Decision Engineering
Homework Assignment 4

Shashank Iyengar, Johann Koshy

The traveling salesman problem is a problem in graph theory requiring the most efficient (i.e., least total distance) Hamiltonian cycle a salesman can take through each of *n* cities [1].

Suppose *n* is the number of cities the salesman needs to visit with given condition that he must travel through every city only once and finally return to his home city, there are $(n-1)!/2$ possible tour combinations that he can use. The objective is to find the shortest path possible. For a perfectly optimal solution, the brute force method can be used. But since the number of routes are extremely high, it results in excessive computation time which is not preferred.

To circumvent the issue of computation power, a near optimal solution using dynamic methods is widely accepted since there is minimal loss in accuracy with respect to exponential decrease in computation time. For the given problem statement, the genetic algorithm (GA) approach was used to reliably determine the near optimal shortest path.

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution [2].

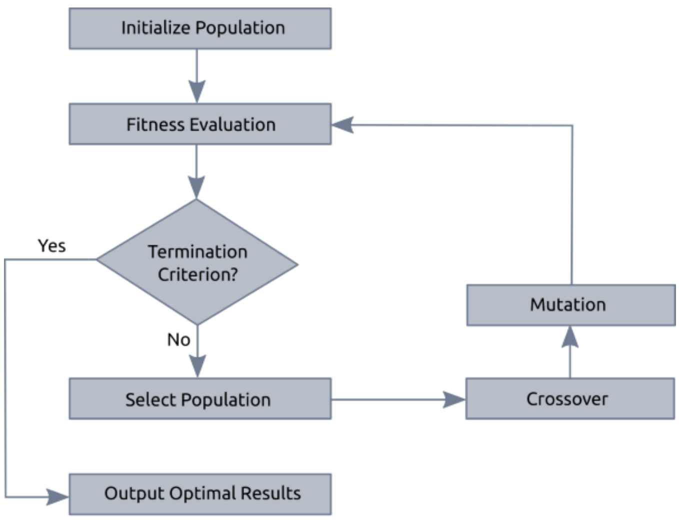

Figure 1: Basic Structure of Genetic Algorithm

Pseudo Code:

Start Program

Input prompt for number of cities
Randomly create cities scattered in grid of 100x100 (x,y) units
Construct Initial Population
For
Randomly select parents for crossover
Produce children from said crossover (PMX)
At random intervals, introduce few entirely new parents into population pool (Gene Flow)
Mutate children with probability increasing to better represent diversity
Add children to population
Compute fitness and reduce population
End
Track global minimum paths
Plot results of optimal solution found from GA

End Program

Observations

- The Partially Mapped Crossover (PMX) method was used to create offspring for every generation.
- Population size is equal to the number of cities.
- At random generations, entirely new parents were introduced into population pool (akin to an immigrant from a different population). This is known as Gene Flow. It transfers genetic variation from one population to another and ensures genetic diversity.
- The objective behind using Gene Flow was to better simulate a more realistic formulation of the genetic algorithm and to also determine the robustness of the code logic since addition of a totally new and different parent could be considered analogous to noise in dynamic systems.
- It was observed that with the inclusion of Gene Flow, the number of generations the algorithm took to reach convergence increased. Quantifying, in one case, for 50 cities, it took 640 generations to converge on an optimal path without GF. For the same 50 cities and the addition of GF, it took 960 generations to converge.
- Probability of mutation was increased with increase in generations
- Crossover sub-tour size was increased up to 500 generations and randomized beyond that.
- For 50 cities, the shortest path obtained was approximately 865.8 units
- This could vary for the same number of cities since they could be randomly scattered for each new run of the program.

- The processing time for 50 cities and 1500 generation was 16 minutes. Computing specs: Intel Core i5 – 5200U @ 2.2GHz (4 cores), 8GB RAM, 11400MB page file.

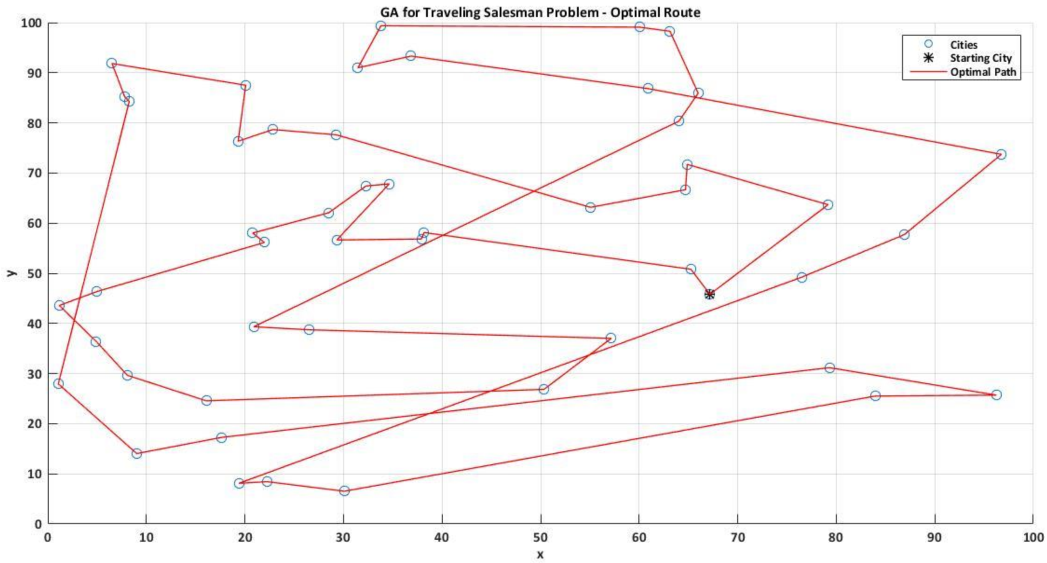

Figure 2: Optimal Route using GA

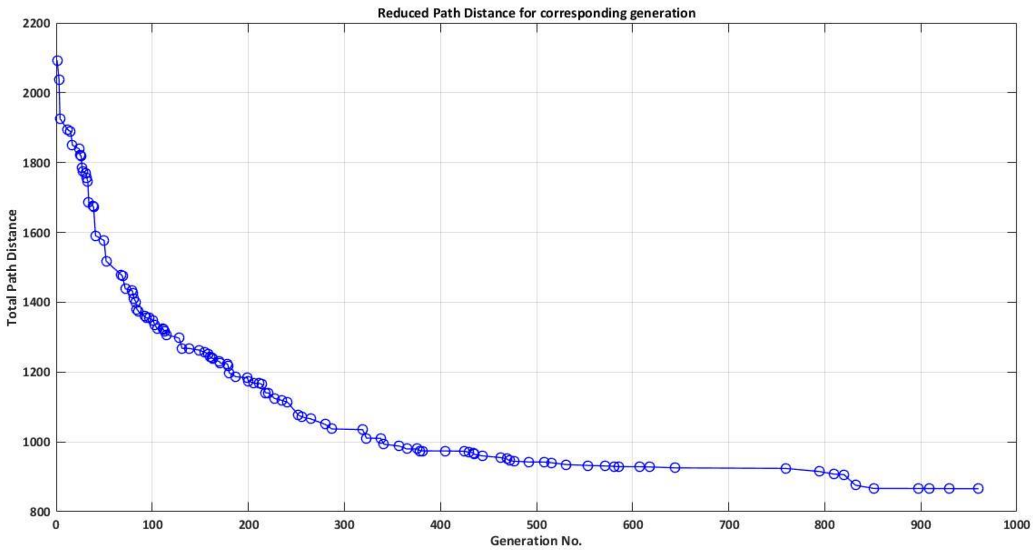

Figure 3: Reduction in path length for successive generations

Conclusion

Genetic Algorithm is a highly accurate technique to solve optimization problems when computation time and available data are key factors. The only required knowledge for using GA is the work flow mentioned in the pseudo code; large amounts of statistical data are not required to find the optimal solution.

References

[1] Weisstein, Eric W. *"Traveling Salesman Problem."* From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/TravelingSalesmanProblem.html

[2] MATLAB Documentation. https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

[3] P. LARRANAGA, C.M.H. KUIJPERS, R.H. MURGA, I. INZA and ˜ S. DIZDAREVIC. *"Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators."* Artificial Intelligence Review 13: 129–170, 1999

APPENDIX

```
%%Homework Assignment 4
%%Genetic algorithm - Shashank Iyengar, Johann Koshy

clear all
close all
clc
set(0,'DefaultAxesFontName', 'Calibri')
set(0,'DefaultAxesFontSize', 11)
set(0,'defaultlinelinewidth',1)
set(0,'DefaultLineMarkerSize', 8)
set(0,'defaultAxesFontWeight','bold')

prompt = 'Enter number of cities: ';
N = input(prompt);

% Create Cities
x = rand(1,N)*100
y = rand(1,N)*100
z = 1;                          % To track global minimum
gen_count = [];                 % To track generation at which min is
obtained

% Selecting first population
if mod(N,2)==0                  % Controlling population length
    pop_length = N;
else
    pop_length = N-1;
end
pop = [];
for i=1:pop_length
    pop(i,2:N) = randperm(N-1);
    for j=1:N
        if pop(i,j)==1
            pop(i,j) = N;
        end
    end
end
for i=1:pop_length
    pop(i,1) = 1;               % Starting and ending with the same city
    pop(i,N+1) = 1;
end
pop
pop_length = length(pop(:,1));
parent_pop = pop;
min_global = 10000;

for gen=1:1500                  % Number of generations

% Creating Random order before crossover for introducing randomness in
crossover
l = randperm(pop_length,pop_length);   % Random order for new population
pop_swap = pop;
for i=1:length(pop(:,1))
    pop([i l(i)],:) = pop_swap([l(i) i],:);
```

```
end
if gen==250 && gen==500 && gen==750    % Gene Flow - Introducing four new
individuals for genetic diversity at gen 500
    for i=1:round(N/5)
        pop(i,2:N) = randperm(N-1);
        for j=1:N
            if pop(i,j)==1
                pop(i,j) = N;
            end
        end
    end
    for i=1:round(N/5)
        pop(i,1) = 1;
        pop(i,N+1) = 1;
    end
end
pop;
gen;


% Crossover PMX
% Determining selection for index for slicing
if gen<300
    start_slice = round(0.35*(N+1));
    stop_slice = round(0.65*(N+1));
else if gen<500
        start_slice = round(0.25*(N+1));
        stop_slice = round(0.75*(N+1));
    else
        start_slice = round((0.5-0.5*rand(1))*(N+1))+1;
        stop_slice = round((0.5+0.5*rand(1))*(N+1))-1;
    end
end
dummy=[];
r=0;

for i=1:2:(length(pop(:,1)))
   dummy(i,:) = pop(i,start_slice:stop_slice);
   dummy(i+1,:) = pop(i+1,start_slice:stop_slice);
   o = stop_slice-start_slice+1;
   % To check if numbers are repeated and swap them
   for j=1:o
       for k=1:o
            if dummy(i,j)==dummy(i+1,k)  % To check if the slice has same
numbers swapped
                dummy(i,j) = 0;
                dummy(i+1,k) = 0;
            end
       end
   end
   n = 1;
   for j=1:o
       if dummy(i,j)~=0                    %% To check if the slice has
same numbers swapped
           for w=1:N+1                     %% To search the population for
repeating elements from crossover chromosomes
```

```
                if dummy(i,j)==pop(i+1,w)   %% To search the population for
repeating elements from crossover chromosomes
                    for u=n:o
                        if dummy(i+1,u)~=0 && r==0 %% To search the
population for repeating elements from crossover chromosomes
                            pop(i+1,w)=dummy(i+1,u);
                            r=1;            %% To exit search after
replacing the city
                            n=u+1;          %% To start searching dummy
variable from that element for next iteration
                            if w==1          %% If first element is swapped
last element should be swapped too
                               pop(i+1,N+1)=dummy(i+1,u);
                            end
                        end
                    end
                    r=0;
                end
            end
        end
    end

    % Swapping for the second child
    n=1;
       for j=1:o
        if dummy(i+1,j)~=0                  %% To check if the slice has
same numbers swapped
            for w=1:N+1                     %% To search the population for
repeating elements from crossover chromosomes
                if dummy(i+1,j)==pop(i,w)   %% To search the population for
repeating elements from crossover chromosomes
                    for u=n:o
                        if dummy(i,u)~=0 && r==0 %% To search the
population for repeating elements from crossover chromosomes
                            pop(i,w)=dummy(i,u);
                            r=1;            %% To exit search after
replacing the city
                            n=u+1;          %% To start searching dummy
variable from that element for next iteration
                            if w==1         %% If first element is swapped
last element should be swapped too
                               pop(i,N+1)=dummy(i,u);
                            end
                        end
                    end
                    r=0;
                end
            end
        end
    end
    pop([i (i+1)],start_slice:stop_slice)=pop([(i+1)
i],start_slice:stop_slice);
end
pop;


% Mutation
```

```
if gen<500
    P_mutation = 0.2;           % Probability of Mutation
else if gen<1000
    P_mutation = 0.3;           % Probability of Mutation
    else
    P_mutation = 0.6;           % Probability of Mutation
    end
end

for i=1:pop_length
    p = rand(1);
    if p<P_mutation
        i;
        swap_mutation_1 = randi([2 N]);
        swap_mutation_2 = randi([2 N]);
        if swap_mutation_2==swap_mutation_2
            swap_mutation_2 = randi([2 N]);
        end
        pop(i,[swap_mutation_1 swap_mutation_2])=pop(i,[swap_mutation_2
swap_mutation_1]);
    end
end

% Adding Children created to population
pop = [parent_pop;pop];


% Computing Fitness for each variable
sum_dist = [];
X = [];
for i=1:length(pop(:,1))
    for j=1:N
        a = pop(i,j); %%City 1
        b = pop(i,j+1);%% City2
        X = [x(a),y(a);x(b),y(b)];
        dist(j) = pdist(X,'euclidean');
    end
    sum_dist(i) = sum(dist);
    [min_dist best_chromosome] = min(sum_dist); % Indexing min from
population
end
totalsum_dist = sum(sum_dist);

for i=1:(length(pop(:,1)))
    fitness(i) = sum_dist(i)/totalsum_dist;
end
fitness;

% Tracking global minimum and path
if min_dist<min_global
    min_globalplot(z) = min_dist;
    best_route = pop(best_chromosome,:);
    min_global = min_dist;
    gen_count(z) = gen;
    z = z+1;
end
newpop = [];
```

```
% Ordering matrix according to fitness
for j=1:length(pop(:,1))/2
    [e i] = min(fitness(1:length(pop(:,1))));
    fitness(i) = 1;
    newpop(j,:) = pop(i,:);
end

% Assigning next Generation
pop = newpop;
parent_pop = pop;
% Plotting best route
x1 = [];
y1 = [];
for i=1:N+1
x1(i) = x(best_route(i));
y1(i) = y(best_route(i));
hold on
end

end

best_route
scatter(x,y)
hold on
plot(x(1),y(1),'*k')            % Starting City
hold on
plot(x1,y1,'-r')                % Optimal Route
grid on
title('GA for Traveling Salesman Problem - Optimal Route')
xlabel('x')
ylabel('y')
legend('Cities','Starting City','Optimal Path')
xlim([0 100])
ylim([0 100])
figure
plot(gen_count,min_globalplot,'-ob')
grid on
title('Reduced Path Distance for corresponding generation')
xlabel('Generation No.')
ylabel('Total Path Distance')
min_globalplot                  % Trend of minimums
gen_count                       % Generation at which new minimum is found
```

Output:

```
Enter number of cities: 50

x =

  Columns 1 through 7

   67.1814   22.8285   96.7526   16.0704   60.9605    4.9109   37.9075

  Columns 8 through 14
```

```
   50.3520    1.0650    1.1600    7.8019   32.2924   19.3869   33.8020

  Columns 15 through 21

   29.2698   79.3063   96.2489   36.8567   86.9033   22.2401    6.5160

  Columns 22 through 28

   64.7042    5.0090   65.2422   64.0298   64.8768   55.0670   20.0843

  Columns 29 through 35

   79.1713    8.0688   17.6272   57.1676   26.4749   29.3244   28.5033

  Columns 36 through 42

   34.6450   20.8952   31.4668   20.7895   38.1761   60.0394   63.1217

  Columns 43 through 49

   21.9713   19.3133   83.9364   76.4943   30.0888    9.0195    8.2614

  Column 50

   65.9928


y =

  Columns 1 through 7

   45.8048   78.6711   73.7200   24.5612   86.8186   36.4049   56.8454

  Columns 8 through 14

   26.8502   27.9330   43.5742   85.2103   67.4164    8.1114   99.3645

  Columns 15 through 21

   77.6148   31.1698   25.6988   93.3155   57.7046    8.4406   91.8380

  Columns 22 through 28

   66.6691   46.3763   50.8101   80.3482   71.7204   63.1534   87.5029

  Columns 29 through 35

   63.6839   29.6586   17.2652   37.0211   38.7323   56.6451   62.0757

  Columns 36 through 42

   67.8398   39.3311   90.9927   58.0307   58.1239   99.0845   98.2729

  Columns 43 through 49

   56.1366   76.3441   25.5064   49.2250    6.5305   14.0361   84.2601
```

Column 50

85.9126

pop =

Columns 1 through 12

1	27	21	42	40	17	15	8	16	48	6	26
1	42	5	18	27	4	19	6	37	25	9	43
1	5	13	35	42	29	12	44	17	23	38	36
1	6	14	16	50	18	48	45	22	31	49	46
1	24	17	4	38	46	35	18	29	47	15	7
1	44	23	10	15	13	25	24	49	20	2	16
1	21	24	44	28	31	29	6	8	50	20	33
1	6	8	19	47	42	43	22	16	45	33	35
1	31	23	45	4	29	37	20	34	21	26	5
1	18	6	3	49	9	10	36	47	31	26	48
1	32	40	16	49	2	5	33	24	38	37	27
1	28	38	45	41	20	39	32	34	47	37	3
1	33	42	17	29	14	43	50	31	9	47	19
1	37	47	49	22	27	26	10	2	7	45	31
1	38	29	43	20	22	37	24	23	41	9	45
1	32	27	6	46	7	36	19	29	14	15	8
1	18	25	11	15	5	23	39	31	7	24	40
1	17	6	49	12	16	34	10	47	35	2	30
1	31	43	39	12	10	2	26	32	25	6	21
1	41	18	13	14	50	16	30	28	3	31	44
1	29	14	6	49	13	8	41	5	36	47	2
1	42	50	31	36	40	43	22	34	45	38	33
1	34	32	43	21	30	23	3	29	14	50	28
1	2	7	22	47	28	36	13	41	50	39	38
1	8	48	6	43	23	42	17	29	4	39	19
1	17	12	9	27	50	25	44	8	10	26	42
1	22	47	7	28	16	34	18	19	49	10	21
1	17	28	44	43	24	5	21	4	40	31	46
1	14	31	35	20	11	42	7	17	44	45	22
1	13	48	15	4	22	31	26	23	17	7	20
1	42	43	20	3	44	10	8	21	37	46	13
1	36	6	24	13	14	23	49	17	34	50	8
1	27	28	25	44	19	9	13	50	3	5	22
1	25	32	8	38	17	18	15	49	4	50	12
1	50	33	19	36	12	38	37	16	20	46	5
1	26	41	12	21	27	45	31	8	37	23	10
1	35	8	47	38	19	33	13	24	14	43	11
1	24	4	25	32	41	38	45	49	48	35	44
1	9	35	6	37	5	41	45	34	27	47	29
1	29	32	43	34	7	18	12	8	42	4	44
1	19	11	38	49	30	20	44	31	45	23	2
1	21	43	19	29	44	24	33	37	4	10	15
1	36	20	43	21	34	28	3	33	27	32	12
1	45	20	41	34	26	50	37	40	7	14	24
1	16	7	27	29	41	13	36	49	42	4	20
1	5	34	7	17	6	15	11	14	49	46	31
1	19	49	48	50	2	27	44	14	42	17	6
1	28	25	45	22	35	14	21	2	17	39	27
1	15	6	28	16	27	44	9	33	17	22	50

1	44	17	29	32	23	42	4	50	35	46	12

Columns 13 through 24

32	18	49	19	9	33	30	11	37	46	4	12
30	44	33	20	22	2	49	23	29	17	50	34
9	28	49	22	48	3	20	4	50	34	33	30
7	13	24	23	38	32	4	39	29	47	27	30
13	25	5	22	6	19	50	37	45	10	16	41
19	34	47	3	48	14	9	35	37	45	17	28
19	41	43	35	3	9	5	42	22	25	37	2
20	40	50	18	24	28	31	39	32	14	13	17
36	10	8	14	33	24	47	43	17	11	30	25
44	5	50	13	7	33	40	19	28	23	39	37
15	14	44	36	6	43	3	23	47	18	28	13
44	42	19	8	27	15	46	7	17	36	43	16
12	20	49	26	27	28	6	15	16	45	46	4
21	50	25	11	40	42	18	43	33	38	3	41
36	46	35	6	19	28	50	13	18	31	15	5
41	50	18	37	39	45	47	4	13	16	28	34
6	37	48	30	19	29	21	16	34	50	38	26
8	46	24	9	14	20	27	23	31	22	33	39
5	4	41	20	47	37	15	22	23	48	45	34
24	7	46	6	47	2	40	48	10	20	32	39
28	21	11	37	3	35	42	23	25	43	27	7
28	29	47	35	8	24	13	25	19	3	27	7
47	5	6	39	22	46	40	17	12	48	9	44
4	17	37	9	5	35	20	24	42	21	3	25
14	15	44	45	13	24	37	22	25	31	46	7
23	36	5	11	49	48	35	14	39	40	18	4
38	5	43	41	40	3	39	20	26	50	36	25
36	48	3	35	14	34	8	6	33	20	2	42
28	23	12	47	5	25	49	32	33	9	6	30
47	37	49	32	41	19	38	12	40	29	2	45
28	2	38	14	19	7	39	50	4	33	16	41
15	27	40	16	28	48	38	12	26	37	7	32
18	39	21	4	8	23	42	10	14	2	31	37
13	14	33	36	28	22	39	20	41	7	35	16
26	4	9	24	49	34	35	31	7	21	11	3
24	4	14	3	5	39	29	34	20	43	22	35
50	15	21	39	23	41	49	32	7	31	30	6
39	42	34	37	10	29	27	31	3	22	9	21
15	43	24	50	23	46	30	28	22	14	32	39
15	23	31	47	17	21	11	33	38	22	50	35
17	16	48	33	47	12	34	10	24	46	25	39
46	47	45	13	18	50	32	22	23	16	38	12
37	25	26	45	41	24	23	14	8	39	10	49
16	2	22	3	39	43	35	6	8	29	10	48
46	50	30	37	32	43	12	2	21	47	31	10
8	33	36	29	40	16	35	37	32	43	25	39
5	12	8	24	16	40	23	18	13	32	47	43
44	7	20	42	40	43	33	12	46	50	41	29
8	39	40	31	24	37	26	21	12	30	38	4
21	39	36	6	14	19	25	26	9	3	20	18

Columns 25 through 36

24	2	3	13	31	45	25	22	36	23	29	14

3	38	39	24	31	16	12	35	26	48	32	45
43	21	31	27	2	40	26	32	6	11	18	25
40	41	42	20	17	10	36	33	28	5	8	15
30	34	28	32	8	44	36	3	42	2	40	12
30	32	4	40	18	46	38	22	31	6	5	39
10	38	27	48	39	49	14	13	15	36	40	32
9	48	10	12	23	44	49	34	4	5	11	27
41	6	3	9	15	50	38	22	19	42	13	46
21	8	27	17	2	4	29	24	42	22	41	35
8	19	17	30	48	12	25	7	11	10	34	4
24	4	50	9	2	30	22	14	18	10	23	6
18	41	5	38	23	3	7	39	34	13	36	25
16	14	5	15	46	19	34	30	13	23	8	32
12	47	14	11	26	16	10	8	30	17	42	4
12	30	35	40	38	33	9	49	10	25	23	42
32	12	20	3	14	13	36	4	28	2	42	9
15	4	19	21	44	25	26	40	32	45	36	29
44	24	33	16	8	14	11	35	7	27	9	30
19	33	22	26	25	5	27	37	38	8	43	4
40	31	50	16	19	9	38	44	12	45	32	33
20	18	30	15	26	48	11	5	44	6	17	49
26	38	37	4	19	20	36	24	18	2	45	7
10	29	34	16	33	40	31	32	49	18	46	26
30	20	47	41	38	16	2	11	34	32	35	40
47	33	22	6	41	34	31	32	45	15	24	30
32	46	33	13	35	24	14	27	31	48	8	29
11	27	15	49	18	38	12	26	7	50	13	16
39	8	21	48	38	16	27	46	29	24	43	19
50	6	16	39	33	27	14	9	36	46	28	18
25	18	12	11	36	24	34	22	49	40	26	23
2	20	35	4	21	33	47	29	31	46	43	22
45	12	41	29	26	16	38	20	35	43	24	30
29	31	9	34	42	45	43	37	5	48	10	26
25	22	14	29	45	18	48	42	43	40	8	6
44	40	15	19	49	30	36	9	11	6	50	28
29	36	10	42	3	18	40	20	16	4	28	12
17	33	47	20	28	50	8	23	7	11	26	36
38	7	49	48	16	44	40	26	4	8	36	33
14	30	36	5	9	39	40	13	10	20	49	6
35	8	13	5	22	7	4	3	15	18	37	28
48	20	9	5	14	49	41	39	34	26	27	28
30	6	29	18	46	22	9	38	47	11	5	50
42	28	47	5	30	4	21	32	44	46	19	18
25	19	34	26	22	15	3	45	18	40	23	33
4	20	42	50	44	3	21	9	45	10	18	12
34	41	4	33	45	9	46	7	10	15	30	37
49	19	10	9	36	47	23	16	15	31	4	6
10	41	2	29	36	47	5	13	3	34	43	14
41	24	2	49	48	30	47	10	15	13	16	43

Columns 37 through 48

47	20	10	28	39	38	34	43	35	44	7	50
13	21	7	15	28	8	47	14	40	36	11	41
37	47	24	14	10	41	45	39	7	46	15	16
21	25	34	11	44	37	2	26	19	9	12	35
26	27	20	23	9	49	11	39	43	14	33	31
36	50	41	21	8	27	33	42	11	29	26	43

30	23	47	26	12	34	45	16	46	7	4	18
2	41	29	7	30	3	38	21	36	25	15	46
32	16	39	7	18	27	49	28	2	48	35	40
25	32	16	34	20	12	46	15	14	11	45	38
35	21	31	41	46	20	22	26	50	29	42	45
35	5	11	12	29	26	13	40	21	49	31	48
11	22	40	48	10	8	2	44	21	30	37	24
35	24	4	9	36	29	28	12	17	44	6	20
40	32	25	34	48	39	33	2	21	7	3	44
31	24	43	3	17	20	26	21	5	48	11	22
33	41	27	46	44	22	45	8	43	49	10	47
7	13	11	28	42	38	43	41	5	3	18	37
28	18	42	29	46	40	38	19	3	13	17	36
42	45	49	15	23	11	17	36	9	34	29	21
34	26	4	48	18	10	20	46	24	22	15	39
10	39	21	4	2	46	37	23	41	32	14	9
25	8	11	16	49	15	13	27	31	33	10	35
11	15	43	8	44	23	27	19	6	45	48	12
10	50	36	9	49	12	28	27	21	5	26	18
21	29	37	20	46	19	38	13	7	3	43	16
12	6	30	44	42	23	45	17	2	15	11	4
39	22	29	23	19	37	41	47	9	10	45	30
18	34	3	10	50	13	26	2	15	4	36	40
35	25	11	10	43	42	8	24	5	30	44	3
27	35	9	48	30	5	17	32	45	47	15	29
9	19	25	45	3	30	42	41	39	10	5	44
32	49	48	46	6	40	17	7	47	11	15	33
23	2	3	44	21	19	30	6	24	11	47	46
10	23	27	41	13	47	28	32	15	17	2	39
16	2	25	46	48	7	42	47	32	18	13	33
44	17	22	37	34	25	48	45	27	46	9	26
43	40	13	6	2	14	19	16	46	18	12	15
13	18	10	20	31	3	11	21	25	42	2	17
16	3	27	28	24	37	25	19	45	41	2	48
14	21	32	26	29	40	36	9	6	42	50	43
3	2	30	17	7	40	25	42	11	35	6	36
31	19	15	2	44	16	48	17	42	13	35	7
27	11	31	23	9	13	17	38	33	25	36	12
44	35	11	5	48	6	8	24	14	38	39	9
48	30	28	2	41	22	26	24	47	23	38	19
31	20	35	38	21	25	36	28	22	39	29	11
26	48	32	11	3	34	38	18	37	5	8	24
11	35	18	45	48	25	46	42	7	20	49	23
38	28	34	5	31	22	40	7	37	27	8	45

Columns 49 through 51

41	5	1
10	46	1
19	8	1
3	43	1
21	48	1
7	12	1
17	11	1
26	37	1
12	44	1
30	43	1
39	9	1

```
    33    25     1
    32    35     1
    48    39     1
    27    49     1
     2    44     1
    17    35     1
    48    50     1
    49    50     1
    12    35     1
    17    30     1
    16    12     1
    41    42     1
    14    30     1
    33     3     1
     2    28     1
     9    37     1
    32    25     1
    37    41     1
    21    34     1
    31     6     1
    18    11     1
    36    34     1
    27    40     1
    44    30     1
    17    38     1
     5     2     1
    30     5     1
    12    19     1
    26    46     1
    41    27     1
    31     8     1
     4    40     1
    49    15     1
    17    28     1
    13    27     1
     3    26     1
    30    13     1
    32    19     1
    33    11     1


best_route =

  Columns 1 through 12

     1    24    40     7    34    36    12    35    39    43    23    10

  Columns 13 through 24

     6    30     4     8    32    33    37    25    50    42    41    14

  Columns 25 through 36

    38    18     5     3    19    46    13    20    47    45    17    16

  Columns 37 through 48

    31    48     9    49    11    21    28    44     2    15    27    22
```

```
  Columns 49 through 51

    26    29     1

min_globalplot =

   1.0e+03 *

  Columns 1 through 7

    2.0926    2.0373    1.9255    1.8943    1.8886    1.8495    1.8392

  Columns 8 through 14

    1.8205    1.8195    1.7838    1.7752    1.7692    1.7558    1.7455

  Columns 15 through 21

    1.6849    1.6765    1.6722    1.5904    1.5767    1.5174    1.4780

  Columns 22 through 28

    1.4743    1.4385    1.4341    1.4258    1.4102    1.4008    1.3802

  Columns 29 through 35

    1.3725    1.3597    1.3548    1.3544    1.3468    1.3336    1.3251

  Columns 36 through 42

    1.3246    1.3205    1.3177    1.3073    1.2970    1.2677    1.2677

  Columns 43 through 49

    1.2627    1.2565    1.2516    1.2438    1.2398    1.2388    1.2299

  Columns 50 through 56

    1.2259    1.2237    1.2177    1.1979    1.1865    1.1832    1.1742

  Columns 57 through 63

    1.1678    1.1676    1.1653    1.1405    1.1401    1.1246    1.1188

  Columns 64 through 70

    1.1129    1.0769    1.0710    1.0659    1.0499    1.0369    1.0350

  Columns 71 through 77

    1.0106    1.0101    0.9932    0.9879    0.9798    0.9797    0.9741

  Columns 78 through 84

    0.9741    0.9740    0.9732    0.9699    0.9691    0.9644    0.9597
```

```
  Columns 85 through 91

    0.9548    0.9511    0.9462    0.9446    0.9425    0.9421    0.9400

  Columns 92 through 98

    0.9348    0.9324    0.9308    0.9298    0.9289    0.9288    0.9280

  Columns 99 through 105

    0.9255    0.9239    0.9152    0.9075    0.9059    0.8755    0.8666

  Columns 106 through 109

    0.8663    0.8663    0.8660    0.8658


gen_count =

  Columns 1 through 12

     1     3     4    12    15    17    24    25    26    27    28    31

  Columns 13 through 24

    32    33    34    38    39    41    50    52    68    69    72    79

  Columns 25 through 36

    80    81    83    84    86    92    94    97   101   103   105   111

  Columns 37 through 48

   112   113   115   128   131   138   149   155   158   160   162   163

  Columns 49 through 60

   170   171   178   179   180   187   199   200   206   211   214   218

  Columns 61 through 72

   221   227   235   241   252   256   265   280   287   319   323   338

  Columns 73 through 84

   341   357   365   376   379   381   405   425   430   434   435   444

  Columns 85 through 96

   463   469   472   477   492   508   516   531   553   571   581   586

  Columns 97 through 108

   607   618   644   759   794   810   820   832   851   897   909   930

  Column 109

   960
```