Journal of the Korean
OR/ MS Society
Vol. 18, No. 2, August 1993

A Model for Material Handling in an Elevator System

Seung-Nam Kim*

Abstract

This study deals with finding a schedule for the movement of a material handling device(elevator) in a manufacturing plant. Two different algorithms(Traveling Salesman Technique and Greedy Algorithm) are used in the scheduling of the elevators using a simulation technique to determine the proper method of scheduling the elevator movement. Based on the simulation analysis, we have found that the Greedy algorithm serves better than the algorithm based on Traveling Salesman technique for scheduling the movement of a material handling device in the manufacturing plant.

1. Introduction

Material handling plays a key role in improving the productivity in any industry[11]. This study deals with finding a schedule for the movement of a material handling device(elevator) in a manufacturing plant, thereby enabling the smooth execution of the various activities of the plant and prevention of bottle-neck situations that may arise due to unorganized material movement. In this paper two different algorithms(Traveling Salesman Technique and Greedy Algorithm[7]) are used in the scheduling of elevators using a simulation technique to determine the proper method of scheduling the elevator movement. This paper begins with a brief description of material handling problem. Next, date cllection, analysis techniques, model analysis are discussed. Finally, the results and associated conclusions are reported.

* Department of Management, Cho Sun University

2. Problem Statement

This study is concerned with the material movement in the building of ALLEN BRADLEY, a firm located in Milwaukee, Wisconsin.

This firm's building has nine floors and two vertical lift conveyors(elevators) that pick up and deliver palliative loads among the different of the building. The performance of the vertical lift conveyors plays a major role in the automated material handling system.

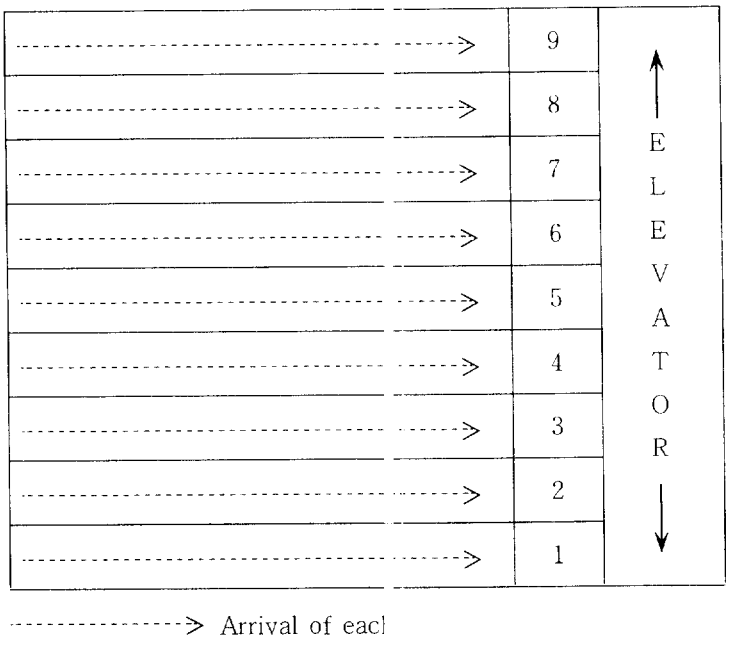

Figure 1. ALLEN BRADLEY Elevator System

Loads present themselves at the various floors of the building, requiring to be moved to other floors. The elevators in the firm are of same capacity. Rather than studying the system with two elevators, for the sake of simplicity, we consider only one elevator for analysis purposes and to justify this move, the arrival of loads at each floor is adjusted by only considering half the arrivals. Thus, we have a system which can be represnted in Figure 1. System specifications(which are based upon the company's personnel considered) are times required regarding the movement of the elevator, which are as follows:

(1) 9 seconds to reach the next floor(up or down) from a given position.

(2) 6 seconds for reaching each of the subsequent floors until the destination is reached.

(3) The load cycling time is 15 seconds to cyc e the load either onto or off the elevator.

Our goal is to identify a method or an algorit m to schedule the movement of the elevator, giving prime importance to the maximum queue le gth of loads waiting for the elevator at each floor. This quantity is of interest as it provides inform ation on the space to be allocated for storing loads which require elevators at each floor and the kı owledge of this information is essential in facilities location[9].

3. Data Collection

Ths data used for the purpose of analysis comes from the firm and is collected by the company's Industrial Engineering Departme it personnel. The data represents the load movements between the various floors and is resented in Table 1. Based on their observations and analysis, they conclude that the loads arri e at the various floors, requiring to be moved to their respective destinations in accordance wit the data given in the Table 1.

Table 1. Ot servation Data

To / From	1	2	3	4	5	6	7	8	9
1	0	79	8	2	15	2	4	4	15
2	85	0	6	0	76	53	80	120	30
3	0	33	0	0	0	0	0	0	0
4	0	179	0	0	0	0	0	0	0
5	0	65	0	9	0	0	0	0	0
6	0	0	0	41	0	0	0	0	0
7	0	0	0	150	0	0	0	0	0
8	0	0	0	128	0	0	0	0	0
9	0	0	0	56	0	0	0	0	0

The data is provided for an eight hour shift. The arrival of loads is found to follow a Poisson distribution[1][2] with the mean depending o the values of observation. Thus, for instance, the arrive of loads at 6th floor to be moved t the 4th floor is assumed to follow a Poisson distribution with a mean of 41.

4. Analysis and Model Development

4.1 Assumptions

To determine the proceding operating characteristics, we need to make certain assumptions about the elevator system[1][2][6][9]. Here are the most important assumptions which affect the operating characheristics of the system.

(1) The arrivals of the loads at the various floors follow Poisson distribution with the mean values given in Table 1.

(2) The service times are deterministic i.e. the lift movement time is known.

(3) The service discipline is FCFS.

(4) The queue length is not restricted.

(5) Every load waiting for the elevator gets its turn to be moved to its destination floor.

(6) The elevator is not subject to breakdown i.e the elevator is assumed to work throughout the time period being considered.

(7) By dividing the arrival rate of the loads by two, only one elevator is considered for the purpose of analysis.

(8) The elevator at a time has room to hold only one load for the purpose of transportation.

4.2 Analysis

The scheduling of the elevators is done using two approaches. The first approach is based on the Traveling Salesman technique which gives the minimzied time for a set of sequence dependent processes. The second approach is based on the Greedy algorithm, where the elevator from a given position is directed to the nearest floor in which a load is waiting to be picked up.

For both the algorithms, the loads waiting at the various floors to be moved to their respective destinations are found according to the load movement. Based on the floor to be reached next(which is provided by the algorithm), the scheduling of the elevator is done.

The productive time of the elevator movement denotes the time needed for the movement of the elevator WITH LOAD between the floors plus the time required for the shuttles to cycle the load onto or off the lift whereas the idle time represents the time needed for the movement of the elevator from one floor to another WITHOUT LOAD to pick up a load at that floor. The

number of loads waiting to be picked up at all the floors is kept track and the maximum queue length of every floor is recorded. A productive move constitutes the movement of the elevator through one floor WITH load, and idle move denotes the movement of the elevator through one floor WITHOUT load. The values of productive time and idle time, the maximum queue length at the different floors and the number of productive and idle moves are provided for both approaches.

The number of loads waiting at the different floors is essential in determining the buffer size to be provided at the various floors or in determining the possibility of the existing buffer size to accommodate the waiting loads.

The data is used to generate the arrivals of load with the interarrival time following exponential distribution since the arrivals are assumed to follow Poisson distribution.

For both the algorithms(based on Traveling Salesman technique and Greedy algorithm), the arrival of the first load should be determined to start the scheduling. For this purpose, the program generates exponential variates with the aid of an in-built random number generator. These exponential variates are also used to determine the subsequent arrivals of loads. The exponential variates are qenerated with reference to the load movements between floors and they thus provide the main input information for the program to be acted on. The loads arriving at the various floors are recorded and the queue lengths at the different floors are updated when the loads arrive at the floors and when the loads are picked up by the elevator.

4.3 Traveling Salesman Algorithm

The algorithm based on Traveling Salesman technique gives prime importance to the time involved in movement between floors[7]. It basically forms a time matrix for the nine floors of the building i.e. the time involved in reaching the different floors with reference to the current elevator position(the floor in which the elevator is present now). The algorithm then proceeds by finding the minimum element in the row and column for all rows and columns of the 9x9 matrix. It then subtracts the minimum row quantity from the elements in that row. This process is extended to all rows of the matrix. Then the minimum column value is subtracted from all elements in that column. All columns are subjected to the same treatment. The algoithm then spots the zeroes of this reduced matrix and labels each zero with the sum of minimum element in the row(the row in which the zero is present) and the minimum element in the column. Among the labelled zeroes, it selects the one with maximum label value and then decides the move to be mode next with the row and column values of that selected zero. The

idle time and productive time values are updated and the waiting loads at the different floors are decremented when the loads are picked up from them. The number of productive and idle moves are also recorded.

To illustrate the working of Traveling Salesman technique, consider a building with 4 floors. Let us retain the same time values for the lift movement i.e. 9 seconds to reach the next floor (up or down) from a given position and 6 seconds to reach the subsequent floors until the destination is reached. The time needed to cycle the load either onto or off the lift is considered to be 15 seconds.

Let us assume that the elevator is currently at the third floor. The algorithm first sets up a cost matrix of size 4x4 to account for the four floors i.e. taking i and j to denote floors, the cost matrix means an array of time values needed to reach floor i from the third floor(current floor) plus the time required to reach floor j from i with i and j taking values from 1 to 4. The load cycling time is also considered in setting up the cost matrix as in Table 3. The algorithm then determines the next floor to be reached based on the relative time values needed to move from one floor to another. The algorithm takes care of the situation in which there is no load present requiring to be moved from one particular floor to another by assigning a large positive number as the time required to move between those two floors. We denote that large number by a very large value of M for purpose of analysis. Also, it can be noted that M minus a given value can be taken as equal to M itself as M is very large. The loads waiting at the different floors are found as in Table 2. Table 3 shows the cost matrix for the loads in Table 2.

Table 2. Load Position

	1	2	3	4
1	0		0	2
2	1		4	1
3	2		0	0
4	5		3	0

Table 3. Cost Matrix

	1	2	3	4
1	M	24+30	M	36+30
2	18+30	M	18+30	24+30
3	15+30	9+30	M	M
4	30+30	24+30	18+30	M

	1	2	3	4
1	M	54	M	66
2	48	M	48	54
3	45	39	M	M
4	60	54	48	M

Then the algorithm proceeds by finding th minimum element in the row subtracts it from elements in the row for all the rows. Thus, th results are presented in Table 4.

Table 4. Row Operations

	1	2	3	4
1	M	0	M	12
2	0	M	0	6
3	6	0	M	M
4	12	6	0	M

Then the minimum element in the column s found and subtracted from all column elements for every column. Thus, the results are preser ted in Table 5. Then the zeroes are labelled with the sum of the minimum element in the row(i which the zero is present) and the minimum element in the column as shown in Table 6.

Among the labelled zeroes the one with the aximum lable value is selected and if there are many zeroes with the same maximum value, th en any such zero can be selected.

Table 5. Co umn Operations

	1	2	3	4
1	M	0	M	6
2	0	M	0	0
3	6	0	M	M
4	12	6	0	M

Among the labelled zeroes the one with the aximum lable value is selected and if there are many zeroes with the same maximum value, th en any such zero can be selected.

Thus, the algorithm chooses the element in he second row and first column. This means that the elevator should now move from the third floor to the second floor and then pick up a load

	1	2	3	4
1	M	0	M	6
2	0	M	0	0
3	6	0	M	M
4	12	6	0	M

from the second floor to the first floor. The move from third to secnd floor is an idle move as the elevator does not carry any load, and the move from the second floor to the first floor constitutes a productive move. The time values for these moves are recorded. Now the loads arriving at the different floors are found and the number of loads waiting at the second floor to be moved to the first floor is decreased by 1.

The loads waiting at the different floors are found again and with reference to the current elevator position(first floor), the algorithm proceeds in a similar manner to decide the next move.

4.4 Greedy Algorithm

The Greedy algorithm directs the elevator from a given position(floor) to the nearest floor in which a load is waiting to be picked up[7]. It can be noted that Greedy algorithm reduces the idle time involved in the scheduling. The values of idle time, productive time, the number of loads waiting at the different floors and the number of productive and idle moves are updated after every move.

The program is written in Turbo-Pascal and the listing is included in the Appendix I.

4.5 Model

The model used is a simple one and can be represented by:

$$Y_{ri} = \mu_i + E_{ri}$$
$$r = 1, \ldots, 6,$$
$$i = 1, 2,$$

where

Y_{ri} = value of the observed performance measure(Total maximum queue length)

μ_i = true mean of the performance measure for model i

$E_{r,i}$ = error term for the rth replication for the i th model.

It is assumed that $E_{r,i}$ are i.i.d. random variables with mean 0.

4.6 Validation and Verification

The ideal way to validate and verify the model when dealing with a real life situation will be to compare the program(output) results with those in the actual situation. Since it was not easy to do this in the situation we are considering now, the verification was done by running the program for various sets of data and observing the results change in accordance with the input i.e. an increased arrival rate resulting in an increase in the queue length at each floor etc.

5. Output Analysis

The program was run and different replications were made by giving different seeds as the initial seeds for every run(Appendix Ⅱ).

5.1 Performance Measurement

The value of interest in every run was the total maximum queue length(at all floors). These observations were recorded for every run. There were also other quantities like productive time, idle time that were provided by the program output.

5.2 Statistical Analysis

There was a clear evidence of the superiority of one method over the other and a total of 10 replications were made to confirm this. The recorded values and the statistical test performed to compare two means is represented in Table 7.

Table 7. The Means of Replications

Replication	Greedy	TSM
1	117	251
2	147	241
3	112	283
4	141	252
5	142	184
6	114	216
7	140	224
8	130	272
9	128	241
10	143	240
Mean	131.4	240.4
Variance	172.93	789.6

$X = Y_1 - Y_2 = -109$

$SE(X) = ((9*172.93) + (9*789.6))/18 = 21.93.$

90% Confidence Interval is as follows :

$$X - t_{\alpha/2}*SE(X) \leq \mu_1 - \mu_2 \leq X + t_{\alpha/2}*SE(X)$$

$$-147.02 \leq \mu_1 - \mu_2 \leq -70.97$$

No zero is in the interval. Therefore, we conclude that there is a difference ($\mu_1 < \mu_2$) in the performance of the algorithms. Greedy algorithm has a smaller value for the Total Maximum Queue Length when compared to the algorithm based on Traveling Salesman technique.

The results obtained indicate that the Greedy algorithm performs better than the algorithm based on Traveling Salesman Technique. Further experimentation is done to see the range over which the Greedy Algorithm gives good results. It can be noted that the second floor of the building has much greater arrival rate than any other floor. Now, the arrival rate at the second floor is decreased so that it is almost equal to that in any other floor and it was observed that the algorithm based on Traveling Salesman technique gave smaller values for total maximum queue length. Thus, we can conclude that the Greedy Algorithm works better than the algorithm based on Travelling Salesman technique in cases like the situation on hand i.e. when the arrival rate of loads at one floor far exceeds the arrival rate in other floors, and for other cases where the arrival rates are uniform over all floors, we need to experiment further by constructing an appropriate model for analysis.

6. Discussion and Conclusion

The results of this study may have several limitations. First, for the sake of simplicity, we considered only one elevator and several assumptions based on the opinion of IE department personnel. A second limitation involves the representativeness of the sample. The present study investigated the data collected by a specific company. It is conceivable that different results would emerge from examinig the same issues in other companies. Clearly, any inferences based on the results would be restricted to other companies. Thus, the specific company limits the generalibility of our findings. Despite these limitations, this study provides a useful perspective on the scheduling for material handling device Based on the simulation analysis, it is clear that the Greedy algorithm serves better than the algorithm based on Traveling Salesman tecnique for this situation. Greedy algorithm as explained before is another form of Shortest Processing Time(SPT) and does well in this context. Also, SPT is extensively used for scheduling purposes in industries today and the results obtained are quite good.

APPENDIX I

```
PROGRAM ELEVATOR :
CONST
        SIZE=9 ;
        MAX_INT=32000 ;
        p=1 ;
TYPE
      mat1=array[1…SIZE,1…SIZE] of integer ;
      mat2=array[1…SIZE,1…SIZE] of real ;
      mat3=array[1…SIZE,1…SIZE] of integer ;
      mat4=array[1…SIZE,1…SIZE] of real ;
      mat5=array[1…SIZE,1…SIZE] of integer ;
      mat6=array[1…SIZE,1…SIZE] of integer ;
      mat7=array[1…SIZE] of integer ;
VAR
      loadavg : matl ;
      lambda : mat2 ;
      floortime : mat3 ;
      totaltime, k, i : integer ;
      movetime, firstentry : real ;
      nextarrival : mat4 ;
      waitingloads : mat5 ;
      maxqlength : mat7 ;
      idletime, prodtime : integer ;
      costmatrix : mat6 ;
      lastfloor : integer :
      datafile, listing : text ;
      seed : integer ;
      itime, ptime : integer ;
      a : char ;
      simtime : real ;
      noloads : boolean ;
      pmove, imove : integer :
      time1, time2 : integer ;
      totqlength : integer ;
```

```
FUNCTION RANDOM(var seed : integer) : real ;
BEGIN
        seed : =seed*3993+1 ;
        if seed<0 then
        seed : =seed+32767+1 ;
        random : =seed*3.051851e-5 ;
END ;
FUNCTION EXPVARIATE(lambda : real) : rea ;
VAR
     x, y : real ;
BEGIN
      if (lambda=MAX_INT) then x:=MAX_INT else
         begin
         y :=random(seed) ;
         x :=-ln(1-y)*(lambda) ;
             end ;
         expvariate:=x ;
END ;
PROCEDURE READAVG ;
VAR
   i, j : integer ;
BEGIN
      for i :=1 to SIZE do
          begin
               for j :=1 to SIZE do
                         readln(datafile, loadavg[i, j]) ;
           end ;
END ;

PROCEDIRE INITIALIZE ;
VAR
   i, j, z : integer ;
BEGIN
       for i :=1 to SIZE do
           begin
```

```
            for j:=1 to SIZE do
              begin
               if loadavg[i, j] < > O then
                  lambad[i, j] :=(2.0*8.0*60.0*60.0)/(loadavg[i, j])
                  else lambda[i, j]:=MAX_INT
               if i=j then floortime[i, j] :=0
               else
                   begin
                        z :=abs(i-j)-1 ;
                        floortime[i, j] :=(z*6)+5 ;
                    end ;
                waitingloads[i, j] :=0 ;
           end ;
      end ;
END ;
PROCEDURE FIRSTARRIVALS(var minimum : real ;
var nextarrival : mat4) ;
VAR
   i, j : integer ;
BEGIN
       minimum :=MAX_INT ;
       for i :=1 to SIZE do
           for j:=1 to SIZE do
               begin
                    nextarival;[i, j] :=expvariate(lambda[i, j]) ;
                    if nextarrival[i, j] < minimum
                      then minimum :=nextarrival[i, j] ;
                end ;
END ;
PROCEDURE FINDWAITINGLOADS(t:real ; var nextarrival : mat 4 ;
              var waitingloads : mat5; var maxqlength : mat7) ;
VAR
     i, j, qlength : integer ;
     t2, next : real ;
     checkload : integer ;
BEGIN
       checkload :=0 ;
```

```
            for i :=1 to SIZE do
            begin
                for j:=1 to SIZE do
                    begin
                        t2 :=nextarrival[i, j] ;
                        while t2<=t do
                       begin
                 waitingloads[i, j]:=waitingloads[i, j]+1 ;
                 next :=expvariate(lambda[i, j]) ;
                 t2 :=t2 + next ;
                        end ;
                           nextarrival[i, j]:t2 -t ;
                    end ;
                qlength:=0 ;
                for j:=1 to SIZE do
                    qlength :=qlength + waitingloads[i, j] ;
                if qlength > maxqlength[i] then
                  maxqlength[i]:=qlength ;
            end ;
                for j:=1 to SIZE do
                  checkload:=checkload + waitingloads[2, j] ;
                  {writeln(listing, TOTAL TIME=; total time : 8,
                  LOADS AT 2 floor=; ch}
END ;
PROCEDURE SET_TSP_MATRIX(lastfloor : integer) ;
VAR
    i, j : integer ;
BEGIN
      for i:=1 to SIZE do
          for j:=1 to SIZE do
              if waitingloads[i, j]=0 then costmatrix[i, j]:=MAX_INT
              else
                costmatrix[i, j]:=floortime[lastfloor, i]+
                floortime[i, j]+30 ;
```

```
END ;
PROCEDURE TSP(var lastfloor, itime, ptime:integer ; mat:mat6 ;
              var waitingloads:mat5 ; var nextarrival:mat4 ; var
              maxqlength:mat7 ; var noloads:boolean) ;
VAR
    i, j : integer ;
    min:array[1…SIZE] of integer ;
    labeli:array[1…SIZE, 1…SIZE] of integer ;
    maxi, maxj, max:integer ;
    found:boolenan ;
    nextentry:real ;
    u:integer ;
    nearestlongq:integer ;
    floorload:integer ;
FUNCTION MINROW(rowno:integer) : integer ;
VAR
    min : integer ;
    i:integer ;
BEGIN
      min:=MAX_INT ;
      for i:=1 to SIZE do
      if ((mat[rowno, i] > =0) and (mat[rowno, i]<min)) then
        min:=mat[rowno, i] ;
        minrow:=min ;
END ;
FUNCTION MINCOL(colno:integer) : integer ;
VAR
    min:integer ;
    j:integer ;
BEGIN
      min:=MAX_INT ;
      for j:=1 to SIZE do
       if ((mat[j, colno]>=0 and (mat[j, colno]<min)) then
       mincol:=min ;
```

```
END ;
BEGIN
      {itime:=0 ; ptime:=0}
       nearestlongq:=0 ;
       for i:=1 to SIZE do
       begin
            floorload:=0 ;
            for j:=1 to SIZE do
            floorload:=floorload+waitingloads[i, j] ;
       end ;
             found:=false ;
             for i:=1 to SIZE do
               for j:=1 to SIZE do
                 if waitingloads[i, j]<>0 then found:=true ;
                 if (not found) then
                   begin
                        noloads:=true ;
                        firstarrivals(nextentry nextarrival) ;
                        u:=round(nextentry)-1 ;
                        itime:=u ;
                        ptime:=0 ;
                        end
                        else
     begin
     for i:=1 to SIZE do min[i]:=minrow(i) ;
       for i:=1 to SIZE do
                  for j:=1 to SIZE do
                      if mat[i, j]>0 then
                        mat[i, j]:=mat[i, j] - min[i] ;
       for i:=1 to SIZE do min[i]:=mincol(i) ;
     for j:=1 to SIZE do
       for i:=1 to SIZE do
           if mat[i, j]>0 then
             mat[i, j]:=mat[i, j]-min[j] ;
       for i:=1 to SIZE do
```

```
            for j:=1 to SIZE do
                if mat[i, j]=0 then
                begin
                    mat[i, j]:=MAX_INT ;
                    labeli[i, j]:=minrow(i) + mincol(j) ;
                    mat[i, j]:=0 ;
            end ;
        max:=-1 ;
        for i:=1 to SIZE do
            for j:= 1 to SIZE do
                if mat[i, j]=0 then
                  if (labeli[i, j]>max) then
                    begin
                        max:=labeli[i, j] ;
                        maxi:=i ;
                        maxj:=j ;
                    end ;
        itime:=floortime[lastfloor, maxi] ;
        imove:=imove+abs(lastfloor-maxi) ;
        ptime:=floortime[maxi, maxj]+30 ;
        pmove:=pmove+abs(maxi + maxj) ;
        lastfloor :=maxj ;
        waitingloads[maxi, maxj]:=waitingloads[maxi, maxj]-1 ;
      end ;
END ;
PROCEDURE GREEDY(var lastfloor, itime, ptime:integer ;
        var waitingloads:mat5 ; var nextarrival:mat4 ; var
        maxqlength:mat7 ; var noloads:boolean) ;
VAR
    i, j:integer ;
    found:boolean ;
    maxi, maxj:integer ;
    g:char ;
    nextentry:real ;
    u:integer ;
    floorload:integer ;
```

```
BEGIN
      for i:=1 to SIZE do
   begin
       floorload:=0 ;
       for j:=1 to SIZE do
         floorload:=floorload+waitingloads[i, j] ;
   end ;
      i:=0 ;
      found:=false ;
      while (not found) and (i <= SIZE) do
         begin
            if (lastfloor-i)>=1 then
              begin
                 j:=1 ;
                 while (not found) and (j<=SIZE) do
                       if waitingloads [lastfloor-i, j]>0 then
                         begin
                              found:=true ;
                              maxi:=lastfloor-i ;
                              maxj:=j
                         end
                    else
                         j:=j+1 ;
              end;
            if (not found) and (lastfloor-i<=SIZE) then
              begin
                 j:=1;
                 while (not found) and (j<=SIZE) do
                     if waitingloads[lastfloor+i, j]>0 then
                       begin
                            found:=true ;
                            maxi:=lastfloor+i ;
                            maxj:=j
                       end
                     else
```

```
                                j:=j+1 ;
                    end ;
                i:=i+1 ;
end ;
    if (not found) then
      begin
            noloads:=true ;
            firstarrivals(nextentry, nextarrival) ;
            u:=round(nextentry)+1 ;
            itime:=u ;
            ptime:=0 ;
      end
    else
    begin
    itime:=floortime[lastfloor, maxi] ;
    imove:=imove+abs(lastfloor-maxi) ;
    ptime:=floortime[maxi, maxj]+30 ;
    pmove:=pmove+abs(maxi-maxj) ;
    lastfloor:=maxj ;
    waitingloads[maxi, maxj]:=waitingloads
                                [maxi, maxj]-1 ;
    end ;
end ;
begin{main}
      writeln('A') ;
      assign(listing, 'elevl') ;
      rewrite(listing) ;
      assign(datafile, : 'elevdata.pas') ;
      reset(datafile) ;
      writeln('SIMULATION APPLICATION__MS915') ;
      writeln ;
      writeln('Enter the Simulation Time in hours') ;
      readln(timel) ;
      time2:=(timel*60*60) ;
      writeln ;
```

```
writeln('Enter the initial Seed Values')
readln(seed) ;
idletime:=0 ;
prodtime:=0 ;
pmove:=0 ;
imove:=0 ;
for i:=1 to SIZE do ;
    maxqlength[i] :=0 ;
lastfloor:=i ;
writeln(listing) ;
writeln(listing) ;
writeln(listing, 'STMULATION ALLPICATION) ;
writeln(listing) ;
writeln(listing) ;
writeln(listing, '**********************') ;
writeln(listing, 'ELEVATOR SCHEDULING') ;
writeln(listing, '**********************') ;
writeln(listing) ;
writeln(listing, SEES=', seed) ;
writeln(listing) ;
readavg ;
initialize ;
firstentry:=MAX_INT ;
firstarrivals(firstentry, nextarrival) ;
totaltime:=0 ;
movetime:=firstentry ;
writeln('Enter T for scheduling using Travelling
        Salesman Technique and')
writeln('G for scheduling using Greedy algorithm') ;
        readln(a) ;
if a='T' then
begin
     writeln(listing, 'Scheduling using Travelling
            Salesman Technique') ;
```

```
if p=1 then writeln(listing  'with destinations
            computed aft( r 1 move')
 else
  writeln(listing, 'with des inations computed
        after', p, 'moves ) ;
 end
    else
        begin
            writeln(listing  'Scheduling using Greedy
                  Algori hm') ;
             if p=1 then writeln(listing, 'with
               destinatio: s computed after 1 move')
             else
             writeln(listir g, 'with destinations
                    com uted after', p, 'moves')
             end ;
 while(totaltim<=(time2)  do
 begin
FINDWAITINGLOADS(movetime, nextarrival, v aitingloads,
                                    maxqlength) ;
    movetime:=0 ;
    k:=0 ;
    noloads:=false ;
    repeat
        k:=k+1 ;
        if a='T' then
        begin
        set_tsp_matr x(lastfloor) ;
        tsp(lastfloor, i me, ptime, costmatrix,
            waitingloac s, nextarrival, maxqlength,
            noloads) ;
        end
        else
GREEDY(lastfloor, itime, ptime, waitingloads, r extarrival,
                maxqleng h, noloads) ;
```

```
                    idletime:=idletime+itime ;
                    prodtime:=prodtime+ptime ;
                    movetime:=movetime+itime+ptime ;
                    totaltime:=totaltime+itime+ptime ;
            until(k=p) or (noloads) ;
                writeln(totaltime) ;
            end ;
        writeln(listing) ;
        writeln(listing) ;
        writeln(listing, 'Productive time=', prodtime) ;
        writeln(listing, 'Idle time=', idletime) ;
        writeln(listing) ;
        totqlength :=0 ;
        for i:=1 to SIZE do
          begin
          writeln(listing, 'Max Queue length at floor', i, '=', maxqlength[i]:5) ;
          totqlength:=totqlength+maxqlength[i] ;
          end ;
          simtime :=totaltime/(60*60) ;
          writeln(listing) ;
          writeln(listing) ;
          writeln(listing, 'Simulated Time =', simtime:9:3, 'hours') ;
          writeln(listing) ;
          writeln(listing, 'No. of productive moves=', pmove) ;
          writeln(listing, 'No. of idle moves=', imove) ;
          writeln(listing) ;
          writeln(listing, 'Total Maximum Queue Length(at all
                floors)=', totqlength) ;
          close(listing) ;
          close(datafile) ;
END.
```

APPENDIX Ⅱ

SIMULATION

****

ALLEN BRADLEY --- Elevator Scheduling

****

SEED = 12

Scheduling using Traveling Salesman Technique
with destinations computed after 1 move
Productive time = 21507
Idle time = 7326

Max Queue length at floor 1 = 1
Max Queue length at floor 1 = 222
Max Queue length at floor 1 = 1
Max Queue length at floor 1 = 4
Max Queue length at floor 1 = 6
Max Queue length at floor 1 = 2
Max Queue length at floor 1 = 3
Max Queue length at floor 1 = 7
Max Queue length at floor 1 = 5

Simulated time = 8.009
No. of productive moves = 897
No. of idle moves = 1050
Total Maximum Queue Length (at all floors) = 21

SIMULATION

****

ALLEN BRADLEY --- Elevator Scheduling

****

SEED = 7

Scheduling using Greedy Algorithm
with destinations computed after 1 move

Productive time = 27627
Idle time = 1197

Max Queue length at floor 1 = 11
Max Queue length at floor 1 = 24
Max Queue length at floor 1 = 1
Max Queue length at floor 1 = 5
Max Queue length at floor 1 = 2
Max Queue length at floor 1 = 4
Max Queue length at floor 1 = 37
Max Queue length at floor 1 = 18
Max Queue length at floor 1 = 15

Simulated time = 8.007
No. of productive moves = 1662
No. of idle moves = 151
Total Maximum Queue Length (at all floors) = 117

References

[1] Aburdene M.F., *Computer Simulation of Dynamic Systems*, Wm. C. Brown, Dubuque, Iowa, 1988.

[2] Allen, A.O., *Probability, Statistics, and Queuing Theory with Computer Science Applications*, Academic Press, New York, 1978.

[3] Chase, R. and N. Aquilano, *Production and Operations Management*, 5th ed., Richard D. Irwin, Homewood, Ill., 1989.

[4] Cohen, J., *Statistical Power Analysis for the Behavior Sciences*, Lawrence Erlbaum, Hillsdale, NJ, 1988.

[5] Kraljic, P., "Purchasing Must Become Supply Management," *Harvard Business Review*, September-October 1983, pp.109-120.

[6] Law, A.M. and W.D. Chelate, *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1982.

[7] Little, J.D., C.I., Mouthy, D.W., Sweeney and C. Karrel, "An Algorithm for the Traveling Salesman Problem," *Operations Research*, Vol. 11(1963), pp. 972-989.

[8] MacDougall, M.H., *Simulating Computer Systems* : Techniques and Tools, MIT Press Cambridge, Mass., 1987.

[9] Newell, G.F., *Applications of Queuing Theory*, 2d ed., Chapman and Hall, London, 1982.

[10] Pedhazur, E.J., *Multiple Regression in Behavioral Research*, Holt, Rinehart & Winston, New York, 1982.

[11] Tersine, R.J., *Material Management and Inventory System*. 3rd ed., Elsevier North-Holl and Publishing, New York, 1987.

[12] Wyman, F.P., *Simulation Modeling:A Guide to Using SIMSRIPT*, Wiley, New York, 1970.