
302	0.049634	347	0.059783	392	0.081944	437	0.093403	482	0.109492	527	0.138716	572	0.162954
303	0.056322	348	0.058475	393	0.076386	438	0.095415	483	0.109594	528	0.12828	573	0.166634
304	0.049272	349	0.062551	394	0.082228	439	0.097875	484	0.111137	529	0.135029	574	0.161491
305	0.054972	350	0.067232	395	0.088136	440	0.092873	485	0.114786	530	0.152591	575	0.179317
306	0.049443	351	0.077246	396	0.080555	441	0.092977	486	0.103859	531	0.133718	576	0.162145
307	0.050517	352	0.066065	397	0.08299	442	0.093114	487	0.113884	532	0.14292	577	0.159306
308	0.051683	353	0.06254	398	0.080915	443	0.109228	488	0.107226	533	0.141134	578	0.161687
309	0.047925	354	0.064672	399	0.088033	444	0.10614	489	0.121788	534	0.145994	579	0.165496
310	0.051735	355	0.064289	400	0.0918	445	0.0928	490	0.113008	535	0.136398	580	0.161347
311	0.051695	356	0.064527	401	0.083108	446	0.098166	491	0.113443	536	0.147061	581	0.166

582	0.161345	627	0.186257	672	0.218672	717	0.235257	762	0.255092	807	0.288372	852	0.317155
583	0.174474	628	0.193067	673	0.21702	718	0.217696	763	0.266747	808	0.281314	853	0.328246
584	0.165071	629	0.201287	674	0.201107	719	0.234205	764	0.259482	809	0.294285	854	0.306834
585	0.187028	630	0.194045	675	0.225414	720	0.255114	765	0.251962	810	0.29562	855	0.310003
586	0.178351	631	0.205258	676	0.214726	721	0.269155	766	0.248777	811	0.276061	856	0.317775
587	0.166579	632	0.198086	677	0.232716	722	0.261846	767	0.26064	812	0.290221	857	0.331932
588	0.155352	633	0.201657	678	0.211819	723	0.240976	768	0.25673	813	0.288708	858	0.33485
589	0.164763	634	0.185596	679	0.230565	724	0.248126	769	0.274694	814	0.278445	859	0.323625
590	0.161498	635	0.194841	680	0.23218	725	0.265099	770	0.262154	815	0.302704	860	0.304236
591	0.171273	636	0.197748	681	0.22301	726	0.251145	771	0.295962	816	0.303443	861	0.318075
592	0.167646	637	0.197296	682	0.220714	727	0.268479	772	0.264457	817	0.305191	862	0.323137
593	0.197598	638	0.187246	683	0.244366	728	0.252691	773	0.306432	818	0.279803	863	0.314479
594	0.179183	639	0.201624	684	0.215054	729	0.270867	774	0.286858	819	0.28755	864	0.327466
595	0.190416	640	0.204352	685	0.248148	730	0.271965	775	0.275711	820	0.301027	865	0.324345
596	0.16799	641	0.206788	686	0.213634	731	0.254589	776	0.261383	821	0.306746	866	0.309079
597	0.188282	642	0.19421	687	0.22711	732	0.250559	777	0.265574	822	0.309499	867	0.360627
598	0.167152	643	0.211009	688	0.226755	733	0.252508	778	0.255646	823	0.301915	868	0.326798
599	0.175589	644	0.188577	689	0.231161	734	0.271048	779	0.267468	824	0.334625	869	0.352602
600	0.177326	645	0.207153	690	0.221505	735	0.257712	780	0.286722	825	0.333384	870	0.324759
601	0.178773	646	0.187694	691	0.224516	736	0.257498	781	0.277184	826	0.293033	871	0.332043
602	0.174834	647	0.202233	692	0.227529	737	0.273615	782	0.268038	827	0.304607	872	0.332075
603	0.187072	648	0.196361	693	0.246255	738	0.257489	783	0.303698	828	0.320828	873	0.331934
604	0.170149	649	0.209688	694	0.244496	739	0.251651	784	0.27394	829	0.327283	874	0.304635
605	0.191974	650	0.191485	695	0.250453	740	0.240921	785	0.279105	830	0.309002	875	0.351154
606	0.17183	651	0.215706	696	0.223324	741	0.239896	786	0.261899	831	0.315958	876	0.331421
607	0.18431	652	0.20814	697	0.24241	742	0.247108	787	0.270544	832	0.311853	877	0.341355
608	0.176135	653	0.236528	698	0.230778	743	0.243381	788	0.257377	833	0.317813	878	0.325094
609	0.186298	654	0.204172	699	0.24992	744	0.244778	789	0.28838	834	0.314081	879	0.339205
610	0.175821	655	0.209202	700	0.220481	745	0.244624	790	0.283574	835	0.311343	880	0.341542
611	0.185358	656	0.201367	701	0.228904	746	0.243572	791	0.296257	836	0.28961	881	0.324568
612	0.177347	657	0.209397	702	0.231239	747	0.244407	792	0.294281	837	0.316606	882	0.325348
613	0.188101	658	0.20638	703	0.235978	748	0.259159	793	0.28365	838	0.304208	883	0.355367
614	0.184572	659	0.228761	704	0.25286	749	0.248001	794	0.273549	839	0.316001	884	0.337408
615	0.189388	660	0.206313	705	0.234606	750	0.244801	795	0.281258	840	0.305505	885	0.361923
616	0.177405	661	0.221062	706	0.232435	751	0.248879	796	0.274867	841	0.327832	886	0.324533
617	0.188094	662	0.19827	707	0.23519	752	0.243777	797	0.29753	842	0.298991	887	0.359702
618	0.180264	663	0.21883	708	0.238918	753	0.24686	798	0.266711	843	0.334941	888	0.338073
619	0.192506	664	0.197309	709	0.244636	754	0.258407	799	0.276998	844	0.329612	889	0.343636
620	0.167786	665	0.207614	710	0.241906	755	0.256352	800	0.265986	845	0.315514	890	0.34831
621	0.187715	666	0.197541	711	0.231378	756	0.251901	801	0.308268	846	0.310561	891	0.357867
622	0.19682	667	0.228569	712	0.256154	757	0.281471	802	0.304168	847	0.325577	892	0.35062
623	0.189798	668	0.207606	713	0.247469	758	0.257179	803	0.286597	848	0.332077	893	0.339457
624	0.186186	669	0.228694	714	0.232139	759	0.269169	804	0.278188	849	0.315672	894	0.353283
625	0.201839	670	0.213445	715	0.252452	760	0.264352	805	0.286683	850	0.304936	895	0.357899
626	0.180468	671	0.219029	716	0.240089	761	0.259024	806	0.294848	851	0.307669	896	0.34884

897	0.360062	942	0.41415	987	0.419604	1032	0.464236	1077	0.492058	1122	0.558999	1167	0.64583
898	0.362986	943	0.390117	988	0.4338	1033	0.487013	1078	0.523239	1123	0.549733	1168	0.596028
899	0.379401	944	0.381421	989	0.423264	1034	0.450454	1079	0.557145	1124	0.551861	1169	0.651869
900	0.346336	945	0.401994	990	0.427487	1035	0.476684	1080	0.490055	1125	0.624044	1170	0.601716
901	0.370431	946	0.393508	991	0.460581	1036	0.477845	1081	0.506149	1126	0.555908	1171	0.643361
902	0.340784	947	0.392286	992	0.423262	1037	0.476292	1082	0.535461	1127	0.603847	1172	0.602442
903	0.369701	948	0.390811	993	0.430413	1038	0.477745	1083	0.566696	1128	0.539448	1173	0.651336
904	0.343675	949	0.409922	994	0.423156	1039	0.474345	1084	0.523009	1129	0.568302	1174	0.638964
905	0.355882	950	0.372703	995	0.456181	1040	0.484335	1085	0.567907	1130	0.574964	1175	0.633991
906	0.357159	951	0.391527	996	0.416093	1041	0.513009	1086	0.528579	1131	0.603133	1176	0.62876
907	0.358834	952	0.386195	997	0.433886	1042	0.484381	1087	0.539864	1132	0.556422	1177	0.624933
908	0.36954	953	0.393315	998	0.414201	1043	0.502175	1088	0.530502	1133	0.607293	1178	0.663614
909	0.381121	954	0.407103	999	0.431867	1044	0.467221	1089	0.544501	1134	0.590113	1179	0.608132
910	0.415321	955	0.436931	1000	0.442123	1045	0.527211	1090	0.524515	1135	0.602665	1180	0.628576
911	0.367779	956	0.414232	1001	0.435047	1046	0.492604	1091	0.5213	1136	0.56345	1181	0.657995
912	0.365113	957	0.415318	1002	0.422984	1047	0.481814	1092	0.502527	1137	0.600545	1182	0.642587
913	0.381422	958	0.391462	1003	0.426554	1048	0.483718	1093	0.523814	1138	0.587288	1183	0.651894
914	0.40135	959	0.456735	1004	0.44694	1049	0.450871	1094	0.510663	1139	0.611956	1184	0.615492
915	0.384062	960	0.433329	1005	0.457836	1050	0.472634	1095	0.510931	1140	0.574912	1185	0.673812
916	0.377863	961	0.446164	1006	0.43781	1051	0.499704	1096	0.510919	1141	0.599652	1186	0.630973
917	0.383667	962	0.42811	1007	0.431464	1052	0.496345	1097	0.569997	1142	0.588129	1187	0.636008
918	0.361425	963	0.427955	1008	0.44134	1053	0.471872	1098	0.534403	1143	0.585943	1188	0.64047
919	0.364759	964	0.394779	1009	0.431248	1054	0.48149	1099	0.546993	1144	0.605489	1189	0.622282
920	0.379819	965	0.392387	1010	0.447358	1055	0.499929	1100	0.536117	1145	0.599473	1190	0.635155
921	0.364265	966	0.409627	1011	0.462517	1056	0.496603	1101	0.542768	1146	0.561352	1191	0.670917
922	0.353814	967	0.414608	1012	0.443102	1057	0.482877	1102	0.536209	1147	0.633862	1192	0.649968
923	0.389389	968	0.420393	1013	0.481669	1058	0.481561	1103	0.565086	1148	0.625836	1193	0.690785
924	0.361234	969	0.398131	1014	0.438604	1059	0.488763	1104	0.553332	1149	0.605058	1194	0.617995
925	0.392107	970	0.40395	1015	0.452857	1060	0.492083	1105	0.566334	1150	0.61209	1195	0.676048
926	0.389233	971	0.41494	1016	0.474521	1061	0.506449	1106	0.536987	1151	0.635355	1196	0.689791
927	0.390528	972	0.398969	1017	0.451666	1062	0.477972	1107	0.564014	1152	0.585995	1197	0.660198
928	0.39482	973	0.427485	1018	0.475584	1063	0.523517	1108	0.546529	1153	0.642283	1198	0.63955
929	0.377122	974	0.405219	1019	0.468017	1064	0.535205	1109	0.551934	1154	0.603189	1199	0.641637
930	0.377617	975	0.419734	1020	0.445078	1065	0.539422	1110	0.563301	1155	0.618173	1200	0.681264
931	0.376539	976	0.407899	1021	0.504952	1066	0.562552	1111	0.549638	1156	0.605939	1201	0.671101
932	0.374788	977	0.411717	1022	0.483345	1067	0.544301	1112	0.538226	1157	0.625554	1202	0.661569
933	0.404085	978	0.413059	1023	0.489951	1068	0.532123	1113	0.539965	1158	0.597489	1203	0.617175
934	0.391533	979	0.415279	1024	0.439338	1069	0.538287	1114	0.580942	1159	0.613601	1204	0.692898
935	0.403797	980	0.421149	1025	0.477422	1070	0.497722	1115	0.60314	1160	0.609134	1205	0.690513
936	0.384653	981	0.426657	1026	0.449716	1071	0.525461	1116	0.565242	1161	0.640295	1206	0.662912
937	0.386721	982	0.457268	1027	0.498129	1072	0.487283	1117	0.570506	1162	0.617504	1207	0.68698
938	0.379814	983	0.430376	1028	0.481738	1073	0.547193	1118	0.56344	1163	0.664082	1208	0.721556
939	0.362884	984	0.426188	1029	0.476211	1074	0.521553	1119	0.564057	1164	0.597936	1209	0.642229
940	0.400535	985	0.462795	1030	0.49217	1075	0.48923	1120	0.542633	1165	0.629889	1210	0.662823
941	0.386959	986	0.427736	1031	0.513642	1076	0.530893	1121	0.576221	1166	0.646018	1211	0.676347

1212	0.632119	1257	0.752196	1302	0.753968	1347	0.761893	1392	0.88604	1437	1.1038	1482	0.987136
1213	0.679277	1258	0.715731	1303	0.752046	1348	0.845626	1393	0.85359	1438	1.12553	1483	1.00371
1214	0.710995	1259	0.700434	1304	0.778566	1349	0.891618	1394	0.921305	1439	1.0359	1484	0.994738
1215	0.7011	1260	0.735467	1305	0.752036	1350	0.871765	1395	0.822806	1440	0.968935	1485	0.972463
1216	0.74941	1261	0.6928	1306	0.772883	1351	0.859093	1396	0.937843	1441	1.06757	1486	1.04985
1217	0.673881	1262	0.71226	1307	0.756829	1352	0.842406	1397	0.895269	1442	1.00989	1487	0.975975
1218	0.687875	1263	0.710674	1308	0.814632	1353	0.889453	1398	0.917569	1443	1.05191	1488	0.950624
1219	0.697476	1264	0.74798	1309	0.766131	1354	0.803539	1399	0.906193	1444	1.07572	1489	1.05449
1220	0.680994	1265	0.767375	1310	0.785938	1355	0.869604	1400	0.977327	1445	0.954288	1490	0.982719
1221	0.690477	1266	0.736308	1311	0.775082	1356	0.816022	1401	0.938095	1446	1.06583	1491	1.05873
1222	0.682618	1267	0.737476	1312	0.799425	1357	0.86363	1402	0.880528	1447	1.05052	1492	1.03139
1223	0.656983	1268	0.746199	1313	0.837269	1358	0.828269	1403	0.912267	1448	1.08645	1493	1.05686
1224	0.636318	1269	0.728779	1314	0.881387	1359	0.819789	1404	1.03129	1449	1.12161	1494	0.992435
1225	0.706844	1270	0.753369	1315	0.867479	1360	0.806533	1405	1.00796	1450	1.00126	1495	1.01696
1226	0.706044	1271	0.758577	1316	0.771715	1361	0.831443	1406	0.918895	1451	1.09815	1496	1.02048
1227	0.689002	1272	0.764228	1317	0.827299	1362	0.877287	1407	0.921608	1452	0.956531	1497	1.03836
1228	0.668913	1273	0.750502	1318	0.882343	1363	0.806386	1408	0.960599	1453	0.991896	1498	1.0578
1229	0.698747	1274	0.777154	1319	0.808738	1364	0.934094	1409	0.961444	1454	0.946686	1499	1.04565
1230	0.711573	1275	0.77739	1320	0.754571	1365	0.911694	1410	1.06308	1455	0.997011	1500	1.03338
1231	0.692171	1276	0.758435	1321	0.795053	1366	0.867462	1411	0.973844	1456	1.00165		
1232	0.707707	1277	0.702703	1322	0.82708	1367	0.90816	1412	0.934247	1457	1.00468		
1233	0.682715	1278	0.758446	1323	0.769722	1368	0.926297	1413	0.98725	1458	1.05487		
1234	0.714083	1279	0.74123	1324	0.766276	1369	0.927024	1414	1.01187	1459	1.03663		
1235	0.690423	1280	0.725355	1325	0.804285	1370	0.877676	1415	1.08755	1460	1.05002		
1236	0.707547	1281	0.728237	1326	0.796711	1371	0.903891	1416	1.09311	1461	1.0333		
1237	0.733954	1282	0.728645	1327	0.832921	1372	0.902779	1417	0.965283	1462	1.03239		
1238	0.679326	1283	0.724802	1328	0.762819	1373	0.871595	1418	1.00686	1463	1.04628		
1239	0.684146	1284	0.716119	1329	0.827643	1374	0.856552	1419	1.09557	1464	0.978658		
1240	0.716155	1285	0.78231	1330	0.927617	1375	0.957993	1420	1.05701	1465	1.08857		
1241	0.685704	1286	0.796552	1331	0.897897	1376	0.913503	1421	1.09592	1466	0.970823		
1242	0.711949	1287	0.74148	1332	0.880964	1377	0.937382	1422	0.966371	1467	1.08458		
1243	0.678536	1288	0.782778	1333	0.855272	1378	0.870836	1423	0.979611	1468	1.04391		
1244	0.707355	1289	0.778979	1334	0.827435	1379	0.86676	1424	0.984225	1469	1.04956		
1245	0.71058	1290	0.751942	1335	0.856955	1380	0.861401	1425	0.996303	1470	1.02765		
1246	0.769408	1291	0.73326	1336	0.863523	1381	0.923882	1426	0.937905	1471	1.08735		
1247	0.735449	1292	0.765038	1337	0.881487	1382	0.885525	1427	0.970937	1472	1.02442		
1248	0.722797	1293	0.805359	1338	0.798427	1383	0.871586	1428	0.940356	1473	1.04708		
1249	0.777616	1294	0.750754	1339	0.85919	1384	0.917915	1429	1.0622	1474	1.03854		
1250	0.688016	1295	0.79362	1340	0.829825	1385	0.972266	1430	1.06493	1475	0.989258		
1251	0.722657	1296	0.790245	1341	0.830585	1386	0.876053	1431	1.11945	1476	0.966172		
1252	0.698606	1297	0.812198	1342	0.847976	1387	0.850604	1432	1.0243	1477	1.02741		
1253	0.744162	1298	0.798804	1343	0.802474	1388	0.88869	1433	1.05049	1478	0.958115		
1254	0.699926	1299	0.780832	1344	0.821603	1389	0.832297	1434	1.02306	1479	1.02418		
1255	0.730369	1300	0.740206	1345	0.804194	1390	0.910171	1435	1.09861	1480	0.958283		
1256	0.702729	1301	0.788793	1346	0.796089	1391	0.905872	1436	1.00632	1481	0.951073		