INTERNATIONAL SCIENTIFIC AND VOCATIONAL JOURNAL (ISVOS JOURNAL)

Vol.: 4 Issue: 1 Date: June 2020 Received: 15.04.2019 Accepted: 26.03.2020 Final Version: 28.06.2020

ISVOS Journal, 2020, 4(1): 1-11

A Tabu Search Algorithm for an Excavator Scheduling Problem

Elifcan Göçmen[a,1], Onur Derse[b]

[a] Department of Industrial Engineering, Faculty of Engineering, Munzur University, Tunceli, TURKEY,
ORCID ID: 0000-0002-0316-281X
[b] Department of Management and Organization, Logistics, Tarsus University, Adana, TURKEY,
ORCID ID: 0000-0002-4528-1999

Abstract

Global sector prompts the construction firms to give a priority to time and cost factors. Thus, scheduling of the jobs is focal important to achieve cost and time objectives. Scheduling problems have gained a great importance in recent years in the construction sector. We have examined a single machine scheduling problem for an excavator used in the construction sector. There are some jobs in which each job has a normal processing time, a due date, earliness penalty and tardiness penalty. This paper presents a meta-heuristic optimization algorithm named Tabu Search (TS) to minimize the total cost and provides how it can be used to solve a wide variety of single machine scheduling problems. Computational results demonstrates that the proposed approach is a good tool for these problems.

Keywords: "Tabu search, single machine scheduling problem, excavator, construction sector"

1. Introduction

Construction sector requires taking complex decisions by considering time and cost. Managing these decisions properly is related with effective planning and scheduling steps. The planning step is generally to determine the resource types. Scheduling step consists of a collection of jobs which are to be scheduled sharing limited resources. The aim of the scheduling problem is to complete all the activities of the jobs satisfying time and cost constraints. Single-machine scheduling problems are very common in practice [1]. The scheduling of a certain number of jobs on a single machine has gained importance for theoretical and practical reasons. In the single machine scheduling problems, the timing and end dates of work are handled in a single machine, in which the business processes can be controlled by the allocation of a common resource and the non-linear problems of the process. The aim is to ensure optimal resource allocation so that all works can be completed, deadlines and the total resource consumption can be minimized [2].

There are many real-world situations involving a single resource used for accomplishing several tasks. Construction sector is a good area to apply the single machine scheduling problems. Construction projects need to finish the various tasks considering due dates. Optimization algorithms have been used widely in the last 20 years [3] and new evolution-inspired algorithms for optimization have been developed [4]. In this paper, we develop a Tabu search-based solution procedure designed for an exactor machine scheduling including several tasks.

The Tabu search begins by local minima. To avoid repeating the steps used, the method records recent moves in one or more tabu lists by tabu search memory. The role of the memory is defined by the algorithm. Some related problems have been studied and are described in the following literature review. The authors studied the one machine scheduling problem with release dates, deadlines, the possibility to reject some jobs, and sequence dependant setup times. The objective is to maximize the revenue, which is the sum of the gains associated with each performed job minus a weighted tardiness penalty. This problem differs from (P) by the fact that setup costs are not taken into account, and in (P) we consider general cost functions. The authors propose a MILP (mixed integer linear programming) formulation for the problem, which is able to solve instances with up to 15 jobs, as well as different heuristics [5].

Local search methods are often effective for scheduling problems. Several neighbourhood structures are studied to solve the single machine scheduling problem to minimize the sum of delay and setup costs. The neighbourhood Reinsert consists taking a job in the schedule and moving it to another position. The neighbourhood Swap consists of swapping two jobs in the schedule.

[1] Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: elifcangocmen@munzur.edu.tr

They conclude that Reinsert is better than Swap, but that using an Hybrid neighbourhood yields better results. Hybrid is the union of Swap and Reinsert [6].

There are many studies dealing with the scheduling problem for a single machine. Two mixed binary integer programming models are proposed that could handle 9 and 10 jobs, respectively, to solve this problem and minimize the average flow time [7-8]. The problem of programming jobs with setup times in a single machine is addressed to minimize the maximum delay. To solve this problem, they developed a new integer programming formulation [9]. The simulation annealing algorithm is developed and compared it to the mathematical model results in order to complete the works in minimum time, to have a minimum of delay times and to make the completion of the works in the best time [10]. A mathematical model and heuristic algorithm in the single machine scheduling problem is presented [11].

A a two-layered algorithm is proposed based on the tabu search algorithm in their work. The result of the calculation shows that when the results obtained are compared, the algorithm can produce optimal or near optimal solution for large problems in an acceptable calculation period [2]. There are other studies dealing with the single machine scheduling problem with heuristic algorithms. A single machine scheduling problem is provided that aims to minimize flexible availability and minimize total completion time [12]. Tabu search approach is focused since it has a single machine problem with several hundred jobs and many families and it is difficult to solve this problem [13]. A single heuristic algorithm that includes artificial bee colony algorithm was used in the problems of the table examined as delayed penalties. The aim is to minimize the sum of delay penalties [14]. A taboo algorithm was used to provide optimal results for grouping and sorting jobs in a single machine environment [15]. A single machine scheduling problem is examined for the most appropriate workflow that made a minimum cost [16]. The single machine schedule problem is investigated with intuitive algorithm by considering the costs of waiting and late [17]. In addition, A single machine scheduling problem is handled with an intuitive algorithm [18-22]. The single machine scheduling problem is interested with the possibility of not performing some jobs. A global deadline is given and jobs cannot be scheduled after it. Moreover, the processing time of each job can be reduced by a compression. The objective function to maximize is the profit of each performed job minus compression costs and tardiness costs. After having introduced a timing algorithm for the problem permitting to find, given a sequence of jobs, the optimal processing time of each job, they introduced two heuristics for the problem. The first heuristic is a GRASP algorithm where a schedule is built by a randomized dispatching rule, then the compression level of each job is obtained by applying the timing algorithm. The second heuristic is an approximation algorithm obtained by the adaptation of an algorithm for the intervals selection problem [23].

2. Material- Method

This paper introduces an algorithm for the single-machine total weighted earliness–tardiness scheduling problem. N jobs (job 1, . . . , job *n*) are to be processed on a single machine that can process at most one job at a time. No pre-emption is allowed and once the machine starts processing a job, it cannot be interrupted. After the machine finishes processing a job, it can be idle even when there exist unprocessed jobs. Each job *j* is given a processing time C_j, due date d_j and release date *Bj*, where $dj \geq Bj + Cj$. It is also given a tardiness weight α_i and a earliness weight β_i, *a* unit cost of idle time *γ, idle time before initializing of j job* W_j. The earliness *Ei* and the tardiness *Ti* are defined by $Ei = \max(dj - Cj - Bj, 0)$, $Ti = \max(Ci - dj + Bj, 0)$, Our objective is to find a schedule that minimizes total cost, $(\alpha jTj + \beta jEj + \gamma W_j)$.

We can give an example for our approach in Table 1. The vector B has start dates (days) of orders. On vector each index indicates the days, the value on the each index also indicates which order is being operated. For example, $B_2=0$; 0.order's processing was begun in the second day. $B_i=-1$ means that in the i.day, no order was begun for processing, but it doesn't mean in the i. day the machine was idle. $C_0=2$ means that 0, order's processing begins in the second day and finishes in the third day. So, $B_3=-1$ shows that in the third day the machine still works.

Table 1. Example of B vector for 5 days and two orders

i	*0*	*1*	*2*	*3*	*4*
B_i	*-1*	*-1*	*0*	*-1*	*1*

W_i=max{0, initializing day of the order i- initializing day of the before i. order-the processing time of the order before order i}

For example; W_0=max{0,2-0-0}=2(idle time)

The dimension of the B vector is the sum of processing times of all orders and it is symbolized with the *numday* variable.

$$numday=\begin{cases}\sum_{i\in j} C_i < \max\{d_0,...d_{j-1}\}=\max\{d_0....d_{j-1}\}\\ \sum_{i\in j} C_i > \max\{d_0,...d_{j-1}\}=\sum_{i\in j} C_i\end{cases}$$

The machine can't operate the two orders in the same time. A variable called *overlap* is used. If this variable is 0, it means that in the same time two orders aren't being operating. Otherwise, the variable will have the value 1.

Overlap function;

the initializing day of the order before existing order+ the processing time of the order before existing order>the initializing day of the existing order=1

the initializing day of the order before existing order+ the processing time of the order before existing order<=the initializing day of the existing order=0

In this paper one tabu structure is used which is used for showing of tabu moves. Tabu search algorithm is applied to the excavator scheduling problem. The flow chart of the problem is shown in Figure 1.

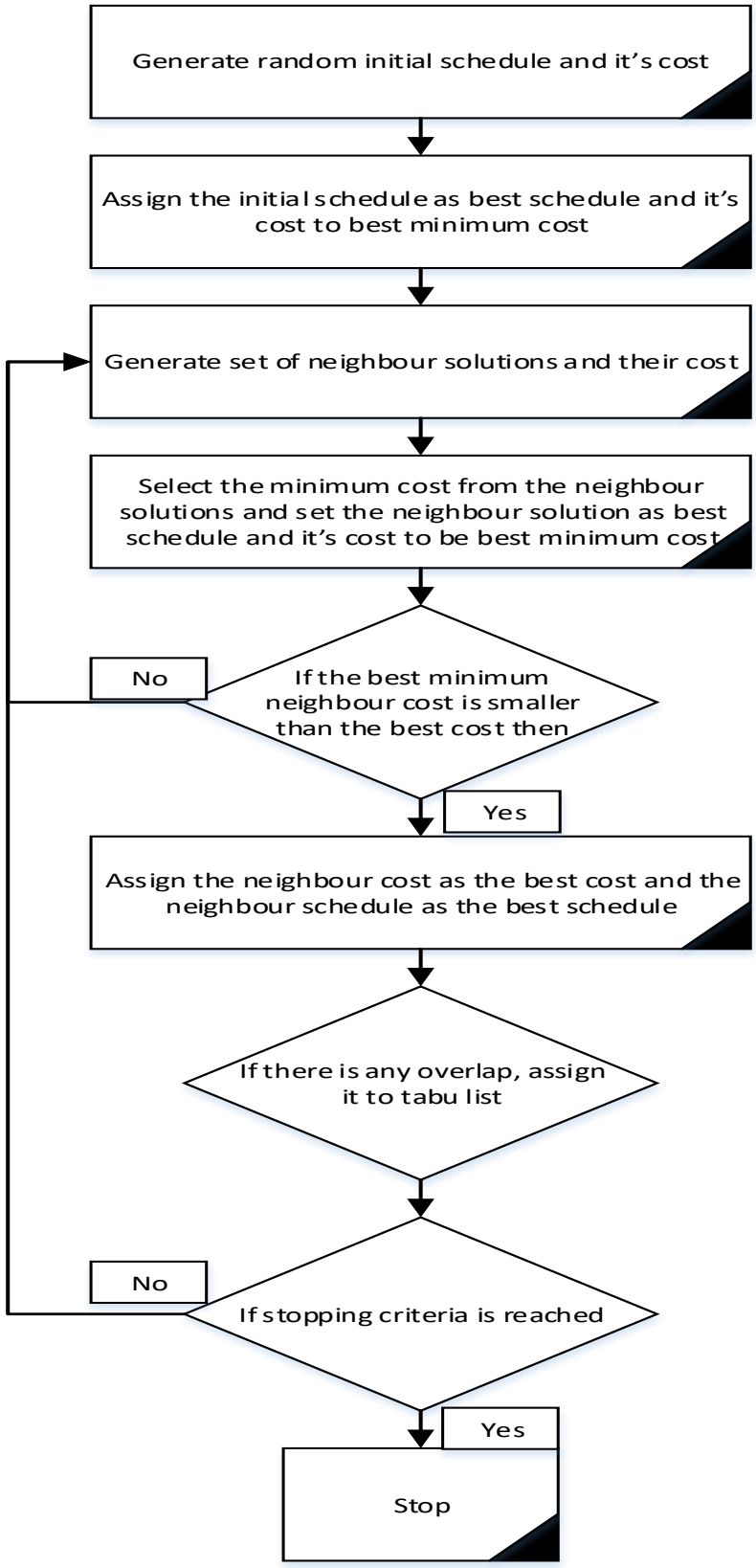

Figure 1. Flowchart of the proposed tabu search algorithm

3. Results

In our algorithm one tabu list is used in the tabu search. The overlaps for each iteration are forbidden for the next determined number of iteration which is five in our algorithm. Each iteration a number of neighbour solution is evaluated and attend of each iteration if the best neighbour solution is better than the best solution, it is assigned to be new best solution. The stopping criteria is that if a new better result is not generated in the last determined number of iteration or it is reached to number of iteration then the algorithm will be stopped. For this purpose the iteration number is given as 1200, in each iteration 400 neighbour solution is evaluated and 100 iteration is determined as stopping criteria.

When the search finished, the best cost was obtained as 76.66 at the iteration number 878. The resulting schedule doesn't contain any overlap.

In Table 2, job schedules are illustrated. Since each job was not completed by due date, tardiness and earliness penalty costs were assigned. The change in the evaluated costs is illustrated in Figure 2. The total cost is decreased by the iteration number.

Table 2. The results of the job sequences

Job sequence	Due date	Processing time	Tardiness cost	Earliness cost
0	1	1	8.61	1.01
1	28	4	3.19	0.81
2	38	3	0.82	2.37
3	8	4	0.50	1.47
4	92	2	7.75	1.64
5	70	4	7.18	1.53
6	17	2	4.66	1.23
7	83	2	4.82	0.75
8	88	2	7.73	4.88
9	50	5	8.27	0.10
10	15	2	5.01	0.11
11	60	1	7.74	3.25
12	77	4	5.58	1.03
13	68	3	9.55	3.22
14	100	2	6.76	1.48
15	10	4	4.95	4.41
16	51	3	9.53	3.42
17	35	1	7.05	4.94
18	78	4	1.98	0.72
19	91	4	5.89	4.32

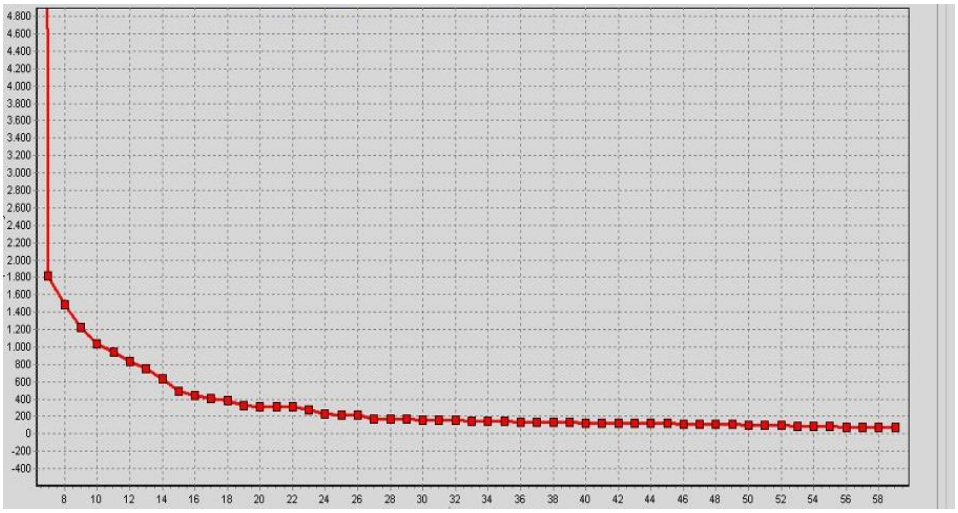

Figure 2. The change in the evaluated costs

The source code of the software is given in Appendix.

4. Conclusion

Time and cost are important parameters for the construction firms. All tasks must be finished considering due dates and customer demands. Therefore, machine, personnel and the other resources should be scheduled. Scheduling problems have been studied at the construction sector by both of the academics and practitioners. We investigated the single machine called "excavator" scheduling problem used in the construction sector. The jobs assigned to the excavator considering normal processing time, a due date, earliness penalty and tardiness penalty were scheduled. A meta-heuristic optimization algorithm, TS was provided to minimize the total cost. Total cost is decreased by our algorithm and we gave insights for the other scheduling problems. For further works, larger problems including multi machines could be solved by hybrid heuristic approaches.

References

[1] M. Ben-daya and M. Al-Fawzan, "An efficient Tabu search algorithm for the single machine mean tardiness problem". Production Planning and Control, vol. 8, no, 7, 1997.

[2] K. Xu, Z. Feng, and K. Jun, "A tabu-search algorithm for scheduling jobs with controllable processing times on a single machine to meet due-dates", Computers & Operations Research, vol. 37, no. 11, pp. 1924–1938, 2010.

[3] V. Ateş, and N. Barışçı, "Short-term load forecasting model using flower pollination algorithm", International Scientific and Vocational Studies Journal, vol. 1, no. 1, pp. 22-29.2017.

[4] İ. B. Koç, A. Al Janadi, and V. Ateş, "Interlock optimization of an accelerator using genetic algorithm", International Scientific and Vocational Studies Journal, vol. 1, no. 1, pp. 30-41, 2017.

[5] C. Oguz, F. Salman, and Z. Yalçın, "Order acceptance and scheduling decisions in make-to-order systems", International Journal of Production Economics, vol. 125, pp. 200–211, 2010.

[6] M. Laguna, J. W. Barnes, and F. Glover, "Tabu search methods for a single machine scheduling problem", Journal of Intelligent Manufacturing, vol. 2, pp. 63–74, 1991.

[7] J. S. Chen, "Using integer programming to solve the machine scheduling problem with a flexible maintenance activity", Journal of Statistics and Management Systems, vol. 9, no. 1, pp. 87–104, 2006.

[8] J. S. Chen, "Optimization models for the machine scheduling problem with a single flexible maintenance activity", Engineering Optimization, vol. 38, no. 1, 53–71, 2006.

[9] O. Hinder, and A. J. Mason, "A novel integer programing formulation for scheduling with family setup times on a single machine to minimize maximum lateness", European Journal of Operational Research, vol. 262, no. 2, pp. 411-423, 2017.

[10] A. Ghodratnama, M. Rabbani, R. Tavakkoli-Moghaddam, and A. Baboli, "Solving a single-machine scheduling problem with maintenance, job deterioration and learning effect by simulated annealing", Journal of Manufacturing Systems, vol 29, no. 1, pp. 1-9, 2010.

[11]V. Kayvanfar, I. Mahdavi, G.M. Komaki, "Single machine scheduling with controllable processing times to minimize total tardiness and earliness", Computers & Industrial Engineering, vol. 65, no. 1, pp. 166-175, 2013.

[12] I. Adiri, J. Bruno, E. Frostig, A. H. G. Rinnooy Kan, "Single machine flow time scheduling with a single breakdown", Acta Informatica, vol. 26, no. 7, pp. 679–696, 1989.

[13] F. Jin, J. N. Gupta, S. Song, C. Wu, "Single machine scheduling with sequence-dependent family setups to minimize maximum lateness", Journal of the Operational Research Society, vol. 61, no. 7, pp. 1181-1189, 2010.

[14] A. Yurtkuran and E. Emel, "A discrete artificial bee colony algorithm for single machine scheduling problems", International Journal of Production Research, vol. 54, no. 22, pp. 6860-6878, 2016.

[15] Y. Suppiah, and M. K. Omar, "A hybrid tabu search for batching and sequencing decisions in a single machine environment", Computers & Industrial Engineering, vol. 78, pp. 135-147, 2014.

[16] M. T. Almeida, and M. Centeno, “A composite heuristic for the single machine early/tardy job scheduling problem”, Computers & Operations Research, vol. 25, no. 7-8, pp. 625–635, 1998.

[17] J. M. S Valente, and R. A. F. S. Alves, “Heuristics for The Single Machine Scheduling Problem with Quadratic Earliness and Tardiness Penalties”, Computers & Operational Research, vol. 35, pp. 3696–3713, 2008.

[18] F. Della Croce, and V. T'kindt, “A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem”, Journal of the Operational Research Society, vol. 53, no. 11, pp. 1275-1280, 2002.

[19] A. Jouglet, D. Savourey, J. Carlier, and P. Baptiste, “Dominance-based heuristics for one-machine total cost scheduling problems”, European Journal of Operational Research, vol. 184, no. 3, pp. 879-899, 2008.

[20] M. Vila, and J. Pereira, “Exact and heuristic procedures for single machine scheduling with quadratic earliness and tardiness penalties”, Computers & Operations Research, vol. 40, no. 7, pp. 1819-1828, 2013.

[21] F. Della Croce, F. Salassa, and V. T'kindt, “ A hybrid heuristic approach for single machine scheduling with release times”, Computers & Operations Research, vol. 45, pp. 7-11, 2014.

[22] R. M’Hallah, and A. Alhajraf, “Ant colony systems for the single-machine total weighted earliness tardiness scheduling problem”, Journal of Scheduling, vol. 19, no. 2, pp. 191-205, 2016.

[23] B. Yang, J. Geunes, and W. J. O'Brien, “A heuristic approach for minimizing weighted tardiness and overtime costs in single resource scheduling”, Computers & Operations Research, vol. 31, no. 8, pp. 1273-1301, 2004.

Appendix

The initialization of the tabu search algorithm:

```
procedure TForm1.Button1Click(Sender: TObject);
var
i,r: integer;
begin
SetLength(j,MemoJ.Lines.Count);
SetLength(d,MemoD.Lines.Count);
SetLength(c,MemoC.Lines.Count);
SetLength(alfa,MemoAlfa.Lines.Count);
SetLength(e,MemoE.Lines.Count);
for i:=0 to MemoJ.Lines.Count-1 do
  begin
   j[i]:=StrToInt(MemoJ.Lines[i]);
   d[i]:=StrToInt(MemoD.Lines[i]);
   c[i]:=StrToInt(MemoC.Lines[i]);
   alfa[i]:=StrToFloat(MemoAlfa.Lines[i]);
   e[i]:=StrToFloat(MemoE.Lines[i]);
  end;
  gunsay:=0;
  sayac:=0;
  for i:=0 to MemoJ.Lines.Count-1 do
    begin
      gunsay:=gunsay+c[i];
      if sayac<d[i] then
         sayac:=d[i];
    end;
    if sayac>gunsay then
       gunsay:=sayac;
//ShowMessage(IntToStr(gunsay));
MemoEE.Lines.Clear;
MemoAA.Lines.Clear;
MemoCC.Lines.Clear;
MemoDD.Lines.Clear;
```

```
MemoJJ.Lines.Clear;
PageControl1.ActivePageIndex:=1;
for i:=0 to MemoJ.Lines.Count-1 do
  begin
    MemoEE.Lines.Add(MemoE.Lines[i]);
    MemoAA.Lines.Add(MemoAlfa.Lines[i]);
    MemoCC.Lines.Add(MemoC.Lines[i]);
    MemoDD.Lines.Add(MemoD.Lines[i]);
    MemoJJ.Lines.Add(MemoJ.Lines[i]);
    Application.ProcessMessages;
  end;
SGisCizelge.ColCount:=gunsay;
tvy1.ColCount:=gunsay;
tvy1.RowCount:= MemoJ.Lines.Count;
for i:=0 to tvy1.ColCount-1 do
  for r:=0 to tvy1.RowCount-1 do
    tvy1.Cells[i,r]:='0';
for i:=0 to gunsay-1 do
  begin
    SGisCizelge.Cells[i,0]:=IntToStr(i);
    SGisCizelge.Cells[i,1]:='-1';
  end;
while MemoJJ.Lines.Count<>0 do
  begin
    Randomize;
    r:= RandomRange(0,gunsay-1);
    if SGisCizelge.Cells[r,1]='-1' then
       begin
          SGisCizelge.Cells[r,1]:=MemoJJ.Lines[0];
          MemoJJ.Lines.Delete(0);
       end;
    Application.ProcessMessages;
  end;
for i:=0 to MemoJ.Lines.Count-1 do
  begin
    MemoJJ.Lines.Add(MemoJ.Lines[i]);
    Application.ProcessMessages;
  end;
SGisSirasi.ColCount:=MemoJ.Lines.Count;
r:=0;
for i:=0 to gunsay-1 do
    if SGisCizelge.Cells[i,1]<>'-1' then
      begin
        SGisSirasi.Cells[r,0]:=IntToStr(r);
        SGisSirasi.Cells[r,1]:=IntToStr(i);
        SGisSirasi.Cells[r,2]:=SGisCizelge.Cells[i,1];
        r:=r+1;
      end;
TotalCost:=0;
for i:=0 to SGisSirasi.ColCount-1 do
 TotalCost:=TotalCost+cost(i);
SetLength(best_schedule,gunsay);

TabuSearch;
 end;
```

Cost calculation for each job is given as:

```
function TForm1.cost(isno: integer): real;
var
OncekiGun,SimdikiGun,SonrakiGun: integer;
OncekiISno,SimdikiISno,SonrakiISno: integer;
i,r: integer;
ustuste: integer;
Tardiness, Earliness,IdleTime: integer;
```

```
begin
for i:=0 to SGisSirasi.ColCount-1 do
  if StrToInt(SGisSirasi.Cells[i,2])=isno then
    begin
    r:=i;
    SimdikiGun:=StrToInt(SGisSirasi.Cells[i,1]);
    SimdikiISno:=isno;
    end;
if r<>0 then
  begin
    OncekiGun:=StrToInt(SGisSirasi.Cells[r-1,1]);
    OncekiISno:=StrToInt(SGisSirasi.Cells[r-1,2]);
  end;
if r<>SGisSirasi.ColCount-1 then
  begin
    SonrakiGun:=StrToInt(SGisSirasi.Cells[r+1,1]);
    SonrakiISno:=StrToInt(SGisSirasi.Cells[r+1,2]);
  end;
ustuste:=0;
if r<>0 then
  begin
    if OncekiGun+StrToInt(MemoC.Lines[OncekiIsno])>SimdikiGun then
      begin
      ustuste:=100000;
      if tvy1.Cells[SimdikiGun,isno]='0' then
        tvy1.Cells[SimdikiGun,isno]:=inttostr(5);
      end
    else
      ustuste:=0;
  end;
Tardiness:=Max(0,SimdikiGun+StrToInt(MemoC.Lines[SimdikiISno])-StrToInt(MemoD.Lines[SimdikiISno]));
Earliness:=max(0,StrToInt(MemoD.Lines[SimdikiISno])-SimdikiGun-StrToInt(MemoC.Lines[SimdikiISno]));
if r<>0 then
 IdleTime:=Max(0,SimdikiGun-OncekiGun-StrToInt(MemoC.Lines[SimdikiISno]))
else
  Idle Time:=0;
Result:=StrToFloat(MemoAlfa.Lines[SimdikiISno])*Tardiness+StrToFloat(MemoE.Lines[SimdikiISno])*Earliness+1*IdleT
ime+ustuste;
end;
```

The tabu search algorithm is given as:

```
procedure TForm1.TabuArama;
var
i,j,r,temp: integer;
a,b: integer;
itersay, iterfark, adaysay: integer;
gun1, gun2: integer;
tempsira:real;
begin
for i:=0 to gunsay-1 do
  eniyi_cizelge[i]:=StrToInt(SGisCizelge.Cells[i,1]);
eniyi_Cost:=ToplamCost;
for i:=0 to SGAdaylar.RowCount do
   begin
     SGAdaylar.Cells[0,i]:='';
     SGAdaylar.Cells[1,i]:='';
     SGAdaylar.Cells[2,i]:='';
   end;
for i:=0 to SGCost.RowCount do
     SGCost.Cells[0,i]:='';
SGCost.RowCount:=StrToInt(IterSayisi.Text);
SGAdaylar.RowCount:=StrToInt(AdaySayisi.Text);
SGGunler.RowCount:=StrToInt(AdaySayisi.Text);
```

```
SGCost.Cells[0,0]:=FloatToStr(ToplamCost);
itersay:=0;
iterfark:=0;
while (itersay=StrToInt(IterSayisi.Text)) or (iterfark<StrToInt(Edit1.Text)) do
//for itersay:=1 to StrToInt(IterSayisi.Text) do
  begin
itersay:=itersay+1;
    adaysay:=0;
    while adaysay<>StrToInt(AdaySayisi.Text) do
       begin
       Randomize;
       gun1:=RandomRange(0,gunsay-1);
       Randomize;
       gun2:=RandomRange(0,gunsay-1);
       Application.ProcessMessages;
       if (gun1<>gun2) and (SGisCizelge.Cells[gun1,1]<>SGisCizelge.Cells[gun2,1]) and (TabuCheck(gun1,gun2)=true) then
         begin
          SGAdaylar.Cells[0,adaysay]:=SGisCizelge.Cells[gun1,1];
          SGAdaylar.Cells[1,adaysay]:=SGisCizelge.Cells[gun2,1];
          SGGunler.Cells[0,adaysay]:=IntToStr(gun1);
          SGGunler.Cells[1,adaysay]:=IntToStr(gun2);
          Application.ProcessMessages;
          temp:=StrToInt(SGisCizelge.Cells[gun1,1]);
          SGisCizelge.Cells[gun1,1]:=SGisCizelge.Cells[gun2,1];
          SGisCizelge.Cells[gun2,1]:=IntToStr(temp);
          SGisSirasi.ColCount:=MemoJ.Lines.Count;
           r:=0;
          for i:=0 to gunsay-1 do
             if SGisCizelge.Cells[i,1]<>'-1' then
               begin
                 SGisSirasi.Cells[r,0]:=IntToStr(r);
                 SGisSirasi.Cells[r,1]:=IntToStr(i);
                 SGisSirasi.Cells[r,2]:=SGisCizelge.Cells[i,1];
                 r:=r+1;
               end;
          ToplamCost:=0;
          for i:=0 to SGisSirasi.ColCount-1 do
            begin
           ToplamCost:=ToplamCost+cost(StrToInt(SGisSirasi.Cells[i,2]));
            end;
          SGAdaylar.Cells[2,adaysay]:=FloatToStr(ToplamCost);
          Application.ProcessMessages;
          temp:=StrToInt(SGisCizelge.Cells[gun1,1]);
          SGisCizelge.Cells[gun1,1]:=SGisCizelge.Cells[gun2,1];
          SGisCizelge.Cells[gun2,1]:=IntToStr(temp);
          SGisSirasi.ColCount:=MemoJ.Lines.Count;
          r:=0;
          for i:=0 to gunsay-1 do
             if SGisCizelge.Cells[i,1]<>'-1' then
               begin
                 SGisSirasi.Cells[r,0]:=IntToStr(r);
                 SGisSirasi.Cells[r,1]:=IntToStr(i);
                 SGisSirasi.Cells[r,2]:=SGisCizelge.Cells[i,1];
                 r:=r+1;
               end;
          adaysay:=adaysay+1;
        end;  // if (gun1<>gun2) and ...
       end;
    Application.ProcessMessages;
    for a:=0 to adaysay-2 do
      for b:=a+1 to adaysay-1 do
         if StrToFloat(SGAdaylar.Cells[2,b])<StrToFloat(SGAdaylar.Cells[2,a]) then
           begin
```

```
            tempsira:=StrToFloat(SGAdaylar.Cells[2,b]);
            SGAdaylar.Cells[2,b]:=SGAdaylar.Cells[2,a];
            SGAdaylar.Cells[2,a]:=FloatToStr(tempsira);
            tempsira:=StrToFloat(SGAdaylar.Cells[1,b]);
            SGAdaylar.Cells[1,b]:=SGAdaylar.Cells[1,a];
            SGAdaylar.Cells[1,a]:=FloatToStr(tempsira);
            tempsira:=StrToFloat(SGAdaylar.Cells[0,b]);
            SGAdaylar.Cells[0,b]:=SGAdaylar.Cells[0,a];
            SGAdaylar.Cells[0,a]:=FloatToStr(tempsira);
            tempsira:=StrToFloat(SGGunler.Cells[0,b]);
            SGGunler.Cells[0,b]:=SGGunler.Cells[0,a];
            SGGunler.Cells[0,a]:=FloatToStr(tempsira);
            tempsira:=StrToFloat(SGGunler.Cells[1,b]);
            SGGunler.Cells[1,b]:=SGGunler.Cells[1,a];
            SGGunler.Cells[1,a]:=FloatToStr(tempsira);
            Application.ProcessMessages;
            end;
    iterfark:=iterfark+1;
    if StrToFloat(SGAdaylar.Cells[2,0])<eniyi_Cost then
       begin
         eniyi_Cost:=StrToFloat(SGAdaylar.Cells[2,0]);
         iterfark:=1;
           temp:=StrToInt(SGisCizelge.Cells[strtoint(SGGunler.Cells[0,0]),1]);
          SGisCizelge.Cells[strtoint(SGGunler.Cells[0,0]),1]:=SGisCizelge.Cells[strtoint(SGGunler.Cells[1,0]),1];
           SGisCizelge.Cells[strtoint(SGGunler.Cells[1,0]),1]:=IntToStr(temp);
         for i:=0 to gunsay-1 do
             eniyi_cizelge[i]:=StrToInt(SGisCizelge.Cells[i,1]);
         i:=0;
         while SGCost.Cells[0,i]<>'' do
           i:=i+1;
         SGCost.Cells[0,i]:=FloatToStr(eniyi_Cost);
         if i>8 then SGCost.TopRow:=i-7;
       end;
   for i:=0 to tvy1.ColCount-1 do
     for r:=0 to tvy1.RowCount-1 do
       if strtoint(tvy1.Cells[i,r])>0 then
         tvy1.Cells[i,r]:= inttostr(strtoint(tvy1.Cells[i,r])-1);
Label22.Caption:='Costte son iyileşme '+ IntToStr(iterfark)+' iterasyon önce oldu.';
  end; //İterasyon döngüsü
i:=0;
while  SGCost.Cells[0,i]<>'' do
 begin
   Series1.Add(StrToFloat(SGCost.Cells[0,i]));
   i:=i+1;
 end;
end;
```

The tabu list check function is as follows:

```
function TForm1.TabuCheck(g1,g2: integer): boolean;
var
x,y: integer;
begin
Result:=true;
x:=strtoint(SGisCizelge.Cells[g2,1]);
y:=strtoint(SGisCizelge.Cells[g1,1]);
if x>-1 then
   if StrToInt(tvy1.Cells[g1,x])>0 then
      Result:=False;
if y>-1 then
   if StrToInt(tvy1.Cells[g2,y])>0 then
      Result:=False;

end;
```