A Model for Single Machine Job Scheduling Without Due Date

Abdelaziz Hamad Elawad, Mohammed Hassan Elzubair, Bahrom Sanugi

***Abstract*— In this paper we described the single machine Job scheduling without due date as a Symmetric Traveling Salesman Problem (STSP). After formulated the problem we used two methods for solving our model to determined the minimum schedule length, which is our desired objective . One belongs to the construction procedures called Farthest –insertion Algorithm. Another one is Tow-opt algorithm, which belongs to the improvement procedure heuristic. We simulate two C programs to solved our model. our simulation work can cover until 506 jobs.**

***Index Terms*— Single Machine, Scheduling, Traveling Salesman Problem, Farthest –insertion, Two- Opt algorithm.**

I. INTRODUCTION

In this Section we shall introduce our model, i.e. scheduling with a single machine without due date criteria. We begin with giving some remarks about the travelling salesman problem, which is related to our model, followed by local search and insertion heuristic algorithm, which will be used to solve our model.

II. TRAVELLING SALESMAN PROBLEM

The travelling salesman problem (TSP) is one of the oldest problems in combinatorial optimization problems, belonging to the NP-hard class (Garey&Johson, 1979). By an NP-hard class it is meant that the class of those problems for which no polynomial time algorithm has been found. The statement of the TSP problem is deceptively simple. A travelling salesman must visit every city in his territory exactly once and then return to his starting point (Flood, 1956).

The study of this problem has attracted many researchers from different fields, e.g., Mathematics, Operations Research, Physics, Biology and Artificial Intelligence, and there is vast amount of literature on it (Gerhad, 1994). This is due to the fact that, although it is easily formulated, it exhibits all aspects of combinatorial optimization and has served and continues to serve as the benchmark problem for the development of new algorithms such as simulated annealing, tabu search, genetic algorithms, and artificial neural networks

Nevertheless, the TSP is interesting not only from a theoretical point of view but also in the practical aspects. Many practical applications can be modeled as travelling salesman problems. The most common practical interpretation of the TSP is that of a salesman seeking the shortest tour through *n* cities or clients. This basic problem underlines several routing applications, but in this case a number of side constraints usually come into play. Several interesting permutation problems not directly associated with routing can also be described as TSP. Examples of such problems are computer wiring, hole punching, crystallography and job scheduling (Laporte, 1992).

The travelling salesman problem can be categorized as symmetric and asymmetric. Let $W = [w_{ij}]$ denotes the weight matrix of a graph. Then, the edge weights may not be symmetric which means not all $w_{ij} = w_{ji}$. In this paper we will consider our model, i.e. the single machine scheduling without due date as a symmetric TSP.

III. THE MODEL

Suppose n independent jobs must be performed sequentially on a single machine. Let c_{ij} is the cost of travelling from city i to city j. We also assume that
$c_{ij} = c_{ji}$ which makes the problem symmetric. The matrix $C = [c_{ij}]$ is therefore a symmetric cost matrix. This problem can be described as a symmetric travelling salesman problem (TSP). We will elaborate on the use of heuristics algorithm, farthest insertion and 2-opt algorithm to determine the minimum schedule length, which is our desired objective (Golden *et al*, 1980).

IV. APPROXIMATE ALGORITHM

As mentioned earlier, the traveling salesman problem is one of those combinatorial optimization problems for which no efficient algorithms (i.e., with computation time bounded by a polynomial in the number of nodes n) are known. When faced with such computationally hard (NP-hard) problems, one practical approach is to relax the requirement that an algorithm produces an optimal result, and instead be satisfied with a sub-optimal solution reasonably close to the optimal solution. Such relaxation of the optimality constraint often reduces the computation time from an exponential to a polynomial function (of the size of the input). Such approximate algorithms are the only realistic methods of solving computationally hard problems of large size.

Manuscript received May 19, 2014.
Abdelaziz Hamad Elawad, Taif University, Kingdom of Saudi Arabia, Raniah Branch, Faculty of Sciences and Arts, Department of Mathematics
Mohammed Hassan Elzubair, Taif University, Kingdom of Saudi Arabia, Raniah Branch, Faculty of Sciences and Arts, Department of Mathematics
Bahrom Sanugi, Universiti Sains Islam, Malaysia

For the travelling salesman problem, a number of such approximate algorithms have been proposed and studied. Here we will consider two of the most effective, each represents its class of approximate algorithms (Golden at el., 1980).The first is the farthest insertion algorithm which represent the class of insertion algorithms, and the second one is the 2-opt algorithm which represent the local –search class of algorithms

A. *Insertion Algorithm*

An insertion algorithm arbitrarily picks a city as the starting node, say *s*, of tour. From among the remaining *(n-1)* cities it selects another city (according to a criterion to be discussed shortly*)*, say *p*. Now we have a subtour or cycle of two nodes *(s, p, s)*. Then a third node, *q*, out of the *(n-2)* unvisited nodes is selected, and is inserted into the current cycle to produce cycle *(s, p, q, s)* or *(s, q, p, s)* whichever is cheaper. This enlargement is continued. At any stage let V_T denotes the set of nodes included in the subtour, and let V be the entire nodes set for this problem.

The *k*th iteration *($1 \le k \le n-1$)* enlarges the cycle of size *k* to one of size *k+1* by means of the following two steps: (Golden at el., 1980)

Step 1 (selection step). In the set V- V_T of unvisited nodes, determine which node is to be added to the cycle next.

Step 2 (insertion step). Determine where (i.e., between which two nodes of the cycle) the newly selected nodes is to be inserted to enlarge the current subtour.

A number of heuristics have been suggested and experimented with for the selection step. Some of these are

(i) Pick any unvisited nodes at random (the algorithm is called arbitrary insertion);

(ii) Pick that unvisited node which is nearest to the current subtour (the algorithm is called nearest insertion);

(iii) Pick that unvisited node which is farthest from the current subtour (the algorithm is called farthest insertion).

Of the three heuristics mentioned, farthest insertion appears to be the best overall strategy, according to a number of extensive and independent empirical studies (Golden et al., 1980). The underlying intuition behind this approach is that if a rough outline of the tour can be constructed through the widely –separated nodes, then the finer details of the tour resulting from the inclusion of the nearer nodes can be filled in without greatly increasing the total length of the tour. Therefore, we will try to use the farthest insertion algorithm as initial solution to solve our first model.

In order to find efficiently that unvisited node, which is farthest from the nodes in the current cycle V_T. Let us maintain a distance array dist (of length n) such that for every node *v* not in cycle, dist (v) is the distance to *v* from that node in the current cycle from which v is closest. The farthest node f is the one with the largest value in the dist array, and it is the node to be inserted next. Each time a new node f is added to the cycle, the dist array is updated such that its entries are the minimum of the current entries in the dist array and the fth row in W, where W=[w_{ij}] denote the weight matrix.

Having settled on the selection step, let us now look at the insertion step. Assume that there are k nodes in the current cycle, and the next (farthest) nodes to be inserted is f. We examine every edge (i, j) in the current tour to determine the insertion cost of f between nodes i and j, which is

$$c_{ij} = w_{if} + w_{fj} - w_{ij}.$$

Among all k edges in the cycle we select edge (t, h) with tail t and head h for which c_{th} has the smallest value (c_{ij} could be negative). Then insert node f between t and h. The weight of the cycle is updated. We also update the dist array. The following FITSP (farthest insertion travelling salesman procedure) algorithm describes this heuristic.

Algorithm Farthest- Insertion
W=[w_{ij}] denote the weight matrix,
V_T denotes the set of nodes included in the subtour,
V denotes the entire nodes set of the problem.
Initialization:
V_T = {s}; / Initial cycle of one node s and cycle of zero weight at s /
E_t = {(s, s)};
w_{ss} = 0;
tweight = 0; / total weight of the subtour /
for every node u ∈ $V - V_T$ **do** dist(u) = w_{su};
Iteration:
while $|V_T| < n$ **do**
begin
f = node in V-V_T with largest value of dist(f);
for every edge (i,j)∈ E_T **do**
$c_{ij} = w_{if} + w_{fj} - w_{ij}$ / Insertion cost /
(t, h) = edge in E_T with smallest value of c_{th};
$E_T = E_T \cup \{(t, f),(f, h)\} - \{(t, h)\};$
$V_T = V_T \cup \{f\};$
tweight = tweight + c_{th};
for all x∈ (V-V_T) **do** dist(x)=min{dist(x), w_{fx}}
end

B. *Local Search Heuristics*

The local search is one of the best known and most successful heuristics for TSP to obtain a nearly optimal solution.

(i) Starting with an arbitrary Hamiltonian cycle (as initial TSP tour) H, delete r edges from H, thus producing r disconnected paths. Reconnect these *r* paths in such a way as to produce another TSP tour H'. Thus H and H' differ from each other by exactly *r* edges; the remaining (n - r) edges are common.

(ii) Compute the total weight w (H') of tour H'. If w (H') < w (H), replace H with H' and repeat the process, otherwise, pick another set of r edges from H to

exchange. Such exchanges of set of *r* edges continue until exchanging make no additional improvement can be made by exchanging r edges.

(iii) The final solution, which cannot be improved by exchanging any of its *r* edges, is called an *r*-optimal or (*r*-opt for short) solution.

Clearly, the edges-exchange procedure will terminate at a local optimum, thus producing an approximate solution. This approximate algorithm illustrates a general approach to solving many combinatorial optimization problems, known as local search or neighborhood search. For solving our model we will consider the 2-opt (r = 2) exchange.

Two-opt Algorithm

The 2-opt move in general consists of eliminating two edges and reconnecting the two resulting paths in a different way to obtain a new tour. Let the initial cycle consists of the following set of edges H = {x_1, x_2, x_3,…, x_n} in the order x_1, x_2, x_3,…, x_n .Let X ={x_i, x_j} be a set of two edges in H which we delete and replace with edges Y ={y_p, y_q}, if there is an improvement. That is, H' = (H - X) ∪ Y is a new and improved tour.

The improvement is given by δ where

$$\delta = w(H) - w(H') = w(x_i) + (x_j) - w(y_p) - w(y_q).$$

We will examine all tours H`and retain the one for which δ is maximum. If δ_{max} is negative or zero, we have found a 2-opt solution. If $\delta_{max} > 0$, we use the corresponding solution as the initial tour and then repeat the entire procedure. We continue this successive tour improvement until δ_{max} is a non-positive number. The following algorithm describes this method of achieving a 2-opt approximation solution of the TSP.

Two-opt algorithm
begin
let H = (x_1, x_2,…, x_n) be the current tour;
repeat
$\delta_{max} = 0$;
for i=1 to (n-2) **do**
for j = (i+2) to n
if $(w(x_i) + w(x_j)) - (w(y_p) + w(y_q)) > \delta_{max}$ **then**
begin
$\delta_{max} = (w(x_i) + w(x_j)) - (w(y_p) + w(y_q))$;
save i and j
end;
if δ_{max} > o then $H = H - \{x_i, x_j\} \cup \{y_p, y_q\}$
until δ_{max} = o
end

V. Computer Simulations

In this section we show the results of computer simulations with inputs chosen to reflect various level of complexity.

A. *Computer Simulation for Farthest Insertion Heuristic*

We are using the farthest insertion heuristic to construct a tour. For this purpose we simulate a symmetric cost matrix of n nodes or stations as the input of our program. The program code gives us the final tours constructed (see Example 1). The costs of tours are also given as the result of this program. Table.1 shows the output of this program for 200 nodes, which will be used as the input for the 2-opt heuristic.

B. *Computer simulation for 2-opt Heuristic*

We are using the 2-opt algorithm to improve initial route given. In our simulation we used the tour construct by farthest insertion as initial tour for this heuristic which will be improved later. The program code gives us the final improvement tours and their weights as the result of this program (see Example 2). Table 2. shows the result of this program for 200 nodes.

Example 1

The n×n matrix is generated by the simulation rule given in the following for general n. By putting n = 200 the initial set of data as follows.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#undef abs
#define n  200
#define s  0
#define inf   20000
        void main()
 {
 int end1, end2, farthest, i, j,index,inscost;
 int maxdist, newcost, nextindex, cycle[n+1], dist[n+1];
 int n1,w[n+1][n+1],route[n+1],tweight ;
 n1 =n-1;
 for (i=0;i<=n1;++i)
 {
 for (j=0;j<=n1;++j)
 {
 if (i != j)
 w[i][j]= abs((i*i)+(j*j)-i*j);
 else
 w[i][j]=0;
 w[i][j] =fmod(w[i][j],200);
 if(i != j && w[i][j]< 20)
 w[i][j]= w[i][j]+20;
 }
 }
```

Final constructed route:
0,199,195,155,151,98,69,185,104,119,198,159,101,125,134
,23,49,27,166,25,189,164,39,88,29,144,136,95,99,41,105,1
37,80,100,20,121,56,64,66,177,52,127,46,191,
55,188,158,57,85,181,43,62,42,112,170,154,48,70,162,192
,15,71,147,94,171,187,107,172,190,118,90,18,78,110,132,
32,122,153,163,182,10,178,197,140,109,65,156,91,102,50,
37,19,60,40,22,82,28,143,117,6,4,5,129,34,186,84,115,31,
54,33,139,106,67,193,53,128,8,76,174,51,176,108,138,87,
150,63,47,175,72,97,157,183,194,83,35,131,93,142,21,1,1
80,38,7,58,77,146,169,11,16,114,149,24,26,167,152,86,17,
13,126,73,74,173,148,123,14,81,116,12,45,160,179,124,79
,135,3,44,9,165,120,111,75,36,161,61,141,2,96,184,89,113
,133,145,196,130,92,30,59,103,68,168.
Total weight = 5160, Number of nodes = 200

1)

2)

3) Example 2

4)

We use the 2-opt heuristic to improve the results obtained from the farthest insertion heuristic. The following is the initial set of for this example data.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#undef abs
#define  n  200
        void main()
{
 int  i, i1, i2, j, j1, j2, s1, s2,t1,t2;
 int  n1,n2;
 int next,last,index,limit,ahead;
 int max, max1,tweight = 5160;
 int w[n+1][n+1];
int  ptr[n+1],route[n+1]=
{0,199,195,155,151,98,69,185,104,119,198,159,101,125,13
4,23,49,27,166,25,189,164,39,88,29,144,136,95,99,41,105,
137,80,100,20,121,56,64,66,177,52,127,46,191,55,188,158
,57,85,181,43,62,42,112,170,154,48,70,162,192,15,71,147,
94,171,187,107,172,190,118,90,18,78,110,132,32,122,153,
163,182,10,178,197,140,109,65,156,91,102,50,37,19,60,40
,22,82,28,143,117,6,4,5,129,34,186,84,115,31,54,33,139,1
06,67,193,53,128,8,76,174,51,176,108,138,87,150,63,47,1
75,72,97,157,183,194,83,35,131,93,142,21,1,180,38,7,58,7
7,146,169,11,16,
114,149,24,26,167,152,86,17,13,126,73,74,173,148,123,14
,81,116,12,45,160,179,124,79,135,3,44,9,165,120,111,75,3
6,161,61,141,2,96,184,89,113,133,145,196,130,92,30,59,1
03,68,168};
 n1 = n - 1;
 n2 = n1 - 1;
            for (i=0;i<=n1;++i)
            {
            for (j=0;j<=n1;++j)
            {
            if (i != j)
            w[i][j]= abs((i*i)+(j*j)-i*j);
            else
            w[i][j]=0;
            w[i][j] = fmod(w[i][j],200);
            if(i != j && w[i][j]< 20)
                                    w[i][j]= w[i][j]+20;
 }
          }
```

Final route:
0,160,120,111,75,36,161,61,141,167,26,23,134,1,21,76,10
5,16,155,4,5,129,169,
146,77,103,68,168,63,37,43,62,113,145,196,195,11,175,17
6,125,101,159,73,126,
13,107,187,49,24,25,189,164,39,55,191,136,95,99,58,188,
48,154,183,194,83,114,41,85,44,3,197,178,10,182,163,6,1
17,143,28,93,142,12,116,35,131,17,86,152,2,50,102,91,15
6,65,84,115,31,54,33,139,106,67,193,53,128,8,170,112,42,
51,71,15,59,64,9,165,81,14,123,29,144,46,127,52,177,66,8
9,184,96,181,140,109,186,19,60,40,22,82,172,190,118,90,
18,78,110,132,32,122,153,27,166,149,47,98,69,185,104,11
9,198,
150,148,88,130,92,30,192,162,70,158,38,7,87,74,173,34,1
51,171,94,147,72,97,157,45,121,56,135,79,124,179,199,13
3,174,137,57,138,108,180,80,100,20,

a) Total weight of the final route = 4722, Total number of nodes = 200

Table : Total weight of farthest and two-opt heuristics

No of Jobs	Farthest	2-opt	Improvement	Percentage
10	375	375	0	0.00
20	1036	827	209	0.25
50	1574	1298	276	0.21
80	2411	2030	381	0.19
100	2787	2456	331	0.13
125	3447	3030	417	0.14
150	3908	3597	311	0.09
200	5160	4722	438	0.09
300	7658	7046	612	0.09
350	8786	8181	605	0.07
400	10074	9396	678	0.07
450	11308	10508	800	0.08
500	12608	11649	959	0.08

VI. Conclusions

In this paper we considered two methods for solving our model for single machine scheduling without due date criteria. One belongs to the construction procedures called farthest -insertion. Another one is 2-opt algorithm, which belongs to the improvement procedure heuristic. We simulate two C programs to solving our model. Our program can schedule over 1000 nodes (jobs). But our simulation work can cover until 506 nodes (jobs) due to the computer storage requirements. The Table shows the result of our simulation using several values of n. Example.1 shows the total cost (5160) and the constructed route of 200 jobs by farthest heuristic. While Example.2 shows the total cost

(4722) which is the improved route of the same 200 jobs by using 2-opt heuristics.

ACKNOWLEDGMENT:

The authors would like to thanks the Head Department of Mathematics Rianah Branch , Taif University Kingdom of Saudi Arabia for their facilities

REFERENCES:

[1] Abdelaziz Hamad PhD Thesis (2002) "Application of neural network for solving job scheduling problems, UTM, Malaysia
[2] Abdelaziz Hamad and Bahrom Sanugi (2011) . Neural Network and Scheduling. Germany : Lap Lambert Academic Publishing.
[3] Baker, K.R., (1974). Introduction to Sequencing and Scheduling. New York: John Wiley.
[4] Flood, M. M. (1956). "The Traveling Salesman Problem". Operations Research. 4; 61-75.
[5] Graham, R., E. Lawler, J. Lenstra, and A. Rinnooy Kan (1979)." Optimization and Approximation in Deterministic Sequencing and Scheduling Theory: A Survey Annals of Discrete Mathematics. 5; 287-326.
[6] Garey, M. and Johson, D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: W.H. Freeman.
[7] Golden B., Bodin, L. and Stewart, W. (1980)."Approximate Traveling Salesman Algorithm." Journal of Operation Research. 28; 694-711.
[8] Laporte, G. (1992), "The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithm." European Journal of Oper. Res.59, 231-247.
[9] Panwalkar, S.S., Rajagopalan, R.(1992). "Single Machine Sequencing with Controllable Processing Times." European Journal of Operation Research. 59, 298-302.