Problem Chosen	2021	Team Control Number
D	MCM/ICM Summary Sheet	2105529

Figuring Out the Best Packaging Technique in Data Broadcasting

Summary

In complicated topologies, the choice of broadcasting patterns dominates the effectiveness of data transmission. The best strategy involves a combination of singlecasting and multicasting, which demands sophisticated packaging techniques. Therefore, it is vital to find the optimal algorithm for packaging branch information. In this paper, We make a very meaningful attempt to calculate the efficiency of different packaging strategies and compare them.

There exists certain conditions where the optimal algorithm for packaging branch information can be discovered through pure reasoning. However, the best strategy in a certain topology system is not always apparent. Therefore, we decide to develop reasonable models to **quantify the efficiency** of different patterns. After that, we use enumeration to discover the optimum one.

Primarily, we study certain topologies with certain pack capacities. We find that in cases like Task 1, the amount of receiving points and the minimum number of redundancy could be decided by **reasoning**.

Then we study certain topologies with uncertain pack capacities. We find that, in order to reach a comprehensive optimum, we need to reach a **balance** between **the redundancy on edges** and **the loss of capacity caused by headers**. Therefore, we develop a model to **quantify** those losses and then compare them. Afterward, we **optimize the previous model** to fit in more complicated situations. We adopt both models in Task 2 and find the best strategy.

Finally, we manage to find the optimal algorithm for packaging branch information. We achieve this by several steps. First, we simplify the topology by **aborting unconnected and unnecessary branches**. Second, we further **simplify the topology network to an array** and **describe it in two dimensions**. Third, we **modify our previous model** and **use the two dimensions to represent the function value**. Last of all, we use **heat map to enumerate different patterns and compare them in a visualized way**.

Now we are able to find the best strategy of packaging branch information. In the end, we discuss the advantage of our algorithm and draw a conclusion.

Keywords: quantify, simplify to an array, heat map, enumerate, visualize

Figuring Out the Best Packaging Technique in Data Broadcasting

February 3, 2021

Contents

1 Introduction

1.1 Problem Background

Advancing towards a booming flow of information currents, the world is requiring more and more efficient broadcasting techniques to keep up the pace. Meanwhile, the number of Internet users is still under a sharp grow, making the topology blueprint even more sophisticated. As a result, there has been an urge to found the optimal pattern of information transmission.

Previous broadcasting involved the technique of unicast (or singlecast), by which only one pack of information is sent to one specific user at one time. Although its specific destination minimizes the capacity occupied by headers, this method would lead to great redundancy if the same pack needs to be sent to multiple users along the same path. We must appeal to a technique with greater efficiency in face with the rising transmission demand. Therefore, we need to find a balance between the capacity loss caused by headers and the redundancy of packs.

1.2 Our work

This paper aims to find a relatively optimal structure of a transmitting system. By investigating the combination of singlecasting and multicasting, we gradually gain insights into and are able to develop an advanced

2 Assumptions and Definitions

2.1 Our Assumptions

We define **the length of the minimum path** from node U to node V as the **distance** between them. Then we make the following assumptions:

(1) The topology is relatively balanced. Namely, the majority of nodes shares the same length of minimum path from the source and has an approximately equal amount of neighbors in the original topology.

(2) The complexity of the topology is directly proportional to its scale. The longer the minimum length from the source, the larger the amount of nodes.

(3) The topology is undirected. As long as data can be transferred from router A to router B, it can also be transferred backwards.

(4) The topology is connected. We ditch those nodes that are disconnected to the source.

(5) The loss caused by redundancy grows linearly rather than exponentially. It is directly proportional to the amount of loads on an edge.

(6) The effectiveness in the network is only related to the length of path a data packet covered and has nothing to do with the number of packets or the computing power of routers where packets are transferred.

(7) The transmitting network uses IP addresses to recognize different routers. We assume the global percentage of IPv6 is approximate to the percentage of IPv6 addresses visiting Google according to the statistical data given by Google, i.e. 32

(8) The weights of all edges are equivalent. Namely, the consumption is always the same when a same amount of data travels through a single edge.

2.2 Variable Definitions

Symbol	Meaning
f_1	a function describing redundancy
f_2	a better function
f_s	a simpler function equivalent to f_2, only has the difference of constant
V_i	a node in the simplified network which has distance i from the source
N_i	the number of V_i in the simplified network
T_i	a tree structure in the simplified network which use V_i as root
S_i	the number of nodes in a single T_i
U	the number of all users in the simplified network
I_A	the amount of information contained in the whole pack
I_H	the amount of information contained in the head of the pack
I_G	the amount of valid information contained in the pack
n	the amount of T_i contained in the header
i	the rank of T_i contained in the header
N	the distance from the source to the furthest targeted point
R	pure road redundancy

3 Topologies with Certain Packet Capacity – Task 1

3.1 Calculating the maximum number of receiving points

We study possible paths originating from node A. Since duplicate payloads are not allowed to appear, the data can only access node B C D once, respectively. Then we reach the next level of the question.

Given that the maximum number of branches on the header is 3, we know that the pack can get through at most 2 edges after reaching the second point. That means at most two other points can be reached.

However, there is only one edge (D-H) after node D, indicating that only one other point is available. As to node B and node C, the upper limit can both be reached.

Therefore, the number of receiving points cannot exceed 9. In detail, 3 for B and two other points, 3 for C and two other points, 2 for D and one other point and 1 for the source point A.

We can easily find a none-redundant distribution with 9 receiving points. The example is as follows:

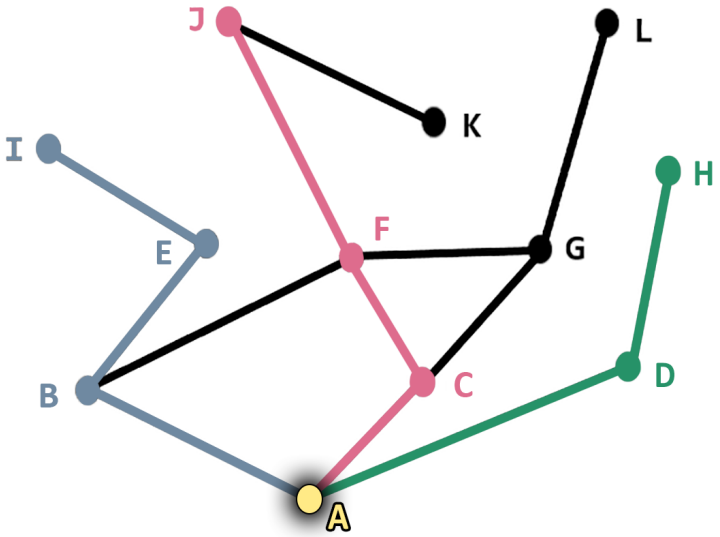

Figure 1: e.g.

(1). A-B-E-F

(2). A-C-F-J

(3). A-D-H

3.2 Calculating the minimum number of redundancy

Primarily, we discover that paths **A-E-H-O** and **A-D-J-M** are mandatory if we want all nodes to be reached. The reason is that, these two paths are the only approaches to get to node O and M – All the other available paths cover more than 3 edges, which would violate the pack capacity.

Therefore, another two headers **AE-EI-EL** and header **AC-CG-CK** can be determined. Since nodes I and L can only be reached through edge A-E if we don't want to violate the maximum capacity, we adopt the former multicast header to keep the number of redundancy as small as possible. The same is with the latter.

Then we find that we need at least two single headers to get to node P, Q, N. The 3 nodes are all 2 edges away from the source, so multitasking can only reach 2 of 3. Apparently, the remaining node requires another singlecast to be covered. These, along with the mandatory path A-D-J-M demonstrated above, suggest that the minimum number of redundancy must be larger or equal to 3 in that edge A-D has already been traveled for 3 times.

We find such a pattern shown in the figure below

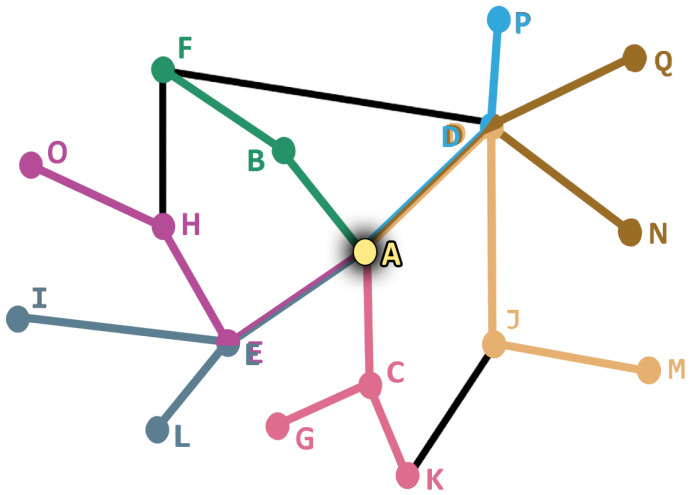

Figure 2: e.g.

Thus, the minimum number of redundancy in this case is 3.

4 Topologies with Uncertain Packet Capacity – Task 2

We try to figure out what makes a system relatively optimal. To do so, we investigate where and how superfluous information is generated. We discover two main sources of such information:

One is redundancy on the edges. Packets with the same payload may pass through the same edge multiple times, although in different headers. This leads to a loss of efficiency.

The other is the headers. They are also invalid information that reduce the actual capacity of the pack.

Therefore, to reach the optimum, we should find a balance between the two.

4.1 Setting up an assessing model – establishing f_1

We define the following variables.

Symbol	Definition
I_H	The amount of header information in a certain pack
I_A	The amount of all information in a certain pack
I_G	The amount of valid information in a certain pack

Note 1: I_A I_G I_H are all measured by how many users' information the whole pack / valid part / head part can hold, and differs from pack to pack.

Note 2: The amount of equivalent redundancy include any node that is covered in a transmission path. In this specific task this doesn't matter, but we may need the declaration later on.

We establish the following formula:

$$f_1 = R + \frac{E * I_H}{I_G}$$

where f_1 stands for the average number of edges that a receiving point utilizes to transmit one unit of valid information.

It can represent the efficiency of a system to a fair extent.

Based on the model, we are now able to calculate the degree of efficiency of a certain transmitting pattern. The minimum value of f_1 represents the most advanced pattern.

4.2 Optimization of the present model – establishing f_2

However, we find that this model may not work well in certain complicated situations. On one hand, the model does not take into account whether the load of an edge changes from zero to one. But in real situations this change will also add to the consumption of the system. On the other hand, the model may suggest the pack go through unnecessarily long paths to avoid redundancy on a single edge. We have raised an example in the following topology to illustrate this.

THIS IS AN IMAGE (CUE QYH)

Consequently, we create another model to handle the dilemma.

We define the following variables.

Symbol	Definition
E	The amount of utilized edges
U	The amount of receiving nodes
I_A	The scale of information per pack
I_G	The scale of valid information per pack
I_H	The scale of header information per pack

Note 1: I_A I_G I_H are all measured by how many users' information the whole pack / valid part / head part can hold.

Note 2: The amount of receiving nodes include any node that is covered in a transmission path. In this specific task this doesn't matter, but we may need the declaration later on.

We establish the following formula:

$$f_2 = \frac{I_A * E}{\sum I_G * I_H} * \frac{E}{U}$$

where f_2 stands for the average number of edges that a receiving point utilizes to transmit one unit of valid information.

4.3 Implementation of the models – Path Enumeration

No matter in f_1 or f_2, we can all easily find that a redundancy cost a lot. Because of this, we try to find a solution of no redundancy. Since there are only four points connected to A, at most four packs can be sent. When there are no pack contain more than five users' information, we can easily prove that at least one user would be travelled twice because of B and D. We find a 5-5-4-4 method. Or if we allow a package contains a header of 6 users' information, we can have a method of 6-4-4-3.

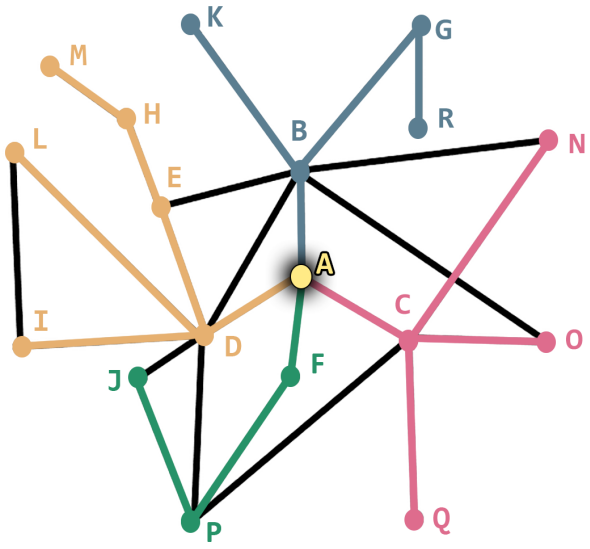

Figure 3: 6443

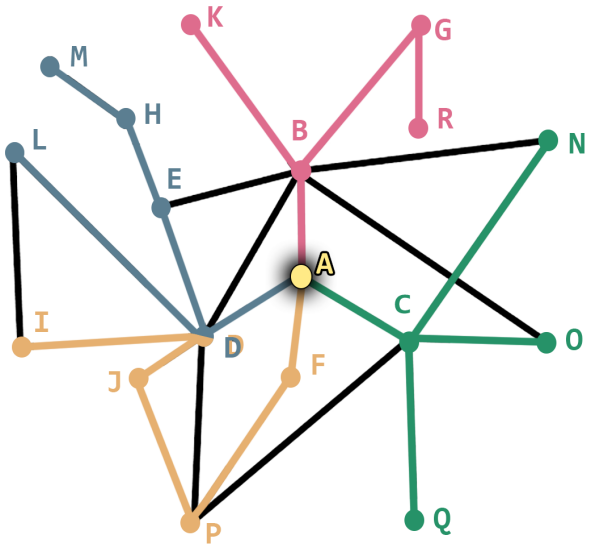

Figure 4: 5544

After calculation, we find that the 6-4-4-3 strategy is better.

5 Generalization to Topologies with Uncertain Packet Capacity – Task 3

5.1 Reintroduction of our assessing models and their equalization

In this part, we will reintroduce our problem. Given a network in reality, the only sender, and the many receivers, we want to find a good packaging strategy that can minimize information lost and redundancy. To simplify the network, we will cut unnecessary edges and only focus on the necessary edges. Method is as follows. For each receiver, we find its shortest path to the sender, and whenever it has multiple choices, we always choose the way that are less visited. Using this method, we got a nearly balanced network. That is to say, for each node having distance i from the sender (called V_i), they have the same status, and the simplified graph is a symmetrical tree where every node in it is receiver, and the root of the tree is the sender. Also, every node V_i is a root to its child tree

(called T_i), because of the symmetry, each V_i and T_i are the same. We call the number of V_i as N_i, and the number of nodes in T_i as S_i. N_i and S_i are sequence that are only related to i. Although the assumption seems rough, because of the law of big number, the model works surprisingly well. Also, since all the nodes become receivers, we can only focus on the point on the peek of the sequence N_i, since if the peek of N_i is covered, those nearer nodes are definitely covered, and for the further nodes, they are absorbed by the peek of N_i in the formula, and they also do not have a high proportion. We use i and n to describe a packaging strategy, meaning a pack has the information of nodes in n T_i. Using this model, we can give a dedicated calculation.

5.2 Function calculation

$$
\begin{aligned}
f_s(i,m) &= \frac{E}{I_A - I_H} = \frac{N_i}{m} \cdot \frac{I_H}{I_A - I_H} \quad (\text{ since } W = \frac{N_i}{m} I_H) \\
&= \frac{N_i S_i}{I_A + i - 1 - \left(I_H + \frac{I_A(i-1)}{I_H}\right)} \quad (\text{ since } I_H = mS_i + i - 1) \\
&\geqslant \frac{N_i S_i}{I_A - 2\sqrt{I_A(i-1)} + i - 1} \qquad (1) \\
&= \frac{N_i + \cdots + N_n}{\left(\sqrt{I_A} - \sqrt{i-1}\right)^2},
\end{aligned}
$$

where the equality in inequality (1) holds if and only if $I_H = \sqrt{I_A(i-1)}$, namely, $m = \frac{\sqrt{I_A(i-1)} - (i-1)}{S_i}$.

$$
\begin{aligned}
f_s(i+1) - f_s(i) =& -\frac{N_i}{\left(\sqrt{I_A} - \sqrt{i}\right)^2} + (N_i + \cdots + N_n)\left(\frac{1}{\left(\sqrt{I_A} - \sqrt{i}\right)^2} - \frac{1}{\left(\sqrt{I_A} - \sqrt{i-1}\right)^2}\right) \\
\approx& \frac{1}{I_A}\left(-N_i\sqrt{i} + \frac{U}{\sqrt{I_A}}\right)
\end{aligned}
$$

In the step of finding the optimal n and i, we also draw a graph.

The data we use comes from real data.

Also, we write a program that can transform network to arrays by breadth first search(BFS), try different i and n, and draw the value of functions under different conditions. We can find that the result of the program is exactly the same with theoretical calculation. And the heat map conveys a good insight that what will occurs when i and n changes.

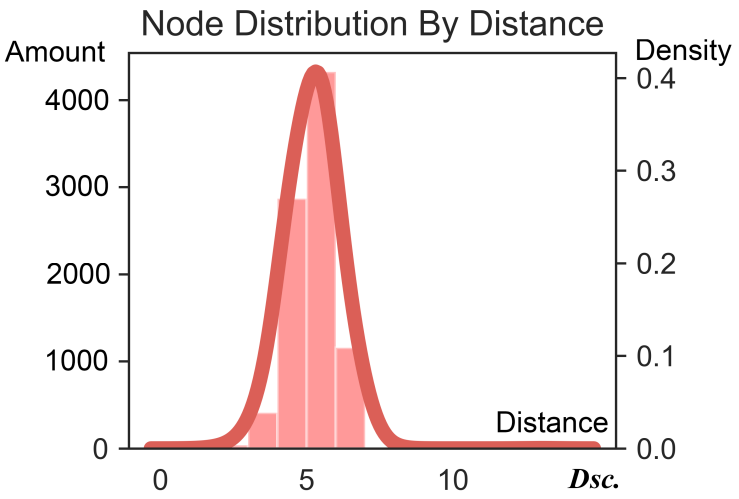

Figure 5: distance distribution of nodes

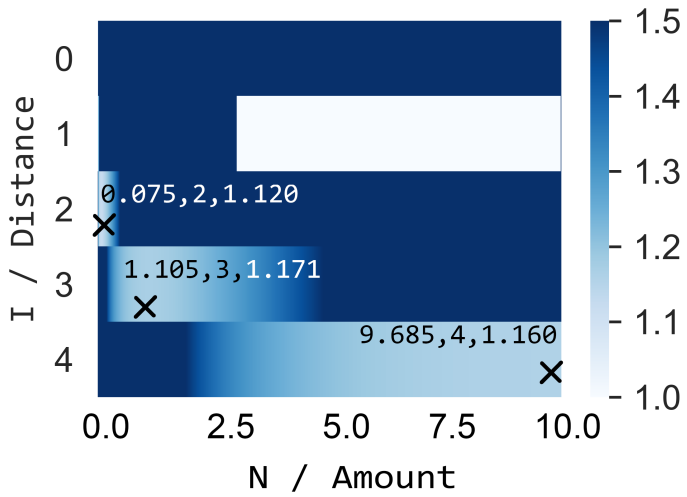

Figure 6: hot map showing the value of $f_s(i, n)$ and its minimum value

Using this code and the Data in Text2 as an example, we calculated that it is the best when $i = 1$ and $n = 1.1$, or $i = 2$ and $n = 2.5$, and on both circumstances $f_1 = 1.27$, having 98% of accuracy from the true value $f_1 = 1.29$! Also, using i and n as guides, we can easily know how much information the header should contain. As an example, in this case, $i = 1$ and $n = 1.1$ means that each pack should solve one of its nearest road, which is exactly the case. Also in classic case like star graph, complete graph and linear graph, this algorithm also find the best solution.

Table 1: e.g

b \ c / a	Symbol	Definition
1	2	3
1	2	
1		3
1	2	3
		3

6 Conclusion

In this paper, we investigate the strategy of multicasting. First we use pure reasoning to solve Task1 which has fixed solution. Then we solved Task2 by both reasoning and function definition. We define a loss function called f_1 to show how much information are lost, and define another function called f_2 to show how much edge a user uses on average. The hardest part is Task3, where we build a model to solve the problem. In this model, we cut edges and simplified the network, compress the network to an array, setting different i and n, using i and n to portray the evaluate function f_s, use math skills to find the best solution of i and n, find an extremely simple yet highly accurate way for human to decide how many information a header should contain, and program to visualize the result. Finally, we verified our result on simple graphs.

References

[1] Thomas H. Cormen et al. *Introduction to algorithms*. 2nd ed. The MIT Press, 2001.

[2] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. `http://snap.stanford.edu/data`. June 2014.

Appendices

Appendix A The Code of Heat Map

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import networkx as nx
import numpy as np
import seaborn as sns

source = 0
eps = 0.01
SUP = 10.0
infoall = 327
Sh = 1.0

f = open("./In.txt")
s = f.readline()
s = f.readline()
s = f.readline()
p1 = s.find(':')+2
p = s.find('Edges')
node = int(s[p1:p])
print(node)
p = p+7
edge = int(s[p:])
print(edge)
s = f.readline()


G = nx.Graph()
G.add_nodes_from(range(0,node+1))
for i in range(0,edge):
    s = f.readline()
    p = s.find('\t')
    x = int(s[:p])
    y = int(s[(p+1):])
    G.add_edge(x,y)

"""
H = nx.Graph()
H.add_nodes_from(range(0,node+1))
for i in range(0,node):
    if not nx.has_path(G,0,i): continue
    path = nx.shortest_path(G,0,i)
    j = path[0]
    for j in range(1,len(path)):
        H.add_edge(path[j-1],path[j])
"""
H = G

Hqu = [source]
Hds = [13]*(node+1)
Hds[source] = 0.5
Hmk = [0]*(node+1)
max = 0
Nds = [1]
Nsn = [float(len(H[source]))]      #the average number of its sons
```

```
head = 0
tail = 0

while head<=tail:
    ngh = H.neighbors(Hqu[head])
    for i in ngh:
        if (i>node): break
        if (Hmk[i]==0):
            tail+=1
            Hqu = Hqu + [i]
            Hds[i] = Hds[Hqu[head]]+1
            if Hds[i]-0.5>max:
                for j in range(max,int(Hds[i]-0.5)):
                    Nds = Nds + [0]
                    Nsn = Nsn + [0.0]
                max = int(Hds[i]-0.5)
                Nds[max] = 1
                Nsn[max] += len(H[i])
            else:
                Nds[int(Hds[i]-0.5)]+=1
                Nsn[int(Hds[i]-0.5)]+=len(H[i])
            Hmk[i] = 1
    head+=1

maxs = 0.0
for i in range(0,max+1):
    Nsn[i] = Nsn[i]/float(Nds[i])
    if maxs < Nsn[i]: maxs = Nsn[i]

def KSH(x):
    sns.set_style('white')

    fig, ax = plt.subplots(figsize=(3,2.5))
    # seaborndistplot
    sns.set_palette("hls")  # hls
    sns.distplot(x, color="r", bins=np.arange(0,max+1,1) ,kde=False)
    ax.set_title('Node Distribution By Distance')

    ax2 = ax.twinx()
    sns.kdeplot(x, bw=.75, linewidth = 5, alpha = 0.5)

    plt.savefig("hist.png", dpi=800)
    plt.show()

KSH(Hds)

print(Nds)

maxn = 0
for i in range(0,max):
    if Nds[maxn]<Nds[i]:  maxn=i;
"""
tmp = Hds
q=max

for i in range(max,maxn,-1):
    print(Nds[i-1]," ",Nds[i],end=" ")
    if (Nds[i]<(Nds[i-1])):
        for j in range(0,node+1):
            if abs(tmp[j]-i-0.5)<1e-6:
                tmp[j] = i-1
                Nds[i-1]+=1
        Nds[i] = 0
        if (q==max): q = i-1
        print("1",end="")
    print()

print(Nds)
KSH(tmp)

q=max
for i in range(max,maxn,-1):
    if Nds[i]/float(Nds[i-1])<eps:
        Nds[i-1] = Nds[i-1]+Nds[i]
        Nds[i] = 0
        if (q==max): q = i-1

rate = [0.0]*(max+1)
for i in range(q,0,-1):
    if Nds[i-1]>0: rate[i] = Nds[i]/float(Nds[i-1])
for i in range(1,q+1,1):
    print('%.1f'%rate[i],end=" ")
```

```
"""

"""
def vsimple(x,i,n,C):
    b = x[:]
    for ii in range(len(x)-1):
        b[ii] = b[ii+1]/b[ii]
    b = b[:-1]
    def D(b_list,i_in_b):
        mul=1
        sum=0
        for var in b_list[i_in_b:]:
            mul *= var
            sum += mul
        return sum
    return x[i]*(i+n*D(b,i))/(n*(C-n*D(b,i)))
"""

def vsimple(x,i,n,C):
    b = x[:]
    for ii in range(len(x)-1):
        b[ii] = b[ii+1]/b[ii]
    b = b[:-1]

    mul=1
    sum=1
    for var in b[i:]:
        mul *= var
        sum += mul
    packcost=i-1+n*sum
    return x[i]*(packcost)/(n*(C-packcost))

def v1(x,i,n,C):
    sm=-1
    for var in x:
        sm+=var
    return vsimple(x,i,n,C)*C/sm


maxs=10.0

x = np.arange(0,maxs,eps,float)
for j in range(0,len(x)-1):
    x[j] = (x[j]+x[j+1])/2.0
col = len(x)
col-=1
data = np.empty([maxn,col], dtype = float, order = 'C')
#print(x)

data[0] = [SUP]*col
for i in range(0,maxn):
    for j in range(0,col):
        if x[j]>Nsn[i-1]:
            j = j-1
            continue
        data[i][j] = v1(Nds,i,x[j],infoall)
        if data[i][j]<Sh: data[i][j] = SUP
        if data[i][j]>SUP: data[i][j]= SUP
    for k in range(j+1,col):
        data[i][k] = SUP

for i in range(1,maxn):
    minall = SUP
    q = 0
    #    print(i,': ',end='')
    for j in range(0,col):
 #       print('%.2f'%data[i][j],end=' ')
        if data[i][j]<minall:
            minall = data[i][j]
            q = x[j]
    print(minall,' ',q)
#   print()


fig, ax = plt.subplots(figsize=(3,2))


sns.set_style("white")

sns.heatmap(data, vmax=1.5, vmin=1.0, cmap = "Blues")
ax.set_xlabel('N / Amount',{'family' : 'Consolas'},rotation=0)
ax.set_ylabel('I / Distance',{'family' : 'Consolas'},rotation=90)
```

```
label_y = ax.get_yticklabels()
plt.setp(label_y , rotation = 0)
label_x = ax.get_xticklabels()
plt.setp(label_x , rotation = 60)

'''
from matplotlib.ticker import MultipleLocator, FormatStrFormatter
xmajorLocator   = MultipleLocator(1) #xn
xmajorFormatter = FormatStrFormatter('%1.1f') #x
ax.xaxis.set_major_locator(xmajorLocator)
ax.xaxis.set_major_formatter(xmajorFormatter)
'''

plt.savefig('Heapmap.png', dpi=800)
plt.show()
```