Volume 9, No. 3, March - 2018 (Special Issue)
International Journal of Mathematical Archive-9(3), 2018, 28-42
Available online through www.ijma.info ISSN 2229 – 5046

STUDY ON, TO FIND THE CRASHING TIME & THEIR COST OF THE NETWORK USING C PROGRAMMING

S. SHANMUGASUNDARAM
Assistant Professor, Department of Mathematics,
Government Arts College (A), Salem - (TN), India.

V. MATHAN KUMAR
Assistant Professor, Department of Mathematics,
Sri Ganesh College of Arts and Science, Salem - (TN), India.

ABSTRACT

In this paper discuss a crashing method for time cost tradeoff problem and the solution using C programming. It is an alternative method to find the optimum duration and their cost. Crashing time as well as cost is a major task for the management to find various cost and time. The objective of crashing is to reduce the project duration with cost. If we have data for given program to find the optimum duration cost, this program produces those results in system (not-manual).This approach will support the construction management to schedule their decision and the project.

Key words: *Critical path method, Time-cost tradeoff, Crashing, C Programming.*

1. INTRODUCTION

Critical Path Method scheduling has proven to be an effective project management tool (3). The critical path (CPM) is commonly implemented the critical management to determine project's life. The CPM identifies a route which is necessary for timely completion of a given project (4).

Crashing is reducing project time by expending additional resources crashing the network is the contracting or compressing the network that means to reduce project duration at minimum cost with minimum project duration.(6)

Crashing is the procedure by which project duration can be shorten up by expediting selective activities within the project. But it requires allocating more resources than usual to compress an activity's duration, which in turns increases the budget of that activity. So, crashing is basically a time-cost trade-off by which specific deadline can be achieved (7).

Time and cost are two main criteria that should be considered carefully in project scheduling. Project managers are always trying to determine an economic time in which total cost of project would be minimum (1). The objective of time-cost trade-off analysis is reducing the original project duration with possible least total cost (2).

The application of project scheduling was for analysis & evaluation of mechanized greenhouse construction project using CPM with WINQSB software (8).

The critical path method has been programmed for UNIVAC I, 1103A & 1105 with a census Bureau configuration. These programs were prepared so that either UNIVAC I or the 1100 series computers used independently with one another (5).

The algorithm proposed for unit crashing reduces the cost of project. When activities are crashed by one day then only the crashing cost corresponding to one day is increased by reducing the project duration as well as cost. A C++ program is been developed to achieve the results. This approach is well suitable for places where cost is major consideration (9).

CONFERENCE PAPER
National Conference dated 26th Feb. 2018, on "New Trends in Mathematical Modelling" (NTMM - 2018), Organized by Sri Sarada College for Women, Salem, Tamil Nadu, India.

In this paper, first chapter presents an introduction, second chapter explains the computational procedure how to determine the values to solve the problem, third chapter presents an application of the model on hypothetical project with an example data applied in a program, fourth chapter explains the algorithm how to apply crashing method, fifth chapter presents the program for obtaining the solution, sixth chapter shows analysis and results with a sample output and conclusions are given in the last chapter.

2. COMPUTATIONAL PROCEDURE

Here, we have written C Programming Language to compute the optimum duration and cost of the project. In this study, the procedure consists of many variables to complete the problem. The Variables are

1) Normal Cost: It is the lowest cost of completing an activity in minimum time employing normal means that is not using overtime or special resources. The variable is ncost.

2) Normal Time: It is minimum time required to achieve the normal cost. The variable is n time.

3) Crash Cost: It is a least cost of completing an activity by employing all possible means like overtime, additional machinery and proper materials. The variable is ccost.

4) Crash Time: It is an absolute minimum time associated with the crash cost. The variable is ctime.

5) Critical Path: It is the sequence of project network terminal elements with the longest overall duration, determining the shortest time to complete the project.

spath[t]=source[i] and
dpath[t]=des[i], where as i=0 to n-1 and slack[i]=0

6) The earliest start time (estime[]): the earliest time at which an activity can start if no delays occur in the project.

7) The earliest finish time (eftime[]): The earliest time at which an activity can finish if no delays occur in the project.

8) The latest start time (lstime[]): The latest time at which an activity can start without delaying the completion of the project.

9) The latest finish time (lftime[]): The latest time at which an activity can finish without delaying the completion of the project.

10) Maximum Reduction duration (MRT[]): Maximum allowable crash duration for activity

11) Total project duration (tdur): Project completion duration computed using critical path method (CPM).

$$\text{tdur} = \sum_{m=0}^{n-1} ntime[m] \quad \text{,where as slack[m]=0.}$$

12) Indirect cost (incost): Project overhead/indirect cost. This is a fixed monetary value expended each duration unit the project elapses.

13) Slack (slack[]): The slack time for an activity refers to the length of time that can be tolerated without incurring a delay in the scheduled project completion time. The slack time per activity needs to be calculated first to identify the critical path(s), by considering either the start time or finish time.

slack[i]=lftime[i]-eftime[i], where as i=0 to n-1.

14) **Cost slope**: The relationship between normal cost, crash cost , normal time and crash time is assumed to be linear. Hence, for each activity a crash cost per period can be derived as follows

slope[i]= (ccost[i]-ncost[i])/mrt[i], where as i=0 to n-1.

3. APPLICATION OF THE MODEL ON HYPOTHETICAL PROJECT

An example project whose activities, activities normal time, crash time, normal cost and crash cost values are assigned by using C program like, if source[0] and destination[0] is equal to 1 and 2 , then the ntime[0] and ctime[0] will be 3, 2 respectively as well as all the activities ncost [i], ccost [i] values are assigned.

Activity	Duration (days)		Cost(Rs.)	
	Normal	Crash	Normal	Crash
1-2	3	2	3000	4000
2-3	3	3	300	300
2-4	7	5	4200	5800
2-5	9	7	7200	8100
3-5	5	4	2500	3000
4-5	0	0	0	0
5-6	6	4	3200	4100
6-7	4	3	4000	4700
6-8	13	10	7800	9000
7-8	10	9	10000	12000
Indirect Cost = Rs. 500 per day				

Table-1: Project Information Table

Based on the formulae, lstime, lftime, estime and eftime to be calculated as well as the resultant values are assigned. Critical path is identified based on the slack value of each activity. If the slack of activity is zero then the critical path exists in that activity.

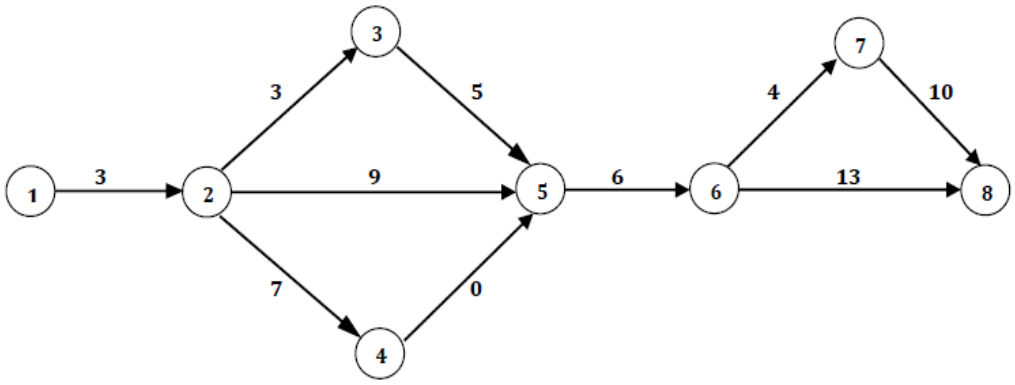

Figure-1: Activity Network diagram

In our example,

slack[0]=lftime[0]-eftime[0] i.e slack[0]=3-3=0 the path exists on 1→2
slackl[3]=lftime[3]-eftime[3] i.e slack[1]=12-12=0 the path exists on 2→5
slackl[6]=lftime[6]-eftime[6] i.e slack[6]=18-18=0 the path exists on 5→6
slackl[7]=lftime[7]-eftime[7] i.e slack[7]=22-22=0 the path exists on 6→7
slackl[9]=lftime[9]-eftime[9] i.e slack[7]=32-32=0 the path exists on 7→8

	Path	Length
1	1-2-3-5-6-7-8	3+3+5+6+4+10=31
2	1-2-3-5-6-8	3+3+5+6+13=30
3	1-2-5-6-7-8	3+9+6+4+10=32
4	1-2-5-6-8	3+9+6+13=31
5	1-2-4-5-6-7-8	3+7+0+6+4+10=30
6	1-2-4-5-6-8	3+7+0+6+13=29

Table-2: Path Table

And the corresponding project completion duration is 32 days. Under the normal circumstance the cost of this project is obtained from the program by retrieving from the variable tcost which is calculated as:

$$\text{sum} = \sum_{i=0}^{n-1} n\cos t[i]$$

tcost = sum+(incost * tdur)

4. APPLYING CRASHING METHOD

The procedure for shortening the project duration can be summarized in the following steps:

Step-1: Declaring variables for normal time, crash time, crash cost, normal cost, earliest start time, earliest finish time, latest start time, latest finish time, slope, mslope and so on.

Step-2: Assigning values to ntime, ctime, ncost and ccost according to their source path to destination path.

Step-3: Compute the value for earliest start time, earliest finish time, latest start time, latest finish time, slack and also cost slope for all activities.

Step-4: Perform CPM calculations and identity the critical path, using slack value which is equal to zero and find the total duration of the project.

Step-5: Start by shortening the activity duration on the critical path which has the least cost slope and not been shortened to its crash duration.

Step-6: Reduce the duration of the critical activities with least slope until its crash duration is reached or until the critical path changes.

Step-7: If the parallel path exists, it compute the sum of critical path activity's cost slope with parallel path activity's least cost slope value then this sum value is compared to critical path activity's next cost slope value to find the smallest value. After the comparison, the smallest value activity is crashed.

Step-8: Having shortened a critical path, adjust timings and floats.

Step-9: The cost increase due to activity shortening is calculated as the cost slope multiplied by the time units shortened

Step-10: Continue until no further shortening is possible, and then the crash point is reached

5. THE PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int cost1,cost2,cost3,z1,x1,x2,k1,f1,mrtp1,source1,des1,csource1,cdes1small1;
int nsstack[10],ndstack[10],nslope[10],dstack[10],fdur1[20],otdur;
int csource[10],cdes[10],cmrt[10],cslope[10],cctime[10],ccsource,ccdes,sstack[10],;
int source[20],des[20],ntime[20],np[20],ns[20],msource[20],mdes[20];
int ntime1[20],ntime2[20],fscost,mr,ft=0,newsum=0,js,fsum=0,temp3,com1,com2,com3;
void path(int spath2[20][20],int dpath2[20][20],int n2,int nop1[20]);
int final(int source[],int des[],int ntime[],int spath2[20][20],int path2[20][20],int nop1[20],int n2,int tdur,int ctime[],int
fscost);
int plength1(int source[],int des[],int ntime[],int spath3[],int dpath3[],int nop,int n2);
int final1(int source[],int des[],int ntime[],int spath2[20][20],int dpath2[20][20],int nop1[20],int n2,int tdur,int
ctime[],int fscost);
int minimum(int m[],int v);
double minimum1(double m[],int v);
int spath3[20],dpath3[20],fm,ctime[20],r,de,dc[20],dd,ts,slope1[20],ssum=0;
int estime[20],eftime[20],lstime[20],lft,spath[20],dpath[20],start,finish,sp[20];
float slope[20],mrt[20],sum,mrt1[20],smrt[20],com,mmrt[20];
int b,a,d,c,g,new3[20],min2[20],slack[20],add[20],tdur,f,count,mslope[20];
int n1,nop,n2,i,j,k,m,p,x,y,temp,e,z,h,es[20],min[20],new2[20],s,u,q,temp1,sum1,net1;
int spath2[20][20],dpath2[20][20],net2[20],cspath[20],cdpath[20],cnet[20],count1;
int ntslope=0,ntctime=0,mrtslope=0,mrtslope1,diff,ctdur[20],ctcost[20],cr,maxccost;
int lastsource[20],lastdes[20],lastmrt[20],ntime1[20],sum2,fd=0,fdur[20];
int fo,n,fmax,ps[20],paths,spath4[20],dpath4[20],temp2,ccount=0,sf,xy,cfs,nop1[20];
float ncost[20],ccost[20],incost;
int fslope[20],cost,com4,mtime[20],mslope1[20],msource1[20],mdes1[20];
double tcost,crcost,ftcost[20],opt;
int su[10],mm,lasts[20],lastd[20],lastsource1[20],lastdes1[20],lastmrt1[20];
```



```
int lastmslope1[20],lastmslope2[20],fmslope1[20],fmslope2[20],t;
int cr1,xb,lastmsource2[20],lastmdes2[20],v,l,cost1,count1;
void main()
{ int new1[20],max[20],lftime[20];
  int w,l,t;
  s1=0; crcost=0;
  clrscr();
  printf("\nEnter total number of source : ");
  scanf("%d",&n);
  printf("\nEnter the source values one by one :");
  for(i=0;i<n+1;i++)
  {  scanf("%d",&source[i]); }
  printf("\nEnter destination values one by one :");
  for(i=0;i<n;i++)
  { printf("\n%d--> :",source[i]);
    scanf("%d",&des[i]);   }
  printf("\nEnter normal time values one by one :");
  for(i=0;i<n;i++)
  { printf("\n%d-->%d :",source[i],des[i]);
    scanf("%d",&ntime[i]);  }
  printf("\nEnter crash time values one by one :");
  for(i=0;i<n;i++)
  { printf("\n%d-->%d :",source[i],des[i]);
    scanf("%d",&ctime[i]);  }
  printf("\nEnter normal cost values one by one :");
  for(i=0;i<n;i++)
  { printf("\n%d-->%d :",source[i],des[i]);
    scanf("%f",&ncost[i]); }
  printf("\nEnter crash cost values one by one :");
  for(i=0;i<n;i++)
  {  printf("\n%d-->%d :",source[i],des[i]);
     scanf("%f",&ccost[i]); }
  crcost=0;
  for(i=0;i<20;i++)
  { for(j=0;j<20;j++)
    { spath2[i][j]=0; dpath2[i][j]=0;}   }
    printf("\nEnter no of other paths in the network : ");
    scanf("%d",&n2);
    for(i=0;i<n2;i++)
    {  printf("\nEnter no of nodes in %d path : ",i+1);
       scanf("%d",&nop1[i]);
       for(j=0;j<nop1[i];j++)
       { printf("\nEnter the value of source : ");
              scanf("%d",&spath2[i][j]);
              printf("\nEnter the value of destination : ");
              scanf("%d",&dpath2[i][j]);
       } }
  for(i=0;i<n;i++)
  {  count=0;
     for(j=0;j<n;j++)
     {       if(source[i]==des[j])
             {   count++; np[i]=count;   }} }
  for(i=0;i<n;i++)
  {  count=0;
     for(j=0;j<n;j++)
     {       if(des[i]==source[j])
             {   count++; ns[i]=count;}} }
  for(i=0;i<n;i++)
  {  if(np[i]==0)  { estime[i]=0;  }
     else if(np[i]==1)
     { for(j=0;j<n;j++)
       {if(source[i]==des[j])
              {  estime[i]=estime[j]+ntime[j];      }}}
```



```
    else
    { if(np[i]>1)
    { k=0; temp=source[i];
           for(j=0;j<n;j++)
           { if(temp==des[j])
             {   new1[k]=source[j]; k++;  }}
     x=0;
     for(w=0;w<n;w++)
     { for(p=0;p<k;p++)
            {  if((source[w]==new1[p])&&(des[w]==temp))
               {  max[x]=estime[w]+ntime[w]; x++; } }}
           for(t=0;t<n;t++)
           {if(source[t]==temp)
             estime[t]=maximum(max,x);
           }}}}
for(i=0;i<n;i++)
{   eftime[i]=estime[i]+ntime[i];
  if(i==n-1)
  { de=des[i];
    for(d=0;d<n;d++)
    { if(de==des[d])
     {dd=eftime[d];dc[y]=dd;y++; }}
    lft=maximum(dc,y);}}
 for(i=n-1;i>=0;i--)
 {  if(ns[i]==0)
    {      lftime[i]=lft;    }
    else if(ns[i]==1)
    { for(j=n;j>=0;j--)
     {if(des[i]==source[j])
           {   lftime[i]=lftime[j]-ntime[j];}}}
    else
    {  if(ns[i]>1)
     { d=0; temp1=des[i];
    for(m=n;m>=0;m--)
           { if(des[i]==source[m])
             {  new2[d]=des[m]; d++; }}
           s=0;
           for(p=n-1;p>=0;p--)
           { for(b=0;b<d;b++)
             {   if(source[p]==new2[b])
                {  min[s]=lftime[p]-ntime[p]; s++; }}}
            x=0;
           for(z=n-1;z>=0;z--)
           { if(source[z]==temp1)
             {    min2[x]=lftime[z]-ntime[z]; x++; }}
           for(t=n-1;t>=0;t--)
           {if(des[t]==temp1)
           {  lftime[t]=minimum(min2,x);}}}}}
    // latest starting
    for(i=0;i<n;i++)
    {       lstime[i]=lftime[i]-ntime[i];   }
    // slack
    for(i=0;i<n;i++)
    { slack[i]=lftime[i]-eftime[i]; }
    // MRT
    for(i=0;i<n;i++)
    {   mrt[i]=ntime[i]-ctime[i]; }
    // slope
    for(i=0;i<n;i++)
    {  add[i]=ccost[i]-ncost[i];
           if((add[i]==0)&&(mrt[i]==0)){ slope[i]=0;}
           else
           slope[i]=(ccost[i]-ncost[i])/mrt[i];
```



```
    }
 printf("\nsource des  ntime np ns estime eftime lftime lstime slack ncost ccost ctime  MRT slope ");
 for(i=0;i<n;i++)
 {
printf("\n%4d%4d%4d%4d%4d%4d%4d%4d%4d%4d%4d%7.0f%7.0f%4d%6.2f%8.2f",source[i],des[i],ntime[i],np[i],ns[i
],estime[i],eftime[i],lftime[i],lstime[i],slack[i],ncost[i],ccost[i],ctime[i],mrt[i],slope[i]);
 }
 printf("\nSlope of critical path : ");
 count=0;
 for(i=0;i<n;i++)
 {  if((slack[i]==0)&&(slope[i]!=0))
    { count++; count1++;mslope[count-1]=slope[i];
      msource[count-1]=source[i];mdes[count-1]=des[i];
      mmrt[count-1]=mrt[i];mtime[count-1]=ntime[i];   } }
for(i=0;i<count;i++)
 { for(j=i+1;j<count;j++)
   { if(mslope[i]>mslope[j])
     {
       temp=mslope[i];temp1=msource[i];temp2=mdes[i];temp3=mtime[i];
       mslope[i]=mslope[j];msource[i]=msource[j];mdes[i]=mdes[j];mtime[i]=mtime[j];
       mslope[j]=temp;msource[j]=temp1;mdes[j]=temp2;mtime[j]=temp3;  }}}
for(d=0;d<count;d++)
  { if(mslope[d]==mslope[d+1])
    { if(mmrt[d]>mmrt[d+1])
com=mmrt[d+1];com1=msource[d+1];com2=mdes[d+1];com3=mslope[d+1];com4=mtime[d+1];mmrt[d+1]=mmrt[d];
msource[d+1]=msource[d];mdes[d+1]=mdes[d];mslope[d+1]=mslope[d];mtime[d+1]=mtime[d];mmrt[d]=com;msourc
e[d]=com1;mdes[d]=com2;mslope[d]=com3;mtime[d]=com4;}}}
  for(i=0;i<count;i++)
  {  mslope1[i]=mslope[i];msource1[i]=msource[i];mdes1[i]=mdes[i];}
  printf("\n\nEnter indirect cost : ");
  scanf("%f",&incost);
  // total normal cost
  for(i=0;i<n;i++)
  { sum=sum+ncost[i]; }
  printf("\n\nTotal normal cost : %8.0f",sum);
  printf("\n\n\n\ncritical path : ");
  f=0;t=0;
  for(i=0;i<n;i++)
  { if(slack[i]==0)
    {  spath[t]=source[i];dpath[t]=des[i];f++;t++;
       printf("    %d-->%d",source[i],des[i]);  } }
     tdur=0;
     for(i=0;i<n;i++)
     { if(slack[i]==0)
       {  tdur=tdur+ntime[i];} }
     ctdur[cr]=tdur; fdur[fd]=tdur; fd++;
     printf("\n\nTotal duration : %d",tdur);
      path(spath2,dpath2,n2,nop1);
      maxccost=plength(source,des,ntime,spath2,dpath2,nop1,n2);
      if(tdur>=maxccost)
      { tcost=sum+(incost*tdur); ftcost[ft]=tcost; ft++;   }
      for(i=0;i<n;i++)
      {  mrt1[i]=mrt[i]; }
      sf=0;
             for(i=0;i<n;i++)
             { slope1[i]=slope[i]; }
     for(cr=0;cr<count;cr++)
     {        for(x=0;x<n;x++)
       {
if((slope1[x]==mslope[cr])&&(slack[x]==0)&&(source[x]==msource[cr])&&(des[x]==mdes[cr]))
             mrtslope=mrt[x];
             cost=slope[x];} }
    z=0;
```



```
  while(z<mrtslope)
  { for(x=0;x<n;x++)
   {
        if((slope1[x]==mslope[cr])&&(slack[x]==0)&&(source[x]==msource[cr])&&(des[x]==mdes[cr]))
        {   if(ntime[x]>ctime[x])
            {       ntime[x]=ntime[x]-1;  }
           cost=slope[x];}}
   tdur=0;
   for(i=0;i<n;i++)
   {  if(slack[i]==0) { tdur=tdur+ntime[i];  }}
   ctdur[cr]=tdur;
   printf("\nTotal duration1 : %d",tdur);
   maxccost=plength(source,des,ntime,spath2,dpath2,nop,n2);
    if(tdur>=maxccost)
    { tcost=sum+(tdur*incost)+(1*mslope[cr])+crcost;
         crcost=crcost+(1*mslope[cr]);ftcost[ft]=tcost; ft++;
         fdur[fd]=tdur; fd++;
      if((sstack[0]==0)&&(dstack[0]==0))
         { goto zloop; }
          else        goto cr1loop;  }
    else
    {     for(p=0;p<n2;p++)
             {       sum1=0;
                     for(q=0;q<nop1[p];q++)
                     {for(i=0;i<n;i++)
               { if((source[i]==spath2[p][q])&&(des[i]==dpath2[p][q]))
               {    sum1=sum1+ntime[i]; }}}
              ntime1[p]=sum1;
if(maxccost==ntime1[p])
{ a=0;
 for(j=0;j<nop1[p];j++)
 { for(k=0;k<n;k++)
   {
if((source[k]==spath2[p][j])&&(des[k]==dpath2[p][j])&&(slack[k]!=0)&&(ntime[k]>ctime[k])){
         csource[a]=source[k]; cdes[a]=des[k]; cmrt[a]=mrt[k];
         cslope[a]=slope[k]; cctime[a]=ntime[k]; a++;
         }//if }//k  }//j  }//if }//p
if((a==0)&&(cmrt[a-1]!=0))
{ for(l=0;l<n;l++)
  {   if((source[l]==csource[a-1])&&(des[l]==cdes[a-1]))
         { ntime[l]=ntime[l]-1;  }  }
         tcost=sum+(tdur*incost)+(1*mslope[cr])+crcost+slope[l];
         crcost=crcost+(1*mslope[cr]);
         ftcost[ft]=tcost; ft++; fdur[fd]=tdur; fd++;
  }
else
{  f=minimum(cslope,a);
   for(i=0;i<a;i++)
   { if(f==cslope[i])
     { ccsource=csource[i]; ccdes=cdes[i]; ccslope=cslope[i];  } }
   for(i=0;i<n;i++)
   { if((source[i]==ccsource)&&(des[i]==ccdes))
     { ntime[i]=ntime[i]-1; } }
   maxccost=plength(source,des,ntime,spath2,dpath2,nop,n2);
   if(tdur>=maxccost)
    { if(cr!=count-1)
         { if((mslope[cr]+ccslope)<mslope[cr+1])
           {         tcost=sum+(tdur*incost)+(1*mslope[cr])+crcost+ccslope;
                     crcost=crcost+(1*mslope[cr])+ccslope;
                     ftcost[ft]=tcost; ft++; fdur[fd]=tdur; fd++;
                     if((sstack[0]==0)&&(dstack[0]==0))
                     { goto zloop; }
                     Else
```



```
                { goto cr1loop; }
            }
            else
            {       for(x=0;x<n;x++){
          if((slope1[x]==mslope[cr])&&(slack[x]==0)&&(source[x]==msource[cr])&&(des[x]==mdes[cr]))
          { ntime[x]=ntime[x]+1; sstack[s1]=source[x]; dstack[s1]=des[x]; s1++; }
          }
          for(i=0;i<n;i++)
          {     if((source[i]==ccsource)&&(des[i]==ccdes))
               {  ntime[i]=ntime[i]+1; } }
          goto crloop;          }//else   }//if
          else
          {             tcost=sum+(tdur*incost)+(1*mslope[cr])+crcost+ccslope;
                        crcost=crcost+(1*mslope[cr])+ccslope;
                        ftcost[ft]=tcost; ft++; fdur[fd]=tdur; fd++;
                        goto crloop;
          }//else    }//if       }//else    }//else
           cr1loop:
          for(cr1=0;cr1<s1;cr1++)
          { for(x=0;x<n;x++)
           {    if((source[x]==sstack[cr1])&&(des[x]==dstack[cr1]))
               { mrtslope1=ntime[x]-ctime[x];cost1=slope[x]; }//if
           }//x
           for(z1=0;z1<mrtslope1;z1++)
           {   for(x=0;x<n;x++)
               { if((source[x]==sstack[cr1])&&(des[x]==dstack[cr1]))
                        {
                           if(ntime[x]>ctime[x])
                           { ntime[x]=ntime[x]-1;source1=source[x];
                                   des1=des[x];cost1=slope[x];
                           }//if  }  //if }//x
                      tdur=0;
                      for(i=0;i<n;i++)
                      { if(slack[i]==0)
                        { tdur=tdur+ntime[i];  }
                      }
                      ctdur[cr]=tdur;
                      maxccost=plength(source,des,ntime,spath2,dpath2,nop,n2);
                      if(tdur>=maxccost)
                      {    tcost=sum+(tdur*incost)+slope[cr1]+crcost;
                           crcost=crcost+slope[cr1];
                           ftcost[ft]=tcost; ft++;
                           fdur[fd]=tdur; fd++;
                      }
                      else
                      {for(p=0;p<n2;p++)
                 {sum1=0;
                      for(q=0;q<nop1[p];q++)
                      {
                      for(i=0;i<n;i++)
                      { if((source[i]==spath2[p][q])&&(des[i]==dpath2[p][q]))
                        {    sum1=sum1+ntime[i];  }
                      }
                      }
                      ntime1[p]=sum1;
                       if(maxccost==ntime1[p])
                       {a=0;
                        for(j=0;j<nop1[p];j++)
                        { for(k=0;k<n;k++)
                {
if((source[k]==spath2[p][j])&&(des[k]==dpath2[p][j])&&(slack[k]!=0)&&(ntime[k]>ctime[k]))     {
                       csource[a]=source[k];csource1=source[k];
                       cdes1=des[k];cdes[a]=des[k];
```



```
                cmrt[a]=ntime[k]-ctime[k];cslope[a]=slope[k];
                cost2=slope[k];cctime[a]=ntime[k];
                goto c3loop;
            }//if
          }//k
        }//j
       }//if
       }//p
       c3loop:
       cost3=0;
       for(f1=0;f1<count;f1++)
       {for(x1=0;x1<s1;x1++)
        {if((sstack[x1]==msource[f1])&&(dstack[x1]==mdes[f1]))
         {  for(x2=cr;x2<count;x2++)
           {  for(k1=0;k1<n;k1++)
           { if((source[k1]==msource[x2])&&(des[k1]==mdes[x2]))
                { mrtp1=ntime[k1]-ctime[k1];
                  if(mrtp1==0)
                  {  goto k1loop;  }
                  else
                  {  cost3=mslope[x2];
                    goto cost3l;
                  }
                  }k1loop:}}}} }
                  cost3l:
       if((cost1+cost2)>cost3)
       {  for(x=0;x<n;x++)
         {  if((source[x]==sstack[cr1])&&(des[x]==dstack[cr1]))
                {   ntime[x]=ntime[x]+1;}
         }  //x
         goto zloop;
       }
       else
       { for(l=0;l<n;l++)
         {  if((source[l]==csource1)&&(des[l]==cdes1))
           { ntime[l]=ntime[l]-1;   }
         }
         tdur=0;
         for(m=0;m<n;m++)
         { if(slack[m]==0)
           { tdur=tdur+ntime[m];   }
         }
ctdur[cr]=tdur;
maxccost=plength(source,des,ntime,spath2,dpath2,nop,n2);
if(tdur>=maxccost)
{       tcost=sum+(tdur*incost)+crcost+cost1+cost2;
        crcost=crcost+cost1+cost2;
        ftcost[ft]=tcost;ft++;
        fdur[fd]=tdur;fd++;
        for(i=0;i<s1;i++)
        {   if((sstack[i]==sstack[cr1])&&(dstack[i]==dstack[cr1]))
           {for(k=0;k<n;k++)
            {if((source[k]==sstack[cr1])&&(des[k]==dstack[cr1]))
             {if(ntime[k]>ctime[k])
                {   goto zloop; }
                else
                {   sstack[i]=0; dstack[i]=0; }
             }
             }
             }
             }
 }
  Else
```



```
    {for(i=0;i<n;i++)
          {  if((source[i]==source1)&&(des[i]==des1))
            {   ntime[i]=ntime[i]+1;  }
          }
          for(i=0;i<n;i++)
          {
           if((source[i]==csource1)&&(des[i]==cdes1))
           {
            ntime[i]=ntime[i]+1;
           }
          }
         }//if
         }//else
         }//else
   }//z1
   }//else
zloop:
z++;
}//z
crloop:
}//cr
    printf("\n\tDuration        Total cost\n");
    for(i=0;i<fd;i++)
    {       printf("\n\t   %d\t\t%7.0lf",fdur[i],ftcost[i]);  }
    opt=minimum1(ftcost,fd);
    p=0;
    for(k=0;k<fd;k++)
    { if(ftcost[k]==opt)
      {  fdur1[p]=fdur[k]; p++;  }
    }
    otdur=minimum(fdur1,p);
    printf("\nOptimum cost : %6.0lf",opt);
    printf("\nDuration = %d",otdur);
    getch();
}
 int final(int source[],int des[],int ntime[],int spath2[20][20],int dpath2[20][20],int nop1[20],int n2,int tdur,int
ctime[],int fscost)
  {
    int su[20],sum1=0,i,j,p,q,h,max=0,d,l,u,t,z=0,maxtemp,sum2=0,zl=0;
  i=0;j=0,g=0;
  for(p=0;p<n2;p++)
    {
     sum1=0;
     for(q=0;q<nop1[p];q++)
     {for(i=0;i<n;i++)
          { if((source[i]==spath2[p][q])&&(des[i]==dpath2[p][q]))
            {  smrt[q]=mrt[i];sum1=sum1+ntime[i]; }
          }}
           su[g]=sum1; ps[g]=p; g++;
          }
          fsum=0;
          for(js=0;js<n2;js++)
          {          if(su[js]==tdur) { goto el; }
          else if(su[js]>tdur)
          {
             t=0; ssum=0;
             for(x=0;x<n;x++)
             {
                    for(u=0;u<nop1[js];u++)
                    {
             if((source[x]==spath2[js][u])&&(des[x]==dpath2[js][u]))//&&(slack[x]!=0))
               {
                    ssum=ssum+ntime[x];
```



```
            }}
           }
           if(ssum==tdur){goto el; }
           else
           {
           for(x=0;x<n;x++)
           {
                  for(u=0;u<nop1[js];u++)
                  {
 if((source[x]==spath2[js][u])&&(des[x]==dpath2[js][u])&&(slack[x]!=0))
            { fslope[t]=slope[x];smrt[t]=mrt[x];spath4[t]=spath2[js][u];
            dpath4[t]=dpath2[js][u];t++; }}
           }
   for(i=0;i<t;i++)
   {
 for(j=i+1;j<t;j++)
 {
  if(fslope[i]>fslope[j])
  { temp=fslope[i];temp1=spath4[i];temp2=dpath4[i];temp3=smrt[i];
    fslope[i]=fslope[j];spath4[i]=spath4[j];dpath4[i]=dpath4[j];smrt[i]=smrt[j];
    fslope[j]=temp;spath4[j]=temp1;dpath4[j]=temp2;smrt[j]=temp3;
  }
 }
}
for(d=0;d<t;d++)
{ if(fslope[d]==fslope[d+1])
  { if(smrt[d]>smrt[d+1])
   { com=smrt[d+1];com1=spath4[d+1];com2=dpath4[d+1];com3=fslope[d+1];
     smrt[d+1]=smrt[d];path4[d+1]=spath4[d];dpath4[d+1]=dpath4[d];fslope[d+1]=
     fslope[d];smrt[d]=com;spath4[d]=com1;dpath4[d]=com2;fslope[d]=com3;
    }
  }
}
 for(g=0;g<t;g++)
 { for(k=0;k<n;k++)
   { if((source[k]==spath4[g])&&(des[k]==dpath4[g]))
     {if(slope[k]==fslope[g])
          {if(ctime[k]==ntime[k])
            { goto gloop;  }
            else if((ntime[k]>ctime[k])&&((fslope[g]+fscost)<fmslope2[xy]))
            {    ntime[k]=ntime[k]-1;
                  fsum=fsum+slope[k]; newsum=0;
                  for(x=0;x<n;x++)
                  {
                    for(y=0;y<nop1[js];y++)
                    {
                     if((source[x]==spath2[js][y])&&(des[x]==dpath2[js][y]))
                     {  newsum=newsum+ntime[x];  }
                    }
                  }
       if(newsum==tdur)
            {        goto el;   }
            else
            {  goto gloop; }
            }
            else if((ntime[k]>ctime[k])&&((fslope[g]+fscost)>fmslope2[xy]))
            { if(fmslope2[xy]==0)
              {  ntime[k]=ntime[k]-1;
                  fsum=fsum+slope[k];
                  newsum=0;
                  for(x=0;x<n;x++)
                  {  for(y=0;y<nop1[js];y++)
                    { if((source[x]==spath2[js][y])&&(des[x]==dpath2[js][y]))
```



```
                        { newsum=newsum+ntime[x];  }
                      }
                     }
              if(newsum==tdur)
              { goto el; }
              else
              { goto gloop; }
             }
             else
             { goto gloop; }
             }
            }
      }
     }
     gloop:continue;
    }
   }
  }
  el:continue;
 }
 ed:printf("\n\n\texit");
 return(fsum+fscost);
}
void path(int spath2[20][20],int dpath2[20][20],int n2,int nop1[20])
{
   int p,q;
   for(p=0;p<n2;p++)
   { ts=nop1[p]; }
}
int plength1(int source[],int des[],int ntime[],int spath3[],int dpath3[],int nop,int n2)
{ int sum1,i,j,p,q,max=0;
  i=0;j=0;sum1=0;
      for(i=0;i<n;i++)
      { for(p=0;p<n2;p++)
       { ts=nop1[p];
             for(q=0;q<ts;q++)
             { if((source[i]==spath2[p][q])&&(des[i]==dpath2[p][q]))
               {   sum1=sum1+ntime[i];  }
             }
      }  }
  ntime1[j]=sum1;  j++;
  max=maximum(ntime1,n2);
    return(max);
  }
int plength(int source[],int des[],int ntime[],int spath2[20][20],int dpath2[20][20],int nop,int n2)
{
  int sum1,i,j,p,q,max=0;
  i=0;j=0;
    for(p=0;p<n2;p++)
    { sum1=0;
      for(q=0;q<nop1[p];q++)
      {for(i=0;i<n;i++)
             {  if((source[i]==spath2[p][q])&&(des[i]==dpath2[p][q]))
               { sum1=sum1+ntime[i]; }
             }
      }
  ntime1[j]=sum1; j++;
  }
    max=maximum(ntime1,n2);
    return(max);
  }
 int maximum(int m[],int v)
  {  int big,i;
```



```
   big=m[0];
   for(i=0;i<v;i++)
   { if(big<m[i])  big=m[i]; }
   return(big);
 }
 int minimum(int m[],int v)
 {  int small,i;
   small=m[0];
   for(i=0;i<v;i++)
   { if(small>m[i]) small=m[i];  }
   return(small);
 }
 double minimum1(double m[],int v)
 { double small;
  int i,ff;
   small=m[0];
   for(i=0;i<v;i++)
   { if(small>m[i]) small=m[i];}
   return(small);
 }
```

6. ANALYSIS AND RESULTS

The purpose of this study was to explore the method of crashing which will not only take into account the activities on the critical path, but will also consider the non-critical activities, which in their turn become critical as the project time increased. A program was written in C programming language and run on an example data set. In Table 3 the results shows the Normal duration of 32 days for the cost of project is Rs.58200, Minimum duration of 25 days for the cost of project is Rs.61100 and the optimum duration of 29 days for the cost of project is Rs.58050.

OUT PUT	
DAYS	**AMOUNT**
32	**58200**
31	58150
30	58100
29	**58050**
28	58250
27	58700
26	59200
25	**61100**

Table-3: Program output

The implementation of the program showed more efficient and reliable results and generated a considerable savings along with an increase in robustness.

7. CONCLUSION

C programming is one of the tool find the crashing schedule and their cost. Here, I present programmatic approach to find the crashing time and cost. The example is also given to show the feasibility of the approach.

REFERENCES

1. Amin Zeinalzadeh, *"An Application of Mathematical Model to Time-cost Trade off Problem (Case Study)"*, Australian Journal of Basic and Applied Sciences, 5(7): 208-214, 2011, ISSN: 1991-8178.
2. Biswas S K, Karmaker C L & Biswas T K, *"Time-Cost Trade-off Analysis in a Construction Project Problem: Case Study"*, International Journal of Computational Engineering Research (IJCER), Vol.6, Issue 10, October 2016, ISSN (e): 2250-3005.
3. Boong Yeol Ryoo & Mike T. Duff, *"Understanding the Occurrence of Two Total Floats in One Activity and Schedule Crashing Approaches for that Situation"*, Journal of Building Construction an Planning Research, 2013,1, 67-74.
4. Harmeet Singh, Joon-Yeoul Oh, Kai Jin, Hua Li & Hee Joong Yang, *"Optimizing College Degree Plan"*, International Journal of Engineering Research & Technology (IJERT), Vol.3, Issue 11, November-2014, ISSN: 2278-0181.

5. James E.Kelley & Morgan R.Walker, *"Critical –Path Planning and Scheduling"*, 1959 Proceedings of the Eastern joint Computer Conference.
6. Komesh Sahu & Meenu Sahu, *"Cost & Time and Also Minimum Project Duration Using Alternative Method"*, International Review of Applied Engineering Research, Vol.4, Number 5 (2014), pp.403-412, ISSN: 2248-9967.
7. Nafish Sarwar Islam, *"Complex Project Crashing Algorithm"*, IOSR Journal of Business and Management (IOSR-JBM), Vol.11, Issue 4, Jul-Aug.2013, pp 10-17,e-ISSN:2278-487X.
8. Nasim Monjezi, Mohammad Javad Sheikhdavoodi, Hadi Basirzadeh &Hassan Zakidizaji, *"Analysis and Evaluation of Mechanized Greenhouse Construction Project using CPM Methods"*, Research Journal of Applied Sciences, Engineering and Technology, 4(18):3267-3273, 2012, ISSN: 2040-7467.
9. Priti Singh, Florentin Smarandache, Dipti Chanhan & Amit Bhaghel, *"A Unit Based Crashing PERT Network for Optimization of Software Project Cost"*, Jan-2009,https://www.research gate.net/publication/268444442.
10. Shanmugasundaram S & Mathan Kumar V, "*Programmatic approach for finding the project duration and associated cost*", Indian Journal of Applied Research, Vol-7, Issue-4, April-2017, ISSN: 2249-555X.

Source of support: National Conference on "New Trends in Mathematical Modelling" (NTMM - 2018), Organized by Sri Sarada College for Women, Salem, Tamil Nadu, India.

