

# **A Comparative Study of Tabu Search and Simulated Annealing for Traveling Salesman Problem**

Project Report  
Applied Optimization MSCI 703



Submitted by

Sachin Jayaswal  
Student ID: 20186226  
Department of Management Sciences  
University of Waterloo

## Introduction

A Classical Traveling Salesman Problem (TSP) can be defined as a problem where starting from a node it is required to visit every other node only once in a way that the total distance covered is minimized. This can be mathematically stated as follows [2]:

$$\text{Min} \quad \sum_{i,j} c_{ij} x_{ij} \quad (1)$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \quad \forall j \neq i \quad (2)$$

$$\sum_j x_{ij} = 1 \quad \forall i \neq j \quad (3)$$

$$u_1 = 1 \quad (4)$$

$$2 \leq u_i \leq n \quad \forall i \neq 1 \quad (5)$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) \quad \forall i \neq 1, \forall j \neq 1 \quad (6)$$

$$u_i \geq 0 \quad \forall i \quad (7)$$

$$x_{ij} \in \{0,1\} \quad \forall i, j \quad (8)$$

Constraints set (4), (5), (6) and (7), together are called MTZ constraints and are used to eliminate any sub tour in the solution. This requires  $n$  extra variables and roughly  $n^2/2$  extra constraints. Without the additional constraints for sub tour elimination, the problem reduces to a simple assignment problem, which can be solved as an LP without binary constraints on  $x_{ij}$  and will still result in binary solution for  $x_{ij}$ . Introduction of additional constraints for sub tour elimination, however, makes the problem an MIP with  $n^2$  integer variables for a problem of size  $n$ , which may become very difficult to solve for a moderate size of problem.

## Tabu Search for TSP

Tabu Search is a heuristic that, if used effectively, can promise an efficient near-optimal solution to the TSP. The basic steps as applied to the TSP in this paper are presented below:

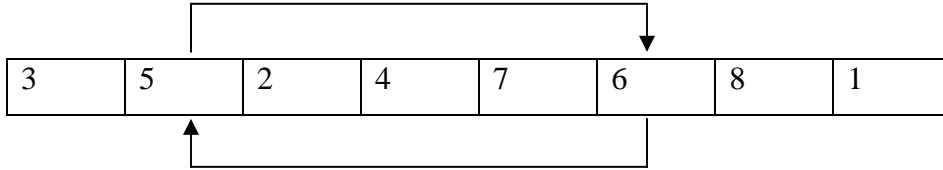
1. **Solution Representation:** A feasible solution is represented as a sequence of nodes, each node appearing only once and in the order it is visited. The first and

the last visited nodes are fixed to 1. The starting node is not specified in the solution representation and is always understood to be node 1.

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 3 | 5 | 2 | 4 | 7 | 6 | 8 | 1 |
|---|---|---|---|---|---|---|---|

**Figure 1: Solution Representation**

2. **Initial Solution:** A good feasible, yet not-optimal, solution to the TSP can be found quickly using a greedy approach. Starting with the first node in the tour, find the nearest node. Each time find the nearest unvisited node from the current node until all the nodes are visited.
3. **Neighborhood:** A neighborhood to a given solution is defined as any other solution that is obtained by a pair wise exchange of any two nodes in the solution. This always guarantees that any neighborhood to a feasible solution is always a feasible solution (i.e, does not form any sub-tour). If we fix node 1 as the start and the end node, for a problem of N nodes, there are  $^{N-1}C_2$  such neighborhoods to a given solution. At each iteration, the neighborhood with the best objective value (minimum distance) is selected.
- 4.



**Figure 2: Neighborhood solution obtained by swapping the order of visit of cities 5 and 6**

5. **Tabu List:** To prevent the process from cycling in a small set of solutions, some attribute of recently visited solutions is stored in a Tabu List, which prevents their occurrence for a limited period. For our problem, the attribute used is a pair of nodes that have been exchanged recently. A Tabu structure stores the number of iterations for which a given pair of nodes is prohibited from exchange as illustrated in Figure 3.
6. **Aspiration criterion:** Tabus may sometimes be too powerful: they may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation of the searching process [3]. It may, therefore, become

necessary to revoke tabus at times. The criterion used for this to happen in the present problem of TSP is to allow a move, even if it is tabu, if it results in a solution with an objective value better than that of the current best-known solution.

7. **Diversification:** Quite often, the process may get trapped in a space of local optimum. To allow the process to search other parts of the solution space (to look for the global optimum), it is required to diversify the search process, driving it into new regions. This is implemented in the current problem using “frequency based memory”. The use of frequency information is used to penalize non-improving moves by assigning a larger penalty (frequency count adjusted by a suitable factor) to swaps with greater frequency counts [1]. This diversifying influence is allowed to operate only on occasions when no improving moves exist. Additionally, if there is no improvement in the solution for a pre-defined number of iterations, frequency information can be used for a pair wise exchange of nodes that have been explored for the least number of times in the search space, thus driving the search process to areas that are largely unexplored so far.

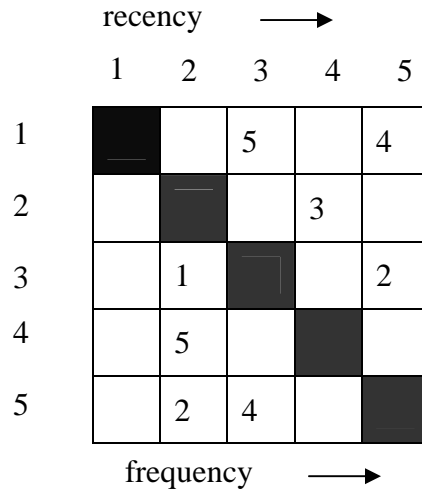


Figure 3: Tabu Structure [1]

8. **Termination criteria:** The algorithm terminates if a pre-specified number of iterations is reached

## Simulated Annealing for TSP

The basic steps of Simulated Annealing (SA) applied to the TSP are described below. The solution representation and the algorithm for initial solution for the SA are same as that for Tabu Search described above.

1. **Neighborhood:** At each step, a neighborhood solution is selected by an exchange of a randomly selected pair of nodes. The randomly generated neighbor solution is selected if it improves the solution else it is selected with a probability that depends on the extent to which it deteriorates from the current solution.
2. **Termination criteria:** The algorithm terminates if it meets any one of the following criteria:
  - a. It reaches a pre-specified number of iterations.
  - b. There is no improvement in the solution for last pre-specified number of iterations.
  - c. Fraction of neighbor solutions tried that is accepted at any temperature reaches a pre-specified minimum.

The maximum number of iterations is kept large enough to allow the process to terminate either using criterion b or c.

**Computational Experience:** Table 1 gives the results of the Tabu Search and SA applied to comparatively small problems that could be optimally solved on GAMS. The distance matrices for the problems are randomly generated using uniform distribution with parameters Min Dist and Max Dist. % Gap gives the relative difference of the solution from the optimal. Table 2 shows the results of the two algorithms applied to large problems from the online library “TSPLIB” for which the optimum solutions are available.

**Results and discussion:** The algorithms for Tabu Search and SA consisting of the strategies described are implemented using MATLAB. The results obtained for the implementation of both SA and Tabu Search for TSP show that they provide good solutions for small size problems tested, with optimal solutions for the tested problems consisting of 10 and 15 cities. The worst optimality gaps obtained are 8.03% and 5.70 % for SA and Tabu Search, respectively. Results for larger problems from TSPLIB, however, are quite far off optimality. Further improvements can be obtained by fine

tuning the various parameters in SA and by better implementation of diversification and intensification in Tabu Search. One improvement that can be employed to Tabu Search is the use of “Reactive Tabu List” that adjusts the Tabu List based on the information from the search procedure [4]

| # Nodes | Min Dist | Max Dist | Optimum | SA    |       | Tabu Search |       |
|---------|----------|----------|---------|-------|-------|-------------|-------|
|         |          |          |         | Obj   | % Gap | Obj         | % Gap |
| 10      | 100      | 1000     | 3043    | 3043  | 0     | 3043        | 0     |
| 15      | 50       | 200      | 1167    | 1167  | 0     | 1167        | 0     |
| 20      | 200      | 1200     | 6223    | 6484  | 4.19  | 6436        | 3.42  |
| 40      | 500      | 2000     | 22244   | 24031 | 8.03  | 23513       | 5.70  |

**Table 1: Comparison of the results with the optimal for small size problems**

| Problem  | # Nodes | Optimum | SA      |       | Tabu Search |       |
|----------|---------|---------|---------|-------|-------------|-------|
|          |         |         | Obj     | % Gap | Obj         | % Gap |
| Berlin52 | 52      | 118282  | 135752  | 14.77 | 125045      | 5.72  |
| Bier127  | 127     | 7542    | 8392.76 | 11.28 | 8667.83     | 14.93 |

**Table 2: Comparison of the results with the optimal for large problems from TSPLIB**

## References:

- [1] Reeves C. R., 1993, Modern heuristic techniques for combinatorial problems, *Orient Longman*, Oxford
- [2] Pataki G., 2003, Teaching integer programming formulations using the traveling salesman problem, *Society for Industrial and Applied Mathematics*, 45 (1), 116-123
- [3] Gendreau M., 2002, An introduction to Tabu Search, [http://www.ifi.uio.no/infheur/Bakgrunn/Intro\\_to\\_TS\\_Gendreau.htm](http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm)
- [4] Battiti R., Tecchiolli G., 1994, The Reactive Tabu Search, *ORSA Journal on Computing*, 6 (2), 126-140
- [5] Glover F., Tilliard E., Werra D.E., 1993, A user's guide to Tabu Search, *Annals of Operations Research*, 41, 3-28

## Appendix A: MATLAB Code for Tabu Search

```
% Travelling Sales man problem using Tabu Search
% This assumes the distance matrix is symmetric
% Tour always starts from node 1

% *****Read distance (cost) matrix from Excel sheet "data.xls"*****
d = xlsread('input_data127.xls');
d_orig = d;
start_time = cputime;
dim1 = size(d,1);
dim12 = size(d);
for i=1:dim1
    d(i,i)=10e+06;
end
% *****Initialise all parameters*****
d1=d;
tour = zeros(dim12);
cost = 0;
min_dist=[ ];
short_path=[ ];
best_nbr_cost = 0;
best_nbr = [ ];

% *****Generate Initial solution - find shortest path from each node*****
% if node pair 1-2 is selected, make distance from 2 to each of earlier
% visited nodes very high to avoid a subtour
k = 1;
for i=1:dim1-1
    min_dist(i) = min(d1(k,:));
    short_path(i) = find((d1(k,:)==min_dist(i)),1);
    cost = cost+min_dist(i);
    k = short_path(i);
    % prohibit all paths from current visited node to all earlier visited nodes
    d1(k,1)=10e+06;
    for visited_node = 1:length(short_path);
```

```

        d1(k,short_path(visited_node))=10e+06;
    end
end

tour(1,short_path(1))=1;
for i=2:dim1-1
    tour(short_path(i-1),short_path(i))=1;
end

%Last visited node is k;
%shortest path from last visited node is always 1, where the tour
%originally started from
last_indx = length(short_path)+1;
short_path(last_indx)=1;
tour(k,short_path(last_indx))=1;
cost = cost+d(k,1);

% A tour is represented as a sequence of nodes startig from second node (as
% node 1 is always fixed to be 1
crnt_tour = short_path;
best_tour = short_path;
best_obj =cost;
crnt_tour_cost = cost;
fprintf("\nInitial solution\n");
crnt_tour
fprintf("\nInitial tour cost = %d\t', crnt_tour_cost);
nbr_cost=[ ];

% Initialize Tabu List "tabu_tenure" giving the number of iterations for
% which a particular pair of nodes are forbidden from exchange
tabu_tenure = zeros(dim12);
max_tabu_tenure = round(sqrt(dim1));
%max_tabu_tenure = dim1;
penalty = zeros(1,(dim1-1)*(dim1-2)/2);
frequency = zeros(dim12);
frequency(1,:)=100000;
frequency(:,1)=100000;

```



```

for i=1:dim1
    frequency(i,i)=100000;
end

iter_snc_last_imprv = 0;
% *****Perform the iteration until one of the criteria is met*****
% 1. Max number of iterations reached*****
% 2. Iterations since last improvement in the best objective found so far
% reaches a threshold*****
best_nbr = crnt_tour;
for iter=1:10000
    fprintf('\n*****iteration number = %d*****\n', iter);
    nbr=[];
    % *****Find all neighbours to current tour by an exchange
    % *****between each pair of nodes*****
    % *****Calculate the object value (cost) for each of the neighbours*****
    nbr_cost = inf(dim12);
    for i=1:dim1-2
        for j=i+1:dim1-1
            if i==1
                if j-i==1
                    nbr_cost(crnt_tour(i),crnt_tour(j))=crnt_tour_cost-d(1,crnt_tour(i))+d(1,crnt_tour(j))-
                    d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1));
                    best_i=i;
                    best_j=j;
                    best_nbr_cost = nbr_cost(crnt_tour(i),crnt_tour(j));
                    tabu_node1 = crnt_tour(i);
                    tabu_node2 = crnt_tour(j);
                else
                    nbr_cost(crnt_tour(i),crnt_tour(j))=crnt_tour_cost-d(1,crnt_tour(i))+d(1,crnt_tour(j))-
                    d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1))-
                    d(crnt_tour(i),crnt_tour(i+1))+d(crnt_tour(j),crnt_tour(i+1))-d(crnt_tour(j-
                    1),crnt_tour(j))+d(crnt_tour(j-1),crnt_tour(i));
                end
            else
                if j-i==1

```

```

        nbr_cost(crnt_tour(i),crnt_tour(j))=crnt_tour_cost-d(crnt_tour(i-1),crnt_tour(i))+d(crnt_tour(i-1),crnt_tour(j))-d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1));
    else
        nbr_cost(crnt_tour(i),crnt_tour(j))=crnt_tour_cost-d(crnt_tour(i-1),crnt_tour(i))+d(crnt_tour(i-1),crnt_tour(j))-d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1))-
        d(crnt_tour(i),crnt_tour(i+1))+d(crnt_tour(j),crnt_tour(i+1))-d(crnt_tour(j-1),crnt_tour(j))+d(crnt_tour(j-1),crnt_tour(i));
    end
end
if nbr_cost(crnt_tour(i),crnt_tour(j)) < best_nbr_cost
    best_nbr_cost = nbr_cost(crnt_tour(i),crnt_tour(j));
    best_i=i;
    best_j=j;
    tabu_node1 = crnt_tour(i);
    tabu_node2 = crnt_tour(j);
end
end
end

% ***** Neighbourhood cost calculation ends here *****

best_nbr(best_i) = crnt_tour(best_j);
best_nbr(best_j) = crnt_tour(best_i);

% ****Replace current solution by the best neighbour.*****
% *****Enter it in TABU List *****
% ****Label tabu nodes such that tabu node2 is always greater than tabu
% node1. This is required to keep recency based and frequency based tabu
% list in the same data structure. Recency based list is in the space
% above the main diagonal of the Tabu Tenure matrix and frequency based
% tabu list is in the space below the main diagonal*****

% *****Find the best neighbour that does not involve swaps in Tabu List***
% ***** OVERRIDE TABU IF ASPIRATION CRITERIA MET*****
% *****Aspiration criteria is met if a Tabu member is better
% than the best solution found so far*****
% while (tabu_tenure(tabu_node1,tabu_node2)|tabu_tenure(tabu_node2,tabu_node1))>0
while (tabu_tenure(tabu_node1,tabu_node2))>0
    if best_nbr_cost < best_obj    %(TABU solution better than the best found so far)

```

```

fprintf('\nbest nbr cost = %d\t and best obj = %d\n, hence breaking',best_nbr_cost, best_obj);
break;
else
%*****Make the cost of TABU move prohibitively high to
%***disallow its selection and look for the next best neighbour
%****that is not TABU Active*****
nbr_cost(tabu_node1,tabu_node2)=nbr_cost(tabu_node1,tabu_node2)*1000;
best_nbr_cost_col = min(nbr_cost);
best_nbr_cost = min(best_nbr_cost_col);
[R,C] = find((nbr_cost==best_nbr_cost),1);
tabu_node1 = R;
tabu_node2 = C;
end
end

%*****Continuous diversification when best nbr cost gt crnt*****
%*****tour cost by penalising objective by frequency of moves*****
if best_nbr_cost > crnt_tour_cost
fprintf('\nbest neighbor cost greater than current tour cost\n');
min_d_col = min(d);
penal_nbr_cost = nbr_cost + min(min_d_col)*frequency;
penal_best_nbr_cost_col = min(penal_nbr_cost);
penal_best_nbr_cost = min(penal_best_nbr_cost_col);
[Rp,Cp] = find((penal_nbr_cost==penal_best_nbr_cost),1);
tabu_node1 = Rp;
tabu_node2 = Cp;
best_nbr_cost = nbr_cost(tabu_node1,tabu_node2);
end

% *****Decrease all Tabu Tenures by 1*****
for row = 1:dim1-1
for col = row+1:dim1
if tabu_tenure(row,col)>0
tabu_tenure(row,col)=tabu_tenure(row,col)-1;
tabu_tenure(col,row)=tabu_tenure(row,col);
end
end
end

```

end

```
%*****RECENCY TABU*****
%Enter current moves in Tabu List with tenure = maximum tenure**
tabu_tenure(tabu_node1,tabu_node2)=max_tabu_tenure;
tabu_tenure(tabu_node2,tabu_node1)= tabu_tenure(tabu_node1,tabu_node2);

%*****FREQUENCY TABU*****
%Increase the frequency of current moves in Tabu List by 1*****
%tabu_tenure(tabu_node2,tabu_node1)=tabu_tenure(tabu_node2,tabu_node1)+1;
frequency(tabu_node1,tabu_node2) = frequency(tabu_node1,tabu_node2)+1;
%*****Update current tour*****
crnt_tour=best_nbr;
crnt_tour_cost=best_nbr_cost;

%*****Update best tour*****
if crnt_tour_cost < best_obj
    best_obj = crnt_tour_cost;
    best_tour = crnt_tour;
    iter_snc_last_imprv = 0;
else
    iter_snc_last_imprv = iter_snc_last_imprv + 1;
    %if iter_snc_last_imprv >= (dim1-1)*(dim1-2)/2
    if iter_snc_last_imprv >= 400
        fprintf('\n NO improvmennt since last % iterations, hence diversify\n',iter_snc_last_imprv);
        min_freq_col = min(frequency); %gives minimum of each column
        min_freq = min(min_freq_col);
        [R,C] = find((frequency==min_freq),1); %find the moves with lowest frequency
        freq_idx1 = R
        freq_idx2 = C
        indx_in_crnt_tour1 = find(crnt_tour==R); %locate the moves in the crnt tour
        indx_in_crnt_tour2 = find(crnt_tour==C);
        %Diversify using a move that has the lowest frequency
        temp = crnt_tour(indx_in_crnt_tour1);
        crnt_tour(indx_in_crnt_tour1) = crnt_tour(indx_in_crnt_tour2);
        crnt_tour(indx_in_crnt_tour2) = temp;
        tabu_tenure = zeros(dim12);
```

```

frequency = zeros(dim12);
frequency(1,:)=100000;
frequency(:,1)=100000;
for i=1:dim1
    frequency(i,i)=100000;
end
tabu_tenure(R,C)=max_tabu_tenure;
tabu_tenure(C,R)=max_tabu_tenure;
frequency(R,C)=frequency(R,C)+1;
frequency(C,R) = frequency(R,C);
%Re-calculate crnt tour cost
crnt_tour_cost = d(1,crnt_tour(1));
for i=1:dim1-1
    crnt_tour_cost = crnt_tour_cost+d(crnt_tour(i),crnt_tour(i+1));
end
iter_snc_last_imprv = 0;
if crnt_tour_cost < best_obj
    best_obj = crnt_tour_cost;
    best_tour = crnt_tour;
end
end
end
%fprintf('\ncurrent tour\n')
%crnt_tour
fprintf('\ncurrent tour cost = %d\t', crnt_tour_cost);
%fprintf('\nbest tour\n');
%best_tour
fprintf('best obj =%d\t',best_obj);
%pause;
end
fprintf('\nbest tour\n');
best_tour
fprintf('best obj =%d\n',best_obj);
end_time = cputime;
exec_time = end_time - start_time;
fprintf('\ntime taken = %f\t', exec_time);

```

## Appendix B: MATLAB Code for Simulated Annealing

```
% *****Read distance (cost) matrix from Excel sheet "data.xls"*****
d = xlsread('input_data20.xls');
d_orig = d;
start_time = cputime;
dim1 = size(d,1);
dim12 = size(d);
for i=1:dim1
    d(i,i)=10e+06;
end

for i=1:dim1-1
    for j=i+1:dim1
        d(j,i)=d(i,j);
    end
end

%d
% *****Initialise all parameters*****
d1=d;
tour = zeros(dim12);
cost = 0;
min_dist=[];
short_path=[];
% *****

% *****Initialize Simulated Annealing paratemers*****
%T0 Initial temperature is set equal to the initial solution value
Lmax = 400; %Maximum transitions at each temperature
ATmax = 200; %Maximum accepted transitions at each temperature
alfa = 0.99; %Temperature decrementing factor
Rf = 0.0001; %Final acceptance ratio
Iter_max = 1000000; %Maximum iterations
```

```

start_time = cputime;

% *****Generate Initial solution - find shortest path from each node*****
% if node pair 1-2 is selected, make distance from 2 to each of earlier
% visited nodes very high to avoid a subtour
k = 1;
for i=1:dim1-1
    min_dist(i) = min(d1(k,:));
    short_path(i) = find((d1(k,:)==min_dist(i)),1);
    cost = cost+min_dist(i);
    k = short_path(i);
    % prohibit all paths from current visited node to all earlier visited nodes
    d1(k,1)=10e+06;
    for visited_node = 1:length(short_path);
        d1(k,short_path(visited_node))=10e+06;
    end
end

tour(1,short_path(1))=1;
for i=2:dim1-1
    tour(short_path(i-1),short_path(i))=1;
end

%Last visited node is k;
%shortest path from last visited node is always 1, where the tour
%originally started from
last_indx = length(short_path)+1;
short_path(last_indx)=1;
tour(k,short_path(last_indx))=1;
cost = cost+d(k,1);

% A tour is represented as a sequence of nodes starting from second node (as
% node 1 is always fixed to be 1
crnt_tour = short_path;
best_tour = short_path;
best_obj =cost;
crnt_tour_cost = cost;

```

```

obj_prev = crnt_tour_cost;

fprintf('\nInitial solution\n');
crnt_tour
fprintf('\nInitial tour cost = %d\t', crnt_tour_cost);
nbr = crnt_tour;

T0 = 1.5*crnt_tour_cost;
T=T0;
iter = 0;
iter_snc_last_chng = 0;
acpt_ratio =1;

%*****Perform the iteration until one of the criteria is met*****
%1. Max number of iterations reached*****
%2. Acceptance Ratio is less than the threshold
%3. No improvement in last fixed number of iterations
while (iter < Iter_max & acpt_ratio > Rf)
    iter = iter+1;
    trans_tried = 0;
    trans_acpt = 0;
    while(trans_tried < Lmax & trans_acpt < ATmax)
        trans_tried = trans_tried + 1;
        city1 = round(random('uniform', 1, dim1-1));
        city2 = round(random('uniform', 1, dim1-1));
        while (city2 == city1)
            city2 = round(random('uniform', 1, dim1-1));
        end
        if (city2>city1)
            i=city1;
            j=city2;
        else
            i=city2;
            j=city1;
        end
        nbr(i)=crnt_tour(j);
        nbr(j)=crnt_tour(i);

```



```

if i==1
    if j-i==1
        nbr_cost=crnt_tour_cost-d(1,crnt_tour(i))+d(1,crnt_tour(j))-
        d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1));
    else
        nbr_cost=crnt_tour_cost-d(1,crnt_tour(i))+d(1,crnt_tour(j))-
        d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1))-
        d(crnt_tour(i),crnt_tour(i+1))+d(crnt_tour(j),crnt_tour(i+1))-d(crnt_tour(j-
        1),crnt_tour(j))+d(crnt_tour(j-1),crnt_tour(i));
    end
else
    if j-i==1
        nbr_cost=crnt_tour_cost-d(crnt_tour(i-1),crnt_tour(i))+d(crnt_tour(i-1),crnt_tour(j))-
        d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1));
    else
        nbr_cost=crnt_tour_cost-d(crnt_tour(i-1),crnt_tour(i))+d(crnt_tour(i-1),crnt_tour(j))-
        d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1))-
        d(crnt_tour(i),crnt_tour(i+1))+d(crnt_tour(j),crnt_tour(i+1))-d(crnt_tour(j-
        1),crnt_tour(j))+d(crnt_tour(j-1),crnt_tour(i));
    end
end
delta = nbr_cost - crnt_tour_cost;
prob1 = exp(-delta/T);
prob2 = random('uniform',0,1);
if(delta < 0 | prob2 < prob1)
    sum = sum+delta;
    crnt_tour = nbr;
    crnt_tour_cost = nbr_cost;
    trans_acpt = trans_acpt + 1;
    if crnt_tour_cost < best_obj
        best_obj = crnt_tour_cost;
        best_tour = crnt_tour;
    end
else
    nbr = crnt_tour;
    nbr_cost = crnt_tour_cost;
end
end

```

```

end
accept_ratio = trans_accept/trans_tried;
fprintf('\niter# = %d\t, T = %2.2f\t, obj = %d\t, accept ratio=%2.2f', iter,T,crnt_tour_cost,accept_ratio);
if crnt_tour_cost == obj_prev
    iter_snc_last_chng = iter_snc_last_chng + 1;
else
    iter_snc_last_chng = 0;
end
if iter_snc_last_chng == 10
    fprintf('\n No change since last 10 iterations');
    break;
end
obj_prev = crnt_tour_cost;
T = alfa*T;
iter = iter + 1;
end

fprintf('\nbest obj = %d', best_obj);
fprintf('\n best tour\n');
best_tour
end_time = cputime;
exec_time = end_time - start_time;
fprintf('\ntime taken = %f\t', exec_time);

```