OPTIMIZATION OF LINK SCHEDULING IN DIRECTIONAL WIRELESS NETWORKS USING HEURISTIC METHODS

Indr-568 Project Report

Anique Akhtar

Abstract

Link scheduling plays an important role in the performance and efficiency of a wireless network. This is why some research has been carried out to schedule the communication links in the most effective way. This scheduling problem gets more complicated with the usage of directional antennas. In this paper, we try to understand the problem of directional wireless link scheduling. Once we understand the problem, we propose multiple heuristic methods to solve the problem at hand and discuss how the problem behaves in different scenarios. We slightly amend and use different portions of famous heuristic methods to obtain the best results. The simulation results show a great amount of improvement in the results obtained from our heuristics.

1. Introduction

The recent emergence of applications such as uncompressed high definition TV (HDTV) and ultra-fast file transfer requires very high throughput (VHT) of up to multi-gigabits of speed in wireless local area networks (WLANs). To cater for such VHT requirements, 60 GHz (Millimeter wave) is being used, to achieve a theoretical maximum throughput of up to 7 Gbit/s in WLANs. Using mmWave also means the communication gets a little more complicated. At 60 GHz waves face high signal path loss due to atmospheric attenuation and oxygen absorption. To compensate for such high path losses, directional antennas are used with narrow beamwidths.

Directional antennas provide high speed communication in 60 GHz band. They also enable spatial reusability due to their narrow beamwidth. This means that the interference models made for omni-directional antennas do not fit with the directional antenna WLAN. Interference models greatly change the scheduling mechanism, therefore, there's a need to find scheduling mechanism for directional WLAN.

In every wireless network, there is an Access Point (AP) that is responsible for scheduling the communication of all the Stations (STA) in the network. Each STA that wants to communicate with any other STA in the network has to first inform the AP of the possible communication. The AP then schedules time slots for the communication to happen. Each communication link in the network has some interference to every other ongoing communication link. Depending on the distance between links and the direction of communication, the interference values changes. The interference of each link with every other link is different. The task of the AP is to schedule the links in such a way as to perform the communication in the most efficient way.

To the best of my knowledge, this specific problem has never been solved using heuristics. Previous work has proposed algorithms like 'Generalized Proportional Fair Scheduling' (Ramjee *et al.* 2006) and 'Minimum Length Scheduling' (Pantelidou *et al.* 2009, Sadi *et al.* 2014). Algorithms like these try to tackle the scheduling problem but none of them get results anywhere close to the optimal results. They propose greedy algorithms that give an acceptable result in the fastest possible way. I could not find any prior work that tried to explain or understand how the problem works.

Metaheuristic has been the studied for decades. The earliest and the most popular ones include tabu search (TS) (Glover 1986), simulated annealing (SA) (Kirkpatrick et al. 1983), and genetic algorithms (GAs) (Holland 1975). In tabu search, an initial solution is selected and then all the neighborhood of the current solution is exhaustively searched. A *neighborhood* of a solution is a group of solutions that can be obtained by a set of moves. A *move* is a step that converts one solution into another solution. In each iteration, the best solution is chosen as the current solution in the next iteration. To prevent cycling, TS keeps a small history of recent moves made in a tabu list. These moves are called tabu moves and should not be made. TS and its hybrids have been used to solve problems like timetabling problem (Santos et al. 2004, Rahoual and Saad 2006), vehicle routing (Taillard et al. 1997), and job shop scheduling (Dell'Amico and Trubian 1993, Nowicki and Smutnicki 1996).

Simulated annealing (SA) is a local search technique that is nature inspired. At the start an initial solution is selected then a move is made on the current solution to form a candidate solution. If the cost of the candidate solution is lower than the current solution, then the candidate solution is made the current solution. If the cost of the candidate solution is higher, than a *transition probability function* is used to determine whether the candidate solution is accepted. A random number is generated and if the random number is less than the value of probability function than the candidate solution is accepted regardless of whether it has a lower value or not. Once a candidate solution is rejected, a move is applied to the current solution to make a new solution. This way SA allows for uphill moves with a certain probability that assures the algorithm doesn't get stuck on a local optimum.

Genetic algorithm is a population based heuristic that mimics the process of natural evolution. GA starts with a population of solutions, from these solutions, a group of best solutions is selected to become parents. These parents are then mated to form children or offspring solutions and added to the population. Crossover and mutation moves are usually used to create children both the moves have their own probability. In crossover, portions of both parents are mixed to form children. In mutation, a small perturbation is usually made to a solution. In GA every member of the population is considered a chromosome.

Recently, new metaheuristics have been proposed greedy randomised adaptive search procedure (GRASP) (Feo and Resende 1995, Aiex et al. 2003), adaptive multi start (Boese et al. 1994), adaptive memory programming (AMP) (Taillard et al. 2001), ant system (AS) (Dorigo and Gambardella 1997), and particle swarm optimization (PSO) (Kennedy and Eberhart 1995). These metaheuristics have been employed to problems that are similar to ours but not the same as ours. Simulated annealing with memory and evolution-based diversification (SAMED) (Azizi *et al* 2010) is used on problems like job-shop scheduling.

Directional link scheduling is an NP-hard problem. Since directional link scheduling has not been optimized before, I had to use different metaheuristic to try to understand the problem. After learning how the problem behaves, different components and features of well-known heuristics are mixed to improve on the current solutions and try to find an optimum value.

The paper is organized as follows. Section 2 describes the directional link scheduling problem. In Section 3, the problem is encoded so it can be used by different metaheuristics. Section 4 shows how the genetic algorithm behaves for this certain problem. In section 5 I implement a simulated annealing. In section 6, simulated annealing with a local search is implemented. In section 7, genetic algorithm is implemented to the results obtained from section 6. In section 8, population based simulated annealing with local search is implemented. Section 9 gives my analysis on the problem. Section 10 concludes the paper.

2. Problem Definition

In wireless networks interference plays an important role in the efficiency of the network. The communication schedule is governed by how different links interfere with each other. In directional wireless networks, this interference becomes more complicated. This is because in directional antennas, interference would depend on which direction a certain STA is transmitting at and which direction a certain STA is receiving from. A pair of transmitter and receiver STA forms a link. How much a certain link interferes with another link depends upon the position of both the transmitter as well as the receiver in both the links. The interference is usually caused by the transmitter of one link and the interference is usually received at the receiver of a link. In directional wireless networks, a link may cause a lot of interference to another link while not receiving any interference from that link.

Our goal is to minimize the amount of time it takes for each link to send a specific amount of data. A link might be able to send the data much faster and in a smaller time if there is no interference to it. On the other hand if a link has a lot of interference from other links, it might take a long time to send the same data. If a certain link takes a long time to transmit data, then it would interfere with more links which are sending the data at the same time. The speed at which a certain link would send data is usually defined at the start of the communication and must be kept constant throughout the transmission. Therefore, before sending any data, a link needs to know the interference from all the other links during the transmission.

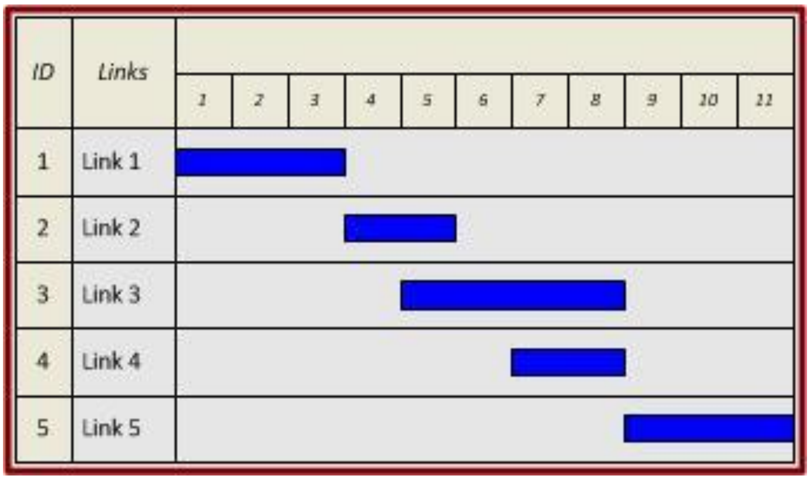

Fig 1. A specific communication schedule.

Fig 1 shows an example of how one scheduling could look like. In this certain example, link 1 and link 5 do not receive any interference and hence can transmit at full speed. Link 3 receives interference from both link 2 and link 4. Depending on the amount of interference, link 3 would choose a certain transmission speed to send data at. If interference is too much for a link, it might not be able to send data at all. So in our problem, we also need to make sure our solution is feasible.

To create a certain interference model, I created a directional WLAN environment in MATLAB. I solved this problem for 10 links in a 10 meter by 10 meter room. 20 nodes were randomly deployed in the room and transmitter receiver pairs were made to form 10 links. Using a beamwidth of 30 degrees I simulated my environment to find out the interference caused by each link to every other link. The *interference.m* file in the appendix of this paper shows the code used. Through this code the interference values and the distance between each link was calculated and these values were later fed into different metaheuristics. This is a very realistic model for this problem.

The problem interference and distance values are also provided in the appendix.

3. Problem Encoding

Encoding a schedule is the key to the success of a metaheuristic method in solving any scheduling problems. The problem was encoded using the starting time of each link communication. The whole communication time is divided into time slots. A schedule of each link's starting time is the encoded solution I use for our metaheuristics. A possible non overlapping solution may look like this:

Solution = [0 145 303 474 658 856 1067 1291 1528 1778]

This means link 1 would start sending data at time 0 whereas link 10 would start sending data at time 1778.

I coded a function called the *comm_time.m* (given in the appendix). This is the cost function of our problem. If we give a possible solution to this function, it returns the schedule of each communication with starting and ending time, tells us if the solution is feasible and provides us with the maximum time this solution would take to transmit all data.

4. Genetic Algorithm

I kept a population size of 100. Out of these 100, the best 20 solutions were chosen to form parents for the prospective offspring. Both crossover and mutation move was used to obtain the children. Different crossover and mutation probability was used to exhaust the solution space and make sure there's no premature convergence.

The results of the genetic algorithm are shown in table 1. p_c is the crossover probability whereas the p_m is the mutation probability.

Possible shortcomings of this algorithm: Metaheuristic converging prematurely.

Problems	p_c = 0.7 p_m = 0.1	p_c = 0.7 p_m = 0.2	p_c = 0.7 p_m = 0.3	p_c = 0.8 p_m = 0.1	p_c = 0.8 p_m = 0.2	p_c = 0.8 p_m = 0.3	p_c = 0.9 p_m = 0.1	p_c = 0.9 p_m = 0.2	p_c = 0.9 p_m = 0.3
Pb 1	1099	**1050**	1370	1083	1320	1297	1152	1070	1131
Pb 2	482	645	**448**	452	463	626	463	449	**448**
Pb 3	553	501	**480**	500	513	501	501	503	552
Pb 4	746	726	756	757	517	713	782	729	**501**
Pb 5	622	**507**	524	525	580	579	666	570	580
Pb 6	718	673	507	677	**493**	645	710	727	660
Pb 7	395	395	395	**329**	447	395	395	396	461
Pb 8	475	477	451	476	474	410	477	**341**	552
Pb 9	434	447	490	447	439	448	500	447	**423**
Pb 10	582	527	579	609	528	516	504	571	**461**

Table 1. Genetic Algorithm results.

5. Simulated Annealing

A neighborhood was defined as a change of +- 10 timeslots for each link’s communication start time. The move that gave the best solution was selected out of the +- 10 timeslots.

For simulated annealing, I used a total of 3 different cooling schedules.

Type 1 =

$$T_i = T_0 - i\,\frac{T_0 - T_N}{N}$$

Type 2 =

$$T_i = T_0\left(\frac{T_N}{T_0}\right)^{\frac{i}{N}}$$

Type 3 =

$$T_i = \frac{A}{i+1} + B$$

$$A = \frac{(T_0 - T_N)\,(N + 1)}{N}$$

$$B = T_0 - A$$

Where Ti is the current temperature, T_O is the initial temperature and T_N is the final temperature.

Table 2 shows the results obtained from simple simulated annealing metaheuristic.

Problems	Type = 1	Type = 2	Type = 3
Pb 1	1578	1578	1578
Pb 2	842	842	842
Pb 3	553	553	553
Pb 4	725	725	725
Pb 5	580	580	580
Pb 6	672	672	672
Pb 7	317	317	317
Pb 8	475	475	475
Pb 9	422	422	422
Pb 10	737	737	737

Table 2. SA results.

The conventional SA approaches might get stuck in local optimum even though they make occasional moves to solution with higher costs (uphill moves). SA gets stuck in local optimum because with time the transition probability declines and the possibility of an uphill movement also decreases.

From the results in Table 2 we can see that the cooling schedule in an SA is not determining the final solution for our problem. This is understandable since in our problem once the solution converges to one local minimum it is a bit difficult for it to go to another minimum. This is because it requires a lot of uphill moves to escape one local minimum.

To alleviate this problem, several methods such as the multi-start approaches (Aarts et al. 1994) and adaptive temperature control mechanism (Kolonko 1999, Azizi and Zolfaghari 2004) have been suggested before. Similar to these approaches, I decided to implement a local search in our simulated annealing approach to help it escape local optimum.

6. Simulated Annealing with local search

The rationale behind implementing a local search with simulated annealing is to try to help the simulated annealing not get stuck in local optimum.

I implemented a local search with a swap operation. The frequency (F) decides how often the swap takes place in the iterations of simulated annealing. The local search would go through all the links starting time and swap one link’s starting time with another to make a candidate solution. If the candidate solution is feasible and if the candidate solution has a lower cost than the current solution then this candidate solution becomes the current solution.

Table 3 shows the results obtained after the local search mechanism. We can clearly see the improvement in the results compared to the simulated annealing results.

Problems	F = 2	F = 4	F = 6	F = 8	F = 10	F = 12	F = 14	F = 16	F = 18	F = 20
Pb 1	975	1104	1104	1040	1000	975	935	1209	**882**	1109
Pb 2	**448**	459	**448**	457	453	**448**	459	**448**	**448**	**448**
Pb 3	**421**	540	**421**	**421**	**421**	500	435	**421**	**421**	**421**
Pb 4	501	501	**500**	724	724	**500**	**500**	501	724	724
Pb 5	**421**	**421**	461	**421**	**421**	447	447	447	**421**	**421**
Pb 6	475	**421**	474	645	645	572	698	**421**	697	474
Pb 7	317	342	**316**	317	342	342	395	**316**	342	**316**
Pb 8	421	**342**	**342**	**342**	**342**	**342**	**342**	**342**	**342**	**342**
Pb 9	**421**	**421**	428	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 10	566	578	578	**500**	566	566	566	578	566	527

Table 3. Results for SA with local search

7. Simulated Annealing with local search followed by Genetic Algorithm

In my search to try to obtain the global optimum value, I implemented genetic algorithm on the schedules obtained from the metaheuristic in the previous section.

Due to shortage of space, I am not going to give all the results. Infact I am only going to show the best results obtained for different probability values of crossover and mutation. The p_c was changed from 0.7 to 0.9 and p_c was changed from 0.1 to 0.3. The results obtained are shown in Table 4.

As you can see, most of the problems converge to one single value no matter what the parameters for the local search are. These are the best results obtained so far. Showing that genetic algorithm alone converges prematurely. SA alone cannot always get out of local optimum. SA with local search is good but does not give good results for all swap frequencies. SA with local search followed by GA has given us the best results so far

Problems	F = 2	F = 5	F = 8	F = 11	F = 14	F = 17	F = 20
Pb 1	**750**	856	778	**750**	**750**	817	817
Pb 2	**448**	**448**	**448**	**448**	**448**	**448**	**448**
Pb 3	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 4	**500**	**500**	**500**	**500**	**500**	**500**	**500**
Pb 5	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 6	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 7	**316**	**316**	**316**	**316**	**316**	**316**	**316**
Pb 8	**342**	**342**	**342**	**342**	**342**	**342**	**342**
Pb 9	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 10	**448**	**448**	**448**	**448**	**448**	**448**	**448**

Table 4. SA with local search followed by GA

8. Multi-start Simulated Annealing with local search

The rationale behind using multi-start technique is to get more diversity in the metaheuristics to try and obtain global optimum for these specific problems. The problem is so complex that a diversification obtained from a single heuristic is usually not enough to get out of local optimum. This is why I want to implement a multi-start approach to try and start the metaheuristic with different initial solutions and then move from there. This provides more diversity for this specific problem than the previous heuristic. I did not follow up this metaheuristic with a genetic algorithm.

I kept a population size of 50. A total of 50 initial solutions are randomly made. On these solutions I ran simulated annealing with local search.

Table 5 shows the results obtained from this metaheuristic.

Problems	F = 2	F = 5	F = 8	F = 11	F = 14	F = 17	F = 20
Pb 1	778	**750**	921	923	921	**750**	**750**
Pb 2	**448**	**448**	**448**	**448**	**448**	**448**	**448**
Pb 3	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 4	**500**	**500**	**500**	**500**	**500**	**500**	**500**
Pb 5	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 6	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 7	**316**	**316**	**316**	**316**	**316**	**316**	**316**
Pb 8	**342**	**342**	**342**	**342**	**342**	**342**	**342**
Pb 9	**421**	**421**	**421**	**421**	**421**	**421**	**421**
Pb 10	**448**	**448**	**448**	**448**	**448**	**448**	**448**

Table 5. Multi-start Simulated Annealing with local search

9. Analysis and Conclusion

We can see from the results that this is not a normal scheduling problem. In this problem the interference model is so complicated that once the problem gets close to convergence it's very difficult to obtain diversity. The final schedule of our solution is in form of clusters. One such solution schedule is shown below:

Link	Start	End
1	1	291
2	150	317
3	1	173
4	1	150
5	150	299
6	1	150
7	1	226
8	150	239
9	2	150
10	12	170

It is obvious from looking at the solutions that the global optimum would be in form of clusters, so would all the local optimum. The only way to find the global optimum is to make clusters but then we take the risk of running into a local optimum. Once clusters are formed, many uphill moves are needed to break the clusters.

This is one of the reasons why adding diversity features from different metaheuristic is giving us better results. Simulated annealing gives fast results but often it fails to swap two different links start time because it requires too many uphill moves. Genetic algorithm also provides very good results but most times it converges prematurely.

Simulated annealing with local search provided one of the best results. This was partly because iterated local search provides a lot of diversification. This diversification works good at the start of the metaheuristic but is not always able to break the clusters.

Similarly having a multi-start with a random initial solution also gives us a lot of diversity at the earlier stages of the metaheuristic. Diversification at the start of the heuristic made sure different clusters were formed for each run. This is why multi-start is giving so much better results.

References

Aarts, E.H.L., et al., 1994. A computational study of local search algorithms for job shop scheduling. ORSA Journal on Computing, 6 (2), 118–125.

Aiex, R.M., Binato, S., and Resende, M.G.C., 2003. Parallel GRASP with path relinking for job shop scheduling. Parallel Computing, 29 (4), 393–430.

Azizi, Nader, Saeed Zolfaghari, and Ming Liang. "Hybrid simulated annealing with memory: an evolution-based diversification approach." International Journal of Production Research 48.18 (2010): 5455-5480.

Dell'Amico, M. and Trubian, M., 1993. Applying tabu search to the job shop scheduling problem. Annals of Operations Research, 41 (3), 231–252.

Dorigo, M. and Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1 (1), 53–66.

Feo, T. and Resende, M., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization, 16 (2), 109–133.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13 (5), 533–549.

Holland, J.H., 1975. Adaptation in natural and artificial systems. Ann Arbor, MI: The University of Michigan Press.

Kennedy, J. and Eberhart, R., 1995. Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, 27 November–1 December 1995, Perth, Australia, Vol. 1, 1942–1948.

Kirkpatrick, S., Gelatt, C.D. Jr, and Vecchi, M.P., 1983. Optimization by simulated annealing. Science, 220 (4598), 671–680.

Kolonko, M., 1999. Some new results on simulated annealing applied to the job shop scheduling problem. European Journal of Operational Research, 113 (1), 123–136.

Nowicki, E. and Smutnicki, C., 1996. A fast tabu search algorithm for the job shop problem. Management Science, 42 (6), 797–813.

Pantelidou, Anna, and Anthony Ephremides. "Minimum-length scheduling and rate control for time-varying wireless networks." *Military Communications Conference, 2009. MILCOM 2009. IEEE*. IEEE, 2009.

Rahoual, M. and Saad, R., 2006. Solving timetabling problems by hybridizing genetic algorithms and tabu search. In: Proceedings of the 6th international conference on practice and theory of automated timetabling, PATAT, 30 August–1 September 2006, Brno, Czech Republic, 467–472.

Ramjee, Tian Bu Li Erran Li Ramachandran. "Generalized Proportional Fair Scheduling in Third Generation Wireless Data Networks." INFOCOM 2006.

Sadi, Yalcin, and S. Coleri Ergen. "Minimum Length Scheduling with Packet Traffic Demands in Wireless Ad Hoc Networks." (2014): 1-1.

Santos, H.G., Ochi, L.S. and Souza, M.J.F., 2004. A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. In: Proceedings of the 5th international conference on practice and theory of automated timetabling, PATAT, 18–20 August 2004, Pittsburgh, USA, 343–359.

Taillard, E.D., et al., 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science, 31 (2), 170–186.

Taillard, E.D., et al., 2001. Adaptive memory programming: a unified view of metaheuristics. European Journal of Operational Research, 135 (1), 1–16.

Appendix

Interference.m

```
clear all;
close all;
clc;

c = 10;  % number of communications.
n = c*2; % number of nodes.
nodes = [];  % position of all the nodes.

BW = 30;  % beam width.

efficiency = 0.9; % Antenna efficiency.
Gm =  efficiency*(360/BW); % main lobe gain.
Gs = (1-efficiency)*(360/(360-BW)); % side lobe gain.

SINR_thr = 10^((5.5)/10); % not in dB.
PT = 10^((10-30)/10); % transmission power not in dBm.
Noise = 10^((-40-30)/10); % not in dBm. (-60 dBm)

Dmm = (0.005/((SINR_thr*Noise/(PT*Gm*Gm))^.5)*4*pi); % Not used. just to
check transmission range.
Dsm = (0.005/((SINR_thr*Noise/(PT*Gs*Gm))^.5)*4*pi);

rates = [18, 3807; 13, 1904; 5.5, 952];     % Transmission rates depending
on the SINR.

r = 10;    % creating a 10x10 meter room to deploy the nodes.

for i = 1:2:n
    check = 1;
    count = 0;
    while check == 1
        x1 = rand*r;
        y1 = rand*r;
        x2 = rand*r;
        y2 = rand*r;
        dist = ((x1-x2)^2+(y1-y2)^2)^0.5;
        if dist < Dmm    % Making sure the nodes created are not outside the
range.
            check = 0;
            nodes = [nodes; x1, y1; x2, y2];
        end
        count=count+1;
        if count > 1000;
            error('Error ! Nodes cannot be initialized with the
parameters')
        end
    end
end

int = zeros(c,c);
for i = 1:2:n
    for j = 1:2:n
        if j~=i
            d = ((nodes(i+1,1)-nodes(j,1))^2+(nodes(i+1,2)-
nodes(j,2))^2)^0.5;
```

```
            s =
inArc(nodes(i+1,:),nodes(i,:),nodes(j,:),BW)+inArc(nodes(j,:),nodes(j+1,:),
nodes(i+1,:),BW);
            if s == 2           % Interference caused by j for i.
                int((i+1)/2,(j+1)/2) = PT*Gm*Gm*(0.005/(4*pi*d));
            elseif s == 1
                int((i+1)/2,(j+1)/2) = PT*Gm*Gs*(0.005/(4*pi*d));
            else
                int((i+1)/2,(j+1)/2) = PT*Gs*Gs*(0.005/(4*pi*d));
            end
        end
    end
end

distance = zeros(c,1);
for i = 1:2:n
    distance((i+1)/2) = ((nodes(i+1,1)-nodes(i,1))^2+(nodes(i+1,2)-
nodes(i,2))^2)^0.5;
end

for i = 1:2:n
    P = PT*Gm*Gm*(0.005/(4*pi*distance((i+1)/2)));
    SINR = P/(sum(int(((i+1)/2),:))+Noise);
end
```

inArc.m

```
function [i] = inArc(x, y, z, BW)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

a = ((x(1)-y(1))^2+(x(2)-y(2))^2)^0.5;
b = ((x(1)-z(2))^2+(x(2)-z(2))^2)^0.5;
c = ((y(1)-z(1))^2+(y(2)-z(2))^2)^0.5;

angle = acosd((a^2+b^2-c^2)/(2*a*b));

if abs(angle)<BW/2
    i = 1;
else
    i = 0;
end
end
```

Comm_time.m

```
function [i, schedule] = comm_time(solution, data, int, c, d)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

rates = [18, 3807; 13, 1904; 5.5, 952];
BW = 30;  % beam width.
efficiency = 0.9; % Antenna efficiency.
Gm =  efficiency*(360/BW); % main lobe gain.
PT = 10^((10-30)/10); % transmission power not in dBm.
Noise = 10^((-40-30)/10); % not in dBm. (-60 dBm).

solution2 = zeros(c,2);
for i = 1:c
    solution2(i,:) = [i, solution(i)];
end
solution2 = sortrows(solution2, 2);

schedule = zeros(c,3);
for i = 1:c
    index = solution2(i,1);
    inter = 0;
    for j = 1:i-1
        if solution2(i,2)>schedule(j,2) && solution2(i,2)<schedule(j,3)
            inter = inter+int(index,schedule(j,1));
        end
    end

    P = PT*Gm*Gm*(0.005/(4*pi*d(index)));
    SINR = P/(inter+Noise);
    c2 = 0;
    for c1 = 1:size(rates,1)
        if SINR > 10^((rates(c1,1))/10) && c2 == 0
            r = rates(c1,2);
            c2 = 1;
        end
    end
    if c2 == 0
        i = 0;
        return;
    end

    endpoint = solution2(i,2)+ceil((data(index)/r)*10^3);

    inter2 = inter;
    change = 1;
    while change == 1
        for j = i+1:c
            if solution2(j,2) < endpoint
                inter2 = inter2 + int(index,solution2(j,1));
            end
        end
        SINR = P/(inter2+Noise);
        c2 = 0;
        for c1 = 1:size(rates,1)
            if SINR > 10^((rates(c1,1))/10) && c2 == 0
                prev = r;
                r = rates(c1,2);
                c2 = 1;
```

```
            end
        end
        if c2 == 0
            i = 0;
            return;
        end
        if prev ~= r
            endpoint = solution2(i,2)+ceil((data(index)/r)*10^3);
            inter2 = inter;
            change = 1;
        else
            change = 0;
        end
    end
    schedule(i,:) = [solution2(i,1), solution2(i,2), endpoint];
end
schedule = sortrows(schedule,1);
i = max(schedule(:,3));
end
```

SA.m

```
clear all;
close all;
clc;

c = 10; % number of communications.

fileID = fopen('SA.txt','wt');

for type = 1:3

lengths = zeros(10,1);

for iter = 1:10
[int, distance] = getpara(iter);
data = [550,600,650,700,750,800,850,900,950,1000]; % Data in kilobytes each
communication will send.

% Creating initial solution with no overlap.
init = [];
prev = 0;
for i = 1:c
    time = ceil((data(i)/3807)*10^3);
    init(i) = prev;
    prev = prev+time;
end

solution = init;
[length, schedule] = comm_time(solution, data, int, c, distance);

To = 1;      % Initial Temperature.
Tn = 1e-006;    % Final Temperature.
```

```
Ti = To;
N = 200;  % For the cooling schedule.
i = 0;

length3 = zeros(1,N);
length = 1000000000;

%schedule
while Ti >= Tn    % Stopping condition
    i= i+1;
    % looping through each communication
    for j = 1:c
        % defining a neighbor
        best = 10000000;
        for k = -10:1:10
            if k ~= 0 && (solution(j)+k)>0
                [length2, schedule2] = comm_time([solution(1:j-
1),solution(j)+k,solution(j+1:c)], data, int, c, distance);
                if length2 < best && length2 ~= 0
                    best = length2;
                    solution2 = [solution(1:j-
1),solution(j)+k,solution(j+1:c)];
                end
            end
        end

        if best ~= 10000000
        d = best - length;
        % Generating a random number
        n = rand;
        if d <= 0
            length = best;
            schedule = schedule2;
            solution = solution2;
        elseif n < exp(-d/Ti)
            length = best;
            schedule = schedule2;
            solution = solution2;
        end
        end
    end
    length3(i) = length;
    if type==1
        Ti = To-i*((To-Tn)/N);
    elseif type == 2
        Ti = To*(Tn/To)^(i/N);
    elseif type == 3
        Ti = ((To-Tn)*(N+1))/(N*(i+1)) + To - ((To-Tn)*(N+1))/(N);
    end
end
length;
length3;
schedule;
distance;
lengths(iter)=length
end

lengths;

fprintf(fileID, '\n\n %s %d \n', sprintf('Type = '), type);
```

```
fprintf(fileID,' %d,   ',lengths);

end

fclose(fileID);
```

GA.m

```
close all;
clear all;
clc;

c = 10; % number of communications.

fileID = fopen('GA.txt','wt');

for p_c = 0.7:0.1:0.9  % Cross over probability
for p_m = 0.1:0.1:0.3;  % Mutation probability

lengths = zeros(10,1);

for iter = 1:10

[int, distance] = getpara(iter);
data = [550,600,650,700,750,800,850,900,950,1000]; % Data in kilobytes each
communication will send.

pop_size = 100;
mating_pool_size = 20;
num_of_generations = 500;

solutions = zeros(100,c);
solutions_value = zeros(100,1);

%randomly generating solutions.
for i = 1:pop_size
    check = 1;
    while(check)
        s = randi(1500,1,c);
        mini2 = min(s);
        if mini2 ~= 0
        for j = 1:c
            s(j) = s(j)-mini2;
        end
        end
        [length, schedule] = comm_time(s, data, int, c, distance);
        if length ~= 0
            solutions(i,:) = s;
            solutions_value(i) = length;
            check = 0;
        end
    end
end
min(solutions_value); % FOR PRINT !!
```

```
gen = 0;
while (gen<num_of_generations)

% selecting parents from the pool
parents = zeros(mating_pool_size,c);
parents_v = zeros(1,mating_pool_size);
for i = 1:mating_pool_size
    [minimum, index] = min(solutions_value);
    parents(i,:) = solutions(index,:);
    parents_v(i) = solutions_value(index);
    solutions_value(index) = [];
    solutions(index,:) = [];
end
    solutions2 = zeros(pop_size-mating_pool_size,c);
    for i = 1:2:(pop_size-mating_pool_size)
        index = randperm(mating_pool_size,2);

        if (rand<p_c)
            tries = 0;
            success = 0;
            while(success==0 && tries < 5)
                index2 = randi(c);
                s1 = parents(index(1),:);
                s2 = parents(index(2),:);
                s1(index2) = parents(index(2),index2);
                s2(index2) = parents(index(1),index2);
                [length1, schedule] = comm_time(s1, data, int, c,
distance);
                [length2, schedule2] = comm_time(s2, data, int, c,
distance);
                if length1 ~= 0 && length2 ~= 0
                    success = 1;
                    mini2 = min(s1);
                    if mini2 ~= 0
                        for j = 1:c
                            s1(j) = s1(j)-mini2;
                        end
                    end
                    mini2 = min(s2);
                    if mini2 ~= 0
                        for j = 1:c
                            s2(j) = s2(j)-mini2;
                        end
                    end
                    solutions2(i,:) = s1;
                    solutions2(i+1,:) = s2;
                end
                tries = tries+1;
            end
            if (success==0 && tries==5)
                solutions2(i,:) = parents(index(1),:);
                solutions2(i+1,:) = parents(index(2),:);
            end
        else
            solutions2(i,:) = parents(index(1),:);
            solutions2(i+1,:) = parents(index(2),:);
        end

        if (rand<p_m)
```

```
            tries = 0;
            success = 0;
            s1 = solutions2(i,:);
            s2 = solutions2(i+1,:);
            while (success==0  && tries < 3)
                index2 = randi(c);
                s1(index2) = randi(1000);
                s2(index2) = randi(1000);
                [length1, schedule] = comm_time(s1, data, int, c,
distance);
                [length2, schedule2] = comm_time(s2, data, int, c,
distance);
                if length1 ~= 0 && length2 ~= 0
                    success = 1;
                    mini2 = min(s1);
                    if mini2 ~= 0
                        for j = 1:c
                            s1(j) = s1(j)-mini2;
                        end
                    end
                    mini2 = min(s2);
                    if mini2 ~= 0
                        for j = 1:c
                            s2(j) = s2(j)-mini2;
                        end
                    end
                    solutions2(i,:) = s1;
                    solutions2(i+1,:) = s2;
                end
                tries = tries+1;
            end
        end
    end
    solutions = [parents;solutions2];
    for i = 1:pop_size
        [length, schedule] = comm_time(solutions(i,:), data, int, c,
distance);
        solutions_value(i) = length;
    end
    gen = gen + 1;
    min(solutions_value);
end
[minimum, index] = min(solutions_value);
solutions_value(index);
solutions(index,:);
[length, schedule] = comm_time(solutions(index,:), data, int, c, distance);
lengths(iter) = length
end
lengths

fprintf(fileID, '\n\n %s %d %s %d \n', sprintf('P_c = '), p_c, sprintf('
P_m = '), p_m);
fprintf(fileID,' %d,  ',lengths);

end
end

fclose(fileID);
```

SA with local search.

```
clear all;
close all;
clc;

c = 10; % number of communications.

fileID = fopen('SAswap.txt','wt');

for type = 2:3
for freq = 2:2:20

lengths = zeros(10,1);

for iter = 1:10
[int, distance] = getpara(iter);
data = [550,600,650,700,750,800,850,900,950,1000]; % Data in kilobytes each communication will send.

% Creating initial solution with no overlap.
init = [];
prev = 0;
for i = 1:c
    time = ceil((data(i)/3807)*10^3);
    init(i) = prev;
    prev = prev+time;
end

% Creating a random initial solution
init3 = [];
check = 1;
while(check)
    init3 = randi(1500,1,c);
    mini2 = min(init3);
    if mini2 ~= 0
        for j = 1:c
            init3(j) = init3(j)-mini2;
        end
    end
    [length, schedule] = comm_time(init3, data, int, c, distance);
    if length ~= 0
        check = 0;
    end
end

solution = init3;
[length, schedule] = comm_time(solution, data, int, c, distance);

To = 1;      % Initial Temperature.
Tn = 1e-006;    % Final Temperature.
Ti = To;
N = 200;  % For the cooling schedule.
i = 0;

length3 = zeros(1,N);
```

```
while Ti >= Tn     % Stopping condition
    i= i+1;
    % looping through each communication
    for j = 1:c
        % defining a neighbor
        best = 10000000;
        for k = -10:1:10
            if k ~= 0 && (solution(j)+k)>=0
                [length2, schedule2] = comm_time([solution(1:j-
1),solution(j)+k,solution(j+1:c)], data, int, c, distance);
                if length2 < best && length2 ~= 0
                    best = length2;
                    solution2 = [solution(1:j-
1),solution(j)+k,solution(j+1:c)];
                    schedule3 = schedule2;
                end
            end
        end

        if best ~= 10000000
        d = best - length;
        % Generating a random number
        n = rand;
        if d <= 0
            length = best;
            schedule = schedule3;
            solution = solution2;
        elseif n < exp(-d/Ti)
            length = best;
            schedule = schedule3;
            solution = solution2;
        end
        end
    end

    % Swapping right here.
    if (i/freq) == fix(i/freq)
        best2 = 10000;
        for x1 = 1:c-1
        for x2 = x1+1:c
        [length4, schedule4] = comm_time([solution(1:x1-
1),solution(x2),solution(x1+1:x2-1),solution(x1),solution(x2+1:c)], data,
int, c, distance);
        if length4 <= best2 && length4 ~= 0
            best2 = length4;
            schedule5 = schedule4;
            best2solution = [solution(1:x1-
1),solution(x2),solution(x1+1:x2-1),solution(x1),solution(x2+1:c)];
        end
        end
        end
        if best2 <= length
            length = best2;
            schedule = schedule5;
            solution = best2solution;
        end
    end

    length3(i) = length;
```

```
    if type==1
        Ti = To-i*((To-Tn)/N);
    elseif type == 2
        Ti = To*(Tn/To)^(i/N);
    elseif type == 3
        Ti = ((To-Tn)*(N+1))/(N*(i+1)) + To - ((To-Tn)*(N+1))/(N);
    end
end

length3;
length;
solution;
schedule;
lengths(iter) = length;
[type, freq, iter]
end
lengths;

fprintf(fileID, '\n\n %s %d %s %d \n', sprintf('Type = '), type, sprintf('
Freq = '), freq);
fprintf(fileID,' %d,  ',lengths);

end
end

fclose(fileID);
```

SA with local search followed by genetic algorithm.

```
clear all;
close all;
clc;

c = 10; % number of communications.

fileID = fopen('SAswapGA.txt','wt');

for type = 1
for freq = 2:3:20
for p_c = 0.7:0.1:0.9  % Cross over probability
for p_m = 0.1:0.1:0.3;  % Mutation probability

lengths = zeros(10,1);

for iter = 1:10

[int, distance] = getpara(iter);
data = [550,600,650,700,750,800,850,900,950,1000]; % Data in kilobytes each
communication will send.

% Creating initial solution with no overlap.
init = [];
prev = 0;
for i = 1:c
    time = ceil((data(i)/3807)*10^3);
```

```
    init(i) = prev;
    prev = prev+time;
end

bestone = 10000;
solutions = zeros(100,c);
solutions_value = zeros(100,1);

% CREATING 10 SOLUTIONS FROM RANDOM SA
for iter2 = 1:10
% Creating a random initial solution
init3 = [];%
check = 1;
while(check)
    init3 = randi(1500,1,c);
    mini2 = min(init3);
    if mini2 ~= 0
        for j = 1:c
            init3(j) = init3(j)-mini2;
        end
    end
    [length, schedule] = comm_time(init3, data, int, c, distance);
    if length ~= 0
        check = 0;
    end
end

solution = init3;
[length, schedule] = comm_time(solution, data, int, c, distance);

To = 1;      % Initial Temperature.
Tn = 1e-006;    % Final Temperature.
Ti = To;
N = 200;  % For the cooling schedule.
i = 0;

length3 = zeros(1,N);

while Ti >= Tn     % Stopping condition
    i= i+1;
    % looping through each communication
    for j = 1:c
        % defining a neighbor
        best = 10000000;
        for k = -10:1:10
            if k ~= 0 && (solution(j)+k)>=0
                [length2, schedule2] = comm_time([solution(1:j-
1),solution(j)+k,solution(j+1:c)], data, int, c, distance);
                if length2 < best && length2 ~= 0
                    best = length2;
                    solution2 = [solution(1:j-
1),solution(j)+k,solution(j+1:c)];
                    schedule3 = schedule2;
                end
            end
        end

        if best ~= 10000000
        d = best - length;
        % Generating a random number
```

```
        n = rand;
        if d <= 0
            length = best;
            schedule = schedule3;
            solution = solution2;
        elseif n < exp(-d/Ti)
            length = best;
            schedule = schedule3;
            solution = solution2;
        end
        end
    end

    % Swapping right here.
    if (i/freq) == fix(i/freq)
        best2 = 10000;
        for x1 = 1:c-1
        for x2 = x1+1:c
        [length4, schedule4] = comm_time([solution(1:x1-
1),solution(x2),solution(x1+1:x2-1),solution(x1),solution(x2+1:c)], data,
int, c, distance);
        if length4 <= best2 && length4 ~= 0
            best2 = length4;
            schedule5 = schedule4;
            best2solution = [solution(1:x1-
1),solution(x2),solution(x1+1:x2-1),solution(x1),solution(x2+1:c)];
        end
        end
        end
        if best2 <= length
            length = best2;
            schedule = schedule5;
            solution = best2solution;
        end
    end
    length3(i) = length;
    if type==1
        Ti = To-i*((To-Tn)/N);
    elseif type == 2
        Ti = To*(Tn/To)^(i/N);
    elseif type == 3
        Ti = ((To-Tn)*(N+1))/(N*(i+1)) + To - ((To-Tn)*(N+1))/(N);
    end
end
solutions(iter2,:)=solution;
solutions_value(iter2) = length;
[iter, iter2, length]; % PRINTING
if length < bestone
    bestsolution = solution;
    bestschedule = schedule;
    bestone = length;
end
end

bestsolution;
bestschedule;
bestone;

% GA STARTS FROM HERE.
```

```
pop_size = 100;
mating_pool_size = 20;
num_of_generations = 500;

for i = 10:pop_size
    check = 1;
    while(check)
        s = randi(1500,1,c);
        mini2 = min(s);
        if mini2 ~= 0
        for j = 1:c
            s(j) = s(j)-mini2;
        end
        end
        [length, schedule] = comm_time(s, data, int, c, distance);
        if length ~= 0
            solutions(i,:) = s;
            solutions_value(i) = length;
            check = 0;
        end
    end
end

min(solutions_value);
solutions_value;

gen = 0;
while (gen<num_of_generations)

% selecting parents from the pool
parents = zeros(mating_pool_size,c);
parents_v = zeros(1,mating_pool_size);
for i = 1:mating_pool_size
    [minimum, index] = min(solutions_value);
    parents(i,:) = solutions(index,:);
    parents_v(i) = solutions_value(index);
    solutions_value(index) = [];
    solutions(index,:) = [];
end
    solutions2 = zeros(pop_size-mating_pool_size,c);
    for i = 1:2:(pop_size-mating_pool_size)
        index = randperm(mating_pool_size,2);
        if (rand<p_c)
            tries = 0;
            success = 0;
            while(success==0 && tries < 5)
                index2 = randi(c);
                s1 = parents(index(1),:);
                s2 = parents(index(2),:);
                s1(index2) = parents(index(2),index2);
                s2(index2) = parents(index(1),index2);
                [length1, schedule] = comm_time(s1, data, int, c,
distance);
                [length2, schedule2] = comm_time(s2, data, int, c,
distance);
                if length1 ~= 0 && length2 ~= 0
                    success = 1;
                    mini2 = min(s1);
                    if mini2 ~= 0
                        for j = 1:c
```

```
                    s1(j) = s1(j)-mini2;
                end
            end
            mini2 = min(s2);
            if mini2 ~= 0
                for j = 1:c
                    s2(j) = s2(j)-mini2;
                end
            end
            solutions2(i,:) = s1;
            solutions2(i+1,:) = s2;
        end
        tries = tries+1;
    end
    if (success==0 && tries==5)
        solutions2(i,:) = parents(index(1),:);
        solutions2(i+1,:) = parents(index(2),:);
    end
else
    solutions2(i,:) = parents(index(1),:);
    solutions2(i+1,:) = parents(index(2),:);
end

if (rand<p_m)
    tries = 0;
    success = 0;
    s1 = solutions2(i,:);
    s2 = solutions2(i+1,:);
    while (success==0  && tries < 3)
        index2 = randi(c);
        s1(index2) = randi(1000);
        s2(index2) = randi(1000);
        [length1, schedule] = comm_time(s1, data, int, c,
distance);
        [length2, schedule2] = comm_time(s2, data, int, c,
distance);
        if length1 ~= 0 && length2 ~= 0
            success = 1;
            mini2 = min(s1);
            if mini2 ~= 0
                for j = 1:c
                    s1(j) = s1(j)-mini2;
                end
            end
            mini2 = min(s2);
            if mini2 ~= 0
                for j = 1:c
                    s2(j) = s2(j)-mini2;
                end
            end
            solutions2(i,:) = s1;
            solutions2(i+1,:) = s2;
        end
        tries = tries+1;
    end
end
end
solutions = [parents;solutions2];
for i = 1:pop_size
    [length, schedule] = comm_time(solutions(i,:), data, int, c,
distance);
```

```
        solutions_value(i) = length;
    end
    gen = gen + 1;
    min(solutions_value);
end

[minimum, index] = min(solutions_value);
solutions_value(index);
solutions(index,:);
[length, schedule] = comm_time(solutions(index,:), data, int, c, distance);
lengths(iter) = length;
%iter

[freq, p_c, p_m, iter]  % PRINTING !!
end

fprintf(fileID, '\n\n %s %d %s %d \n', sprintf('P_c = '), p_c, sprintf('
P_m = '), p_m, sprintf(', Freq = '), freq, sprintf(',  Type = '), type);
fprintf(fileID,' %d,  ',lengths);

end
end
end
end

fclose(fileID);
```

Mutlistart population based SA with local search.

```
clear all;
close all;
clc;

c = 10; % number of communications.

fileID = fopen('SAswapPopulation.txt','wt');

for type = 1:3
for freq = 2:3:20

lengths = zeros(10,1);

for iter = 1:10
[int, distance] = getpara(iter);
data = [550,600,650,700,750,800,850,900,950,1000]; % Data in kilobytes each
communication will send.

% Creating initial solution with no overlap.
init = [];
prev = 0;
for i = 1:c
    time = ceil((data(i)/3807)*10^3);
    init(i) = prev;
    prev = prev+time;
end
```

```
bestone = 10000;
solutions = zeros(100,c);
solutions_value = zeros(100,1);

% population size.
for iter2 = 1:50
% Creating a random initial solution
init3 = [];%
check = 1;
while(check)
    init3 = randi(1500,1,c);
    mini2 = min(init3);
    if mini2 ~= 0
        for j = 1:c
            init3(j) = init3(j)-mini2;
        end
    end
    [length, schedule] = comm_time(init3, data, int, c, distance);
    if length ~= 0
        check = 0;
    end
end

solution = init3;
[length, schedule] = comm_time(solution, data, int, c, distance);

To = 1;      % Initial Temperature.
Tn = 1e-006;    % Final Temperature.
Ti = To;
N = 200;  % For the cooling schedule.
i = 0;

length3 = zeros(1,N);

while Ti >= Tn     % Stopping condition
    i= i+1;
    % looping through each communication
    for j = 1:c
        % defining a neighbor
        best = 10000000;
        for k = -10:1:10
            if k ~= 0 && (solution(j)+k)>=0
                [length2, schedule2] = comm_time([solution(1:j-
1),solution(j)+k,solution(j+1:c)], data, int, c, distance);
                if length2 < best && length2 ~= 0
                    best = length2;
                    solution2 = [solution(1:j-
1),solution(j)+k,solution(j+1:c)];
                    schedule3 = schedule2;
                end
            end
        end

        if best ~= 10000000
        d = best - length;
        % Generating a random number
        n = rand;
        if d <= 0
            length = best;
```

```
            schedule = schedule3;
            solution = solution2;
        elseif n < exp(-d/Ti)
            length = best;
            schedule = schedule3;
            solution = solution2;
        end
        end
    end

    % Swapping right here.
    if (i/freq) == fix(i/freq)
        best2 = 10000;
        for x1 = 1:c-1
        for x2 = x1+1:c
        [length4, schedule4] = comm_time([solution(1:x1-
1),solution(x2),solution(x1+1:x2-1),solution(x1),solution(x2+1:c)], data,
int, c, distance);
        if length4 <= best2 && length4 ~= 0
            best2 = length4;
            schedule5 = schedule4;
            best2solution = [solution(1:x1-
1),solution(x2),solution(x1+1:x2-1),solution(x1),solution(x2+1:c)];
        end
        end
        end
        if best2 <= length
            length = best2;
            schedule = schedule5;
            solution = best2solution;
        end
    end
    length3(i) = length;
    if type==1
        Ti = To-i*((To-Tn)/N);
    elseif type == 2
        Ti = To*(Tn/To)^(i/N);
    elseif type == 3
        Ti = ((To-Tn)*(N+1))/(N*(i+1)) + To - ((To-Tn)*(N+1))/(N);
    end
end
solutions(iter2,:)=solution;
solutions_value = length;
[type, freq, iter, iter2]
if length < bestone
    bestsolution = solution;
    bestschedule = schedule;
    bestone = length;
end
length3;
end

bestsolution;
bestschedule;
bestone;
lengths(iter) = bestone;
end
lengths;

fprintf(fileID, '\n\n %s %d %s %d \n', sprintf('Type = '), type, sprintf('
Freq = '), freq);
```

```
fprintf(fileID,' %d,   ',lengths);

end
end

fclose(fileID);
```

Pb 1

```
int = [0,1.22709305482581e-06,1.47852938733153e-06,7.70146794529873e-
06,2.46979038818382e-07,1.70403457094463e-06,3.13692006726011e-
07,9.19561596773060e-06,7.76362265624550e-08,1.24135923998646e-
07;6.87438922255498e-06,0,1.04416527945753e-07,5.97720041809625e-
07,2.01218136795662e-07,1.43639487050942e-05,1.73562268527142e-
07,8.01238512450863e-06,8.57857809981118e-08,9.99998148512566e-
08;7.05020698081189e-08,5.75851654448544e-08,0,7.49107955092176e-
08,1.09573677771301e-07,7.28704027719164e-08,8.01160571098170e-
08,7.19613069912704e-08,2.28117855042213e-06,1.19445818729757e-
06;8.47754572179062e-06,1.12677673538353e-06,1.72353884980733e-
07,0,8.63596780385423e-07,1.68457698347485e-06,2.77118293335465e-
07,1.01506695437514e-05,9.81365898296951e-08,1.57653273641589e-
07;1.48622729413555e-06,9.18205717788020e-08,3.10948522223905e-
07,1.59148639427293e-06,0,1.27700227932321e-07,1.15055010134497e-
07,1.55469534325286e-07,1.15135305772997e-05,2.77274016698252e-
07;7.38485794772425e-06,9.45019312410158e-07,1.22201140227945e-
07,7.53048118682347e-06,2.90641215699238e-07,0,2.45464891878352e-
05,8.58213451548090e-06,1.02247919199336e-07,1.32750572258829e-
06;5.98567877592116e-07,8.34057050871456e-06,1.00289987796079e-
07,6.16620593795974e-07,1.88641584565534e-06,1.06558402195861e-
05,0,7.70217140998960e-06,1.81346987475511e-07,1.06576435433105e-
06;8.12263077253685e-06,1.27380178976077e-06,1.48693939750483e-
07,8.22248266626791e-06,3.39414574610422e-07,1.89293494927052e-
06,4.08008744267766e-07,0,8.19820863703516e-08,1.44117142328390e-
07;8.15036478751020e-08,5.99060012051663e-08,1.25108140104606e-
07,8.76557047515035e-08,1.22488963720515e-06,7.58107340748548e-
08,7.99447614641537e-08,8.16063136964853e-08,0,1.11118249596159e-
07;4.55157983816220e-07,6.42611418198387e-08,1.00052775577227e-
06,6.33998086126288e-06,8.33583705585726e-06,8.18948831553451e-
07,7.01683788841781e-07,1.69744875812500e-07,7.05463143602490e-06,0];
distance =
[9.20601331911183;6.31250575438819;4.46707833566838;8.11555269618120;3.8841
2545822717;4.71648366296647;4.95147604412916;7.10621456782718;3.68580508463
263;5.87631720094307];
```

Pb 2

```
int = [0,4.93412438258081e-09,5.63363333365114e-07,5.06971572914750e-
09,1.60632524936286e-08,4.62134561308125e-09,8.25983358241462e-
09,5.19708248613336e-09,5.38046361654828e-09,5.75076566458452e-
09;7.33847234417750e-09,0,5.93858469138885e-09,6.94869441502686e-
09,1.19666467061044e-08,7.37892988218639e-07,1.04671963288454e-
08,8.89701705057925e-09,9.45689868577020e-09,9.95766270211346e-
09;7.09957663095554e-07,4.76481189093221e-09,0,5.06293351603299e-
09,1.75517540102484e-08,4.47007633919972e-09,8.41189105767308e-
09,5.02799882528605e-09,5.08018952901076e-09,5.59739171342525e-
09;6.19754883941378e-09,2.72641576564618e-08,5.59483889656425e-
```

```
07,0,5.49667915799838e-09,2.64026069797716e-08,7.54331182691268e-
09,3.20514166063505e-08,8.90009474619192e-09,2.29235465669113e-
06;8.13187566187573e-08,5.40146714213230e-09,2.86308620886780e-
08,9.43275110136830e-09,0,5.02013857571607e-07,2.04567302442314e-
08,5.91842660539312e-09,4.51549564459701e-09,7.28791319929399e-
09;1.06440699113584e-07,5.42520394890639e-07,4.20176588221256e-
08,1.02616915198217e-08,1.10697874364786e-08,0,1.91221908621772e-
06,5.95656272444005e-07,4.43822090651906e-07,7.44645573848943e-
09;9.04061279928857e-07,1.50409939225749e-08,7.74392136745331e-
09,2.02856588144962e-08,7.70851101816623e-09,1.28999542265374e-
08,0,1.99136601121633e-08,8.15556704471567e-09,5.30159249016231e-
08;5.32291179245914e-07,4.79619208351045e-08,4.90009403925732e-
09,9.13382561969007e-09,5.09985686459546e-09,9.38502796608044e-
08,6.51851925125543e-09,0,1.10284140463989e-08,1.65863480438220e-
08;6.55571021540275e-09,5.25023358323692e-09,5.43074803303034e-
09,5.12411048623575e-09,1.40306043793269e-08,4.90334407435987e-
09,8.09012879343620e-09,5.51495835159262e-09,0,6.04477761984025e-
09;5.36980091355774e-09,1.46118802292773e-08,5.08902322837019e-
09,1.04997037896527e-08,4.60805079139603e-09,1.60796079780615e-
08,6.12916392988493e-09,1.45734594075797e-08,7.03174975460858e-09,0];
distance =
[6.87778684114923;5.71948479846412;7.99914092717397;3.77621865831559;3.7302
1366802836;9.18862846534021;3.71142625281994;1.48245187967248;7.97808926475
038;3.80944544855903];
```

Pb 3

```
int = [0,7.23693271119680e-09,6.83867232421291e-08,7.30018961924950e-
07,9.57473315168427e-09,8.76411642248651e-09,2.26999119288555e-
06,5.78995549908915e-09,8.54995244615796e-09,5.89528757616416e-
05;1.78792319206595e-08,0,1.66045781640919e-08,8.98302462829869e-
09,7.91455058324792e-09,9.44214881973288e-07,8.69958881526906e-
09,9.68670982249813e-09,8.67162077390345e-09,1.01369504908097e-
08;1.39410487100645e-08,2.25677878548375e-08,0,7.38540717926773e-
09,6.20367768782359e-09,7.43012110241511e-09,6.94022226328945e-
07,1.07421427920141e-08,6.83483692131854e-09,1.08626043310467e-
08;1.39100994768503e-08,6.92101565354870e-08,7.35722510262546e-
09,0,5.47990280388637e-07,6.98967933753105e-09,5.26687551426849e-
09,1.95061390844361e-08,6.39442559461873e-09,1.82706171900022e-
08;2.19936859697301e-08,9.88538975984147e-09,8.68545042812328e-
09,5.09964649571337e-08,0,2.80319192096045e-08,5.88586551857070e-
07,1.08288817164663e-08,2.05999631351849e-08,1.21530972036144e-
08;1.45898255185549e-08,1.24084348462639e-08,2.24552248561293e-
08,8.41909090766786e-07,8.18627092632241e-09,0,1.00890643932466e-
06,8.34219615337324e-09,8.52202333215845e-07,8.69754040195589e-
09;1.77980734169218e-08,9.41134032360006e-09,7.35873250206982e-
07,4.28457724781995e-08,1.03112668949144e-08,2.01538540117718e-
08,0,1.14018234242990e-08,1.65028086278642e-08,1.28598743075577e-
08;1.12303690305353e-08,6.74550954083147e-07,9.15265401081744e-
09,2.28729103873584e-08,3.22329237154359e-08,5.29344040307913e-
08,6.69953839563263e-09,0,1.65928428383366e-07,7.27778072569380e-
07;9.63348236005897e-09,6.26815569862522e-09,1.02807796853890e-
08,1.42593129793874e-08,1.48179306442474e-07,2.34753127920691e-
08,7.75606850271200e-09,6.02876498888752e-09,0,6.40484686692311e-
09;2.37204991648508e-08,1.10251886579062e-08,1.59719944330962e-
08,1.51224995836420e-08,1.28498918210031e-08,1.84465711087856e-
08,8.48080343652361e-09,9.02531964494843e-05,1.56037727351182e-08,0];
distance =
[5.27142736129224;3.02390795962280;4.39779425735386;6.32560806501752;3.9220
```

1323132768;5.07244189102437;8.89727793547055;6.91430776570304;1.56695881569377;4.74671337586060];

Pb 4

int = [0,5.78829605000429e-09,1.84627122692675e-08,8.66580869488464e-07,5.34192975290830e-09,5.49289562065533e-09,5.89020560264879e-09,8.39278264904798e-09,5.18069042664057e-09,1.33078861633296e-08;8.41730001060932e-09,0,1.08346209603494e-08,2.83085653411004e-08,5.82184350986439e-07,5.85913766541480e-07,1.04434705240182e-08,2.51529164518804e-08,4.76810975958046e-07,8.44460873182803e-09;1.44022960455598e-08,1.21989627241543e-08,0,6.82973706824986e-09,8.62712291699980e-09,9.26783125006403e-09,5.61279215154125e-07,6.62350848013423e-09,1.00207749582214e-08,5.02186574384074e-08;5.72751091963565e-09,4.71619949911753e-07,7.02788825420555e-09,0,5.57802675396287e-09,5.51675443129897e-09,1.78094093875009e-06,4.23307152341728e-06,4.23145256198622e-07,6.01075441526878e-09;5.09475534166057e-09,6.07568592663705e-07,6.60944753128025e-09,1.22442336516576e-08,0,8.29299978607985e-07,2.89903921086016e-08,1.25242479700701e-08,5.36238284316769e-07,5.97684450178616e-07;9.26923278326769e-09,8.07549092177722e-09,1.57349940263564e-08,1.57125236307548e-08,9.06113113754836e-09,0,1.07561655373077e-08,1.47833833386614e-08,6.69234942439482e-07,1.22346616133840e-08;5.52093483196537e-07,5.02222240350908e-09,6.99391879796012e-09,3.11500122130722e-08,6.02989127016599e-07,5.93136794266958e-07,0,3.65978215577443e-08,4.48613636263959e-09,5.98802784092604e-07;4.91367716440344e-09,5.19486671581863e-09,6.17571711889429e-09,1.47301089275427e-06,6.79737316499527e-09,6.51633627667227e-07,1.53611496945574e-05,0,4.65011741142190e-09,5.43366576233277e-05;7.77397700899918e-09,9.36558633818160e-07,1.22276479311185e-08,1.23790329042440e-08,1.21699708404280e-08,1.21525905523916e-06,1.12003329912194e-08,1.19744829068570e-08,0,1.08006085534988e-08;5.26544366534767e-07,5.38037960480679e-09,6.72096139433237e-07,1.95934302913199e-08,6.91036908764328e-09,6.66288333730555e-07,9.77095955548675e-06,2.09827796881528e-08,4.78459921841193e-09,0];
distance =
[2.13007399703610;8.70042013961739;2.04095289084115;1.32794354208334;5.39485111463411;5.13733116261523;1.68674881962118;3.01114461668843;6.14112214276126;7.89480854174655];

Pb 5

int = [0,1.53593864096530e-08,9.78202547715056e-07,6.78701367409507e-09,9.05513862563338e-09,1.46274130648641e-08,9.53396115104179e-09,7.54175145575401e-09,1.91573440144308e-08,1.64747860734785e-08;8.03865248540351e-09,0,4.88911191717748e-09,4.99248822705908e-09,1.28319800998845e-08,6.15732611856918e-09,5.81930920983156e-09,5.26813967002820e-07,6.78082021954823e-07,7.46691969030421e-05;8.57482946186385e-07,1.81510340054009e-08,0,6.92721481828138e-07,8.43708952248447e-09,1.49205511181957e-08,9.80809299832946e-09,7.76871476674898e-09,1.78528551943935e-08,1.49912844120936e-08;6.09151361115910e-07,7.28483536472595e-08,8.71600850373676e-09,0,5.72538249784979e-09,9.10926339747863e-09,7.35374317873584e-09,6.37803448291528e-09,9.14427759717427e-09,8.19965075697278e-09;3.24438672490018e-08,7.43465220379423e-09,1.19030417501578e-08,1.01009175091963e-08,0,2.56612975355900e-08,1.75217859349415e-

08,1.20545690348571e-08,2.92914259528454e-08,3.12441105821958e-08;2.32221437492219e-08,7.27664478165241e-09,2.22205993844548e-08,2.49661408399317e-08,8.59196271876430e-09,0,8.13988702317737e-08,4.17074105688040e-08,1.31298868836011e-06,1.18621333626639e-08;9.87861981906184e-07,8.40113989731698e-09,7.78168816425583e-07,6.16976385776373e-07,1.57829344227559e-08,1.25150139816524e-06,0,6.86716323538438e-07,2.11752624409949e-08,2.61470713070418e-08;1.14436394512247e-08,4.45315944363279e-07,5.92824695410259e-09,6.16586268074336e-09,1.66907253195717e-08,7.72194508425768e-09,7.32507971437749e-07,0,8.48563890650514e-09,9.52132826695808e-09;1.37756272482484e-08,1.21003050565923e-08,1.70324268744524e-08,9.98636791208819e-09,9.95206354680109e-07,4.70107576999125e-08,1.75636222140032e-08,1.17679246379307e-08,0,2.19532015837162e-06;1.16397132726604e-08,4.78582024518471e-07,6.11785385613407e-09,6.07514606705879e-09,2.63150817719086e-08,8.34794842309494e-09,7.57426787028384e-09,6.55930520650442e-07,9.60279545114080e-07,0];
distance =
[5.38747249667784;11.8079399907625;4.49402872220082;7.93182048040550;2.80802018900614;2.24958536516573;5.44764182364908;7.10421877473502;1.35547932382947;4.19814945915000];

Pb 6

int = [0,7.70826444355871e-09,2.40701429328699e-08,1.03962116618776e-08,6.87686885944936e-09,7.45897038231801e-09,1.03714996596822e-08,8.34750009984923e-09,1.72451851885549e-08,8.49174392001284e-09;1.64283264480821e-08,0,1.84868691369934e-08,9.17719541242285e-09,6.29120887721069e-09,6.99428248959224e-09,1.09976004016085e-08,7.47040893230235e-09,1.49349318098625e-08,7.59127053669771e-09;1.06798149193043e-08,7.17247156509687e-07,0,1.73751140960017e-08,3.59275942763852e-08,7.43321276388501e-07,4.86751596773383e-09,3.99016818015954e-08,8.15107364944653e-05,5.22387124524286e-08;1.74310263659723e-08,5.30235313731841e-09,1.05072384926056e-08,0,7.65153439503336e-09,5.24454686808282e-09,6.01433064731438e-09,7.58763309843528e-09,8.30571431694586e-09,8.34878654393373e-07;1.12595993551329e-08,9.13900451559611e-09,1.11989257903725e-08,1.48785927110334e-08,0,9.57053756676972e-09,5.45098852473925e-09,1.16500216551354e-07,1.01727665237879e-08,3.48770356279336e-08;1.63966990271256e-08,6.18153104890648e-07,1.06213258730188e-08,7.73902742789677e-08,2.31241759643889e-08,0,5.06317891223290e-07,1.65983084633908e-06,8.54750155728941e-09,2.50344736303480e-08;1.40392078248379e-08,5.68749513428023e-09,9.18978902034641e-07,3.83113355862594e-08,2.14137524499665e-08,5.70124489582356e-07,0,1.35982940312145e-08,7.61167194335533e-09,1.88145244081922e-08;3.74455438379638e-08,6.58133087413267e-09,1.45250458904085e-08,3.70031205614060e-08,1.24560649571052e-08,6.51539825282310e-07,5.99749393141629e-07,0,1.05443304342143e-08,1.54440069230234e-06;9.53181274431869e-09,5.63681320220924e-07,7.51743183936917e-05,1.73050142472894e-08,8.85848680439796e-08,5.72321735652902e-05,4.21855316711350e-05,1.61303647729731e-08,0,2.20395167529891e-08;1.85484257361350e-08,1.08269181772089e-08,2.21753582471051e-08,1.68342247995658e-08,1.21068791222497e-08,1.08860144077716e-06,6.95915138600117e-07,2.44790797520325e-08,1.74922633793015e-08,0];
distance =
[2.27506757336226;6.53492267941922;5.03670527763028;3.75090502222700;2.64996366087009;7.48105478213769;9.85739751022665;3.67314579273941;7.05571152161532;2.28046411965987];

Pb 7

```
int = [0,6.64429008867803e-09,6.61392319293182e-07,1.37100785333524e-
08,1.22660723617525e-08,4.12890939252707e-09,6.33633165968102e-
07,4.93574772304200e-09,5.84458264356536e-09,1.48942472708993e-
08;9.24448050052720e-09,0,1.70107415654680e-08,1.34533155561760e-
06,1.79238542072162e-08,8.68419894455042e-09,9.42731479179330e-
09,1.23403736475623e-06,2.36046716503492e-08,1.00596482372513e-
08;5.13309352730303e-09,2.53637949864473e-08,0,8.72291521430343e-
09,1.47173314444545e-08,4.75791067024672e-07,5.20276866301069e-
09,5.60550562909830e-09,9.04909992379622e-09,2.10870096917931e-
08;8.74110035542535e-09,1.04619794797917e-08,1.57312235476843e-
08,0,1.59240745420733e-08,9.08395799843071e-09,8.80547921311277e-
07,1.29554698651164e-08,3.05335645951130e-08,9.55358400242011e-
09;6.08362074739648e-09,7.66563330142923e-09,8.68992973748942e-
09,6.13489051639543e-09,0,1.37110022886032e-08,6.11590717586016e-
09,1.34218802181097e-08,1.93775312255245e-08,5.65351133266273e-
09;4.80775049931478e-09,6.99767594347273e-08,6.26764900427317e-
09,6.88836698508908e-09,1.02035788296494e-08,0,4.86070405791861e-
09,6.06809348657715e-09,1.12886941116937e-08,1.02368777042119e-
08;9.63925654836442e-07,7.94960665940540e-09,1.93974171331519e-
06,9.76094062525448e-09,1.06158222950455e-08,1.30937230236042e-
08,0,2.33158338681225e-08,2.52788578863192e-08,7.23736914848689e-
09;1.20841395380843e-08,5.73509828094233e-09,2.01862779479652e-
08,8.00476001951675e-09,7.46549694920364e-07,1.96086128408944e-
08,1.19597176445162e-06,0,1.16295334621024e-08,5.59194663499466e-
09;5.26951199724845e-09,1.72010526347706e-08,7.25356146266725e-
09,6.70134673037943e-07,9.29430770901311e-09,6.91878218184267e-
07,5.32135388257659e-09,7.80628494978902e-09,0,7.94365630607473e-
09;7.67912997477200e-09,8.08093363686850e-07,8.93131720144670e-
09,2.61909748081862e-08,2.38337720867467e-08,4.96548668698207e-
09,7.84694501594322e-09,6.16455078828529e-09,7.68460681269679e-09,0];
distance =
[7.52562097235854;4.65683161999227;7.31115075657693;3.93787532906356;6.6945
7312871127;8.80058184965478;4.80213417708596;0.737546014836232;2.5361949134
2256;2.35635471680572];
```

Pb 8

```
int = [0,1.11403090353748e-08,1.47772597965912e-08,7.96985828305061e-
09,8.80748154802204e-07,9.77253555433380e-09,1.23375827199832e-
08,1.18973525153399e-08,9.42770638862058e-09,1.34541217746927e-
08;7.91370642494729e-09,0,1.38111589903040e-06,5.42288678958087e-
09,1.21508501928773e-08,8.08621764425568e-09,6.09813988577159e-
07,6.53566194946779e-07,1.14665426212858e-08,2.51347661576666e-
08;8.99083552552424e-09,5.77976470396549e-07,0,6.44575689943954e-
09,5.11580402661103e-09,1.03182236781182e-08,1.28759652880942e-
08,8.65691608981235e-09,5.26918827291199e-07,7.36707363986340e-
07;6.42620148840442e-09,7.81223774997040e-09,9.16893122812977e-
09,0,6.46972650478197e-09,7.06363006441685e-09,6.07425263550089e-
08,2.03019076895461e-08,6.94564106599706e-09,7.17564322119786e-
09;8.62580146340993e-07,1.10788370788597e-08,1.46695046849534e-
08,8.06455528303203e-09,0,9.68945104363918e-09,1.26286198881064e-
08,1.21274060839007e-08,9.37875239301751e-09,1.31274690520061e-
08;7.03872441840939e-09,4.65794045381798e-09,5.12228827489127e-
07,5.60268747377564e-09,4.13099254691311e-07,0,9.86733422729711e-
```

```
09,6.91757863018010e-09,4.32356987907162e-09,5.52318588217699e-
07;5.98612602136472e-09,2.10450386546232e-08,1.96352065150190e-
08,5.92025947078692e-07,2.46957481334741e-08,6.14520270061567e-
09,0,6.95002442520878e-07,2.06893044153111e-08,1.22725662218679e-
08;7.08591566260482e-09,1.81037434510597e-08,3.01531639314571e-
08,8.77023563305794e-09,1.26787219142038e-08,7.61489334887439e-
09,1.05617884928001e-06,0,1.39509658351375e-08,1.24051978998460e-
08;5.75566304122310e-09,2.52188966736078e-08,3.84874497706105e-
08,1.08071694605883e-06,1.49112569792113e-06,6.10945110377708e-
09,1.04541955473878e-08,1.50920219414275e-08,0,9.05597752257420e-
09;7.72219715815013e-09,4.98663528308442e-09,5.53216728343694e-
07,5.84726534309861e-09,4.44026930840580e-09,8.53390664645119e-
07,1.07693128654120e-08,7.40813290675566e-09,4.60511321557937e-09,0];
distance =
[5.38692129950818;3.80380163552944;7.07825391117037;4.20346666668679;5.3577
7358825784;6.10768328790006;7.79415632975217;3.74044914962709;2.60588865712
981;7.75118045008909];
```

Pb 9

```
int = [0,5.38582019418721e-07,1.91378639155847e-08,5.66638365208593e-
09,6.23387835738893e-09,2.82268459655771e-08,6.97454085216619e-
09,1.04997931379195e-08,1.51990597129326e-08,4.98923552462738e-
07;1.33599349529548e-08,0,1.90956333805326e-08,6.19995609894049e-
09,1.76862062253901e-08,7.18805904826430e-09,1.83657532827338e-
08,8.21121369470471e-09,2.55252948376989e-08,1.04578031557935e-
08;3.77639114188569e-08,2.02559162120207e-08,0,7.66526682969447e-
09,1.91712996357853e-08,5.76800847243286e-09,1.93231730435374e-
07,8.37255468341440e-09,1.21578086021507e-08,1.46021624220184e-
08;1.66269828887063e-08,1.51836275116215e-08,6.55127072744331e-
07,0,2.10567247683677e-08,4.18835792281790e-09,1.42356630798283e-
08,5.08911938997087e-05,7.16645204162909e-09,1.94064207378113e-
07;1.47764396857901e-08,1.09267535618740e-08,1.42019784034808e-
08,5.91742768986308e-09,0,6.38049274602004e-09,2.00823475649692e-
08,7.31226597009460e-09,1.77634654777502e-08,1.25751870055538e-
08;1.28911325686803e-08,1.39808914266091e-08,7.71085084801471e-
09,1.90169284595189e-08,7.54811360183725e-09,0,1.23888854550237e-
08,1.07675143385756e-08,7.34093506234657e-09,7.60533347329842e-
09;7.02958129156438e-07,6.48638815296442e-09,1.42155717895964e-
08,8.86763680122371e-09,6.12779425731561e-07,1.17097283707461e-
08,0,3.47590514634723e-08,1.12571649271078e-08,5.23959084075849e-
07;1.41112835119927e-08,1.22597122165066e-08,1.30499949097948e-
08,1.17702380077299e-08,9.32196275116690e-09,6.90117413137317e-
09,1.69032003516792e-08,0,1.18611493426198e-08,8.07772612771395e-
09;7.90946558135843e-07,6.84886128646368e-09,8.38690197088636e-
08,6.28960747775619e-09,8.29987653088282e-09,1.31996508957645e-
08,9.59762084342187e-09,1.11986395399069e-08,0,6.32557910472884e-
09;5.54179119013719e-09,4.90217280848822e-09,8.29549808083161e-
09,3.71078550934052e-09,7.47361050478673e-07,6.30759610651877e-
09,6.13253591139950e-09,4.74186850507367e-09,9.54425060380354e-09,0];
distance =
[7.76561664218851;4.66734410489376;4.14338942572860;8.90855338205092;1.8368
7231571063;9.17286256498559;5.89915094296738;3.04993545201891;1.31302133561
636;8.22078892498461];
```

Pb 10

```
int = [0,4.69136216531816e-08,4.05368536551826e-08,2.42229197041071e-
08,4.30117012622143e-08,1.45487124294405e-06,1.36849782978436e-
07,2.39236622155106e-08,6.32048472703157e-08,2.80292072465969e-
08;1.02591353366569e-07,0,1.00692801995153e-06,2.20589902051293e-
08,4.37952887298844e-08,1.53545337349926e-06,3.62945948206195e-
08,2.20948726139747e-08,5.01040223009936e-08,4.04596741633729e-
08;4.27198208840961e-08,2.20617983537960e-08,0,5.34754374134073e-
08,7.68049255187721e-08,1.51882881918736e-07,1.06605004117539e-
06,5.58203251426707e-08,2.10978644640041e-08,8.51667106231120e-
08;4.62841178505944e-07,1.23515802186470e-06,2.70133039286520e-
08,0,3.95824617096734e-06,5.80005609772485e-08,3.94954547190676e-
08,1.31094648806269e-06,3.78766742865472e-08,5.70879167610972e-
08;2.89169845532095e-08,6.60286163389878e-07,1.55611007125362e-
07,2.43010827522690e-08,0,1.14486261082034e-06,7.22350973958539e-
08,2.37172853220918e-08,1.35038134579122e-06,2.03659768485293e-
08;7.40906872460014e-08,1.87657615092991e-06,3.82437753438144e-
08,2.61776533598276e-08,5.20636025204103e-08,0,9.06138445385708e-
08,2.58869329432676e-08,5.60001964414587e-08,3.18083231077337e-
08;4.75118406693038e-08,2.11444852116770e-08,2.13810578942143e-
08,4.17077705478161e-08,7.36753036717789e-08,1.06809623967230e-
07,0,4.30787989068283e-08,2.19963005449238e-08,1.55062474091733e-
07;2.71324779582594e-08,2.69830019624498e-08,2.97100180938987e-
08,1.83278115205955e-07,4.16452537559954e-08,4.90766800984394e-
08,2.41707884681751e-08,0,1.80909642263304e-08,2.92880154859087e-
08;3.28539600394577e-08,7.70410390704725e-08,9.61234851253390e-
08,3.58953304383660e-08,1.80485033179277e-06,3.59837622785639e-
08,4.94580882832871e-08,3.45876880964949e-08,0,2.47021268181847e-
08;2.78453520478152e-08,1.52759472041858e-07,4.23404756755143e-
07,2.81944690943455e-08,1.38586705613112e-06,2.80210967328670e-
08,5.11027858203189e-08,2.73366028239341e-08,2.59789585797325e-08,0];
distance =
[3.31145544462723;7.78801566298734;8.38699760057524;6.36755180428707;6.3136
0064274941;4.97509959417465;7.86433016320318;1.27869677618441;7.20088277111
542;9.06987123710875];
```