"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XIV – 2011 – Issue 2
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania

THE DETERMINATION OF THE OPTIMAL PATHS WITH MINIMUM MULTIPLICATION IN DIRECTED GRAPH THAT HAVE THE VALUE OF THE ARCS OBTAINED FROM THE DIJKSTRA ALGORITHM

Paul VASILIU[1]
[1]Lecturer Ph.D., Naval Academy "Mircea cel Bătrân", Constanța

Abstract: *By defining the optimal path with minimum value in a finite directed graph, with positive value for arcs, we usually understand the determination of the path for which the amount of the arcs values that compose the path is minimum. This problem is solved by classics algorithms, but the most efficient from all is Dijkstra algorithm.*

In this paper we will define the concept of optimal path of minimum value as a result of the multiplication of the arcs values that compose that path in a finite directed graph, we will prove how these paths can be obtained, we will introduce an adaptation of the Dijkstra algorithm and finally, an implementation in C language.
Keywords: *directed graph, Dijkstra algorithm, multiplication*

1. INTRODUCTION

Let the finite directed graph $G=(X,\Gamma)$, $X=\{x_1,x_2\dots,x_n\}$ and U the set composed of the arcs of the graph. Each arc $u=(x_i,x_j)\in U$ has associated a number $l(u)=l(x_i,x_j)>1$ named the value of the arc u.

Definition 1 We assign to the value of the path $\mu=(x_{i_1},\dots,x_{i_k})$ the number $l(\mu)=\prod_{j=1}^{k-1} l(x_{i_j},x_{i_{j+1}})$.

Observation 1 The value of the path is equal to the multiplication of the values of the arcs that compose this path.

Definition 2 We name optimal path of minimum value between x_i and x_j vertices the path $\mu^*=(x_i,x_{p_1},x_{p_2},\dots,x_{p_k},x_j)$ that has the following property: for each other path $\mu=(x_i,x_{r_1},x_{r_2},\dots,x_{r_l},x_j)$ between x_i and x_j vertices we have the inequality $l(\mu^*)\le l(\mu)$.

Let the finite graph $G=(X,\Gamma,l)=(X,U,l)$ of n order, with valued arcs, $X=\{x_1,x_2,\dots,x_n\}$ and the matrix composed of the values of the arcs $A=(a_{ij})_{\substack{i=1,2,\dots,n\\ j=1,2,\dots,n}}$ where $a_{ij}=\begin{cases} l(x_i,x_j) & if\ \ (x_i,x_j)\in U\\ +\infty & if\ \ (x_i,x_j)\notin U\end{cases}$.

In order to define the path with the minimum multiplication of the values of the arcs that compose it, we may choose any algorithm that determines the path having as minimum value the sum between the arc values of the path applied to the logarithms matrix of the value of the arcs $A'=(a'_{ij})_{\substack{i=1,2,\dots,n\\ j=1,2,\dots,n}}$ where $a'_{ij}=\begin{cases} \ln(l(x_i,x_j)) & if\ \ (x_i,x_j)\in U\\ +\infty & if\ \ (x_i,x_j)\notin U\end{cases}$ and taking into account the following equalities $\ln(ab)=\ln a+\ln b$ and $a=e^{\ln a}$ with $a,b>0$.

2. THE USABILITY OF THE DIJKSTRA ALGORITHM

We will introduce Dijkstra algorithm in order to determine the optimal paths with minimum value for the sum between the values of the arcs that set those paths applied to A' matrix. The algorithm was discovered by the computer scientist Edsger Dijkstra in 1956 and published in 1959.

Let the finite directed graph $G=(X,U,l)$, $X=\{x_1,x_2,\dots,x_n\}$, with arcs valued with positive numbers. The Dijkstra algorithm determines the optimal path with minimum value between any x_i and x_j vertices of the graph.

The algorithm uses vertex $x_s\in X$ arbitrarily chosen, defined as the root of the graph (called the root of the graph). Vertex $x_s\in X$ it is also called the start vertex or the source vertex, and it can be any vertex $x_i\in X$ of the graph. We define s as the start vertex. The algorithm gets the minimum values of the paths between s vertex and x_i vertices with $x_i\ne s$. Let $d[x]$ the minimum value of the paths between s vertex and x vertex with $x\ne s$. Let S the set of selected vertices for which it is computed $d[x]$. In this case, $d[x]$ is the minimum value of the paths between s vertex and x vertex, paths with the property that all their vertices are in the S set except x. Let $\Gamma^{+}(x)$ the set of arcs that start from the x vertex and $\Gamma^{-}(x)$ the set of arcs that enter the x vertex.

In order to determine the optimal paths that have minimum value array $p=(p[i])_{\substack{i=1,2,\dots,n \\ i\neq s}}$ is used, also called the array of the predecessors. The array of the predecessors has n elements with $p[s]=0$. The $p[i]$ element of the p array is the predecessor vertex of the i vertex of the optimal path with minimum value.

Dijkstra algorithm consists of the following steps:

Step 1 (initialization)

$S=\{s\}$

$d[s]=0$

For each $i=1,2,\dots,n$ set $p[i]=0$

For any $x\in\Gamma^{+}(s)$, $p[x]=s$

For any $x\in X-S$

if $x\in\Gamma^{+}(s)$ then $d[x]=a'_{sx}$

else $d[x]=+\infty$

Step 2 (current iteration)

repeat

determine vertex $y\in X-S$ where $d[y]=\min\limits_{z\in X-S}\{d[z]\}$

if there are more z vertices for which it is achived the minimum $\min\limits_{z\in X-S}\{d[z]\}$ it will be arbitrary selected one of these vertices

if $d[y]<\infty$ then $S=S\cup\{y\}$

determine $\Gamma^{+}(y)$ and $\Gamma^{+}(y)-S$

if $\Gamma^{+}(y)-S\neq\phi$ then

for $z\in\Gamma^{+}(y)-S$ compute $d[z]=\min\{d[z],d[y]+a'_{yz}\}$

if $\min\{d[z],d[y]+a'_{yz}\}==d[y]+a'_{yz}$

then $p[z]=y$

else stop

until $S==X$ or $\Gamma^{+}(y)-S==\phi$ or $d[y]==\infty$

Step 3

Determining the optimal paths with minimum value between s vertex and all other vertices of the graph.

stop

In every iteration $d[z]$ values remain unchanged for $z\in S$. There are $n-1$ iterations at the second step.

3. THE DETERMINATION OF THE OPTIMAL PATHS

In order to determine the optimal paths from x_s to x_i for any $i=1,2,\dots,n$ with $i\neq s$ it is used the following idea : if x_k is the predecessor of the x_m vertex on an optimal path (where x_m is the successor of x_k), then we have the equality $p[m]=k$.

There are two methods for the determination of an optimal path. The first one, belonging to the predecessors, in which the predecessors of the current vertex are systematically determined, beginnig with x_i vertex and ending with x_s vertex. The second method the successors one, the successors of the current vertex are determined in a systematic manner starting with x_s vertex and finishing with x_i vertex.

As it follows, we will present both predecessors and successors methods that determine the optimal path with minimum value between S vertex and all the other vertices of the graph.

3.1. The predecessors method

Let $\mu = \left(x_s, x_{i_1}, x_{i_2}, \dots, x_{i_{p-2}}, x_{i_{p-1}}, x_{i_p}, x_i\right)$ an optimal path from x_s to x_i. The i_p index is determined from the $p\left[i\right] = i_p$ condition. If $i_p = S$ then the optimal path has been already determined otherwise the index i_{p-1} is extracted from the $p\left[i_p\right] = i_{p-1}$ condition. If $i_{p-1} = S$ then the optimal path has been already established otherwise the index i_{p-2} is obtained from the $p\left[i_{p-1}\right] = i_{p-2}$ condition. If $i_{p-2} = S$ then the optimal path has been determied. Finally, the index of S is determined from the following condition: $p\left[i_1\right] = S$.

3.2. The successors method

Let $\mu = \left(x_s, x_{i_1}, x_{i_2}, \dots, x_{i_{p-2}}, x_{i_{p-1}}, x_{i_p}, x_i\right)$ an optimal path from x_s to x_i. The index i_1 is obtained from the $p\left[i_1\right] = S$ property. If $i_1 = i$ then the optimal path has already been determined otherwise index i_2 is obtained from $p\left[i_2\right] = i_1$ condition. If $i_2 = i$ then the optimal path has been already achieved otherwise the i_3 index is determined from $p\left[i_3\right] = i_2$ condition. If $i_3 = i$ then the optimal path has already been obtained. Finally, it is determined i index from $p\left[i\right] = i_p$ condition.

4. IMPLEMENTATION IN THE C PROGRAMMING LANGUAGE.

```
//Dijkstra algorithm determines the paths and also  the minimum multiplying //value obtained from the values in paths  in a directed graph
//Let G=(X,U) directed graph, without loops
// Vertices are numbered 1,2,...,n and printed x1,x2,...,xn
//The programme receives at input a text file that has m+1 lines on the first line //we find n, the number of vertices and m, the number of arcs in the graph G  //separated by one space
// On the following m lines we find on each line the arc and its value, that belong //to the graph,  separated by one space
// The programme determines the paths with minimum value between a root //vertex arbitrarily chosen from the {1,2,...,n} set and other vertices in the graph
//The root vertex is received as input from the keyboard

#include "stdio.h"
#include "conio.h"
#include "malloc.h"
#include "limits.h"
#include "math.h"
#define INFINIT INT_MAX/2
#define dim 1000
//The prototypes of the functions defined in the programme
float ** alocmat (int);
int * alocvect_int(int);
float * alocvect_float(int);
int citire_n_m(int &, int &, char *);
int citire(char *,int **);
void afism(int **,int);
void afisv_int(int *,int,char *);
void afisv_float(float *,int,char *);
void afis_arce(int);
void afiss(int);
void drum(int *,int);
void transform(float *, int *, int);
void Dijkstra(float **,int *,int *,int *,int,int);
typedef struct arc
{
int x; int y; int val;
} muchie;
typedef struct multime
{
int nelem;
```

```
int S[dim];
}set;
muchie mu[dim];   // mu arc array
set start;
// Matrix allocation
// Function for the allocation of a square matrix with dimension n+1 and elements //of float type
// The function returns the address of the matrix or NULL

float ** alocmat (int n)
{
 int i;
float ** p=(float **) malloc ((n+1)*sizeof (float *));
 if ( p != NULL)
for (i=0; i<=n ;i++)
p[i] =(float *) malloc ((n+1)*sizeof (float));
return p;
}
//  Function for the allocation of a n+1 array with integer elements
// Initialization of components with 0
int * alocvect_int(int n)
{
int *p=(int *)calloc(n+1,sizeof(int));
return p;
}
// Function for the allocation of a n+1 array with real elements
// Initialization of components with 0
float * alocvect_float(int n)
{
float *p=(float *)calloc(n+1,sizeof(float));
return p;
}
// Function for reading an input file
// Read number n of vertices and the number m of arcs
int citire_n_m(int &n, int &m, char *nume)
{
FILE *f;
if((f=fopen(nume,"r"))!= NULL)
{
fscanf(f,"%d %d",&n,&m);
fclose(f);
return 1;
}
else
 return 0;
}
// Function that displays arcs from graph
void afis_arce(int m)
{
int i;
printf("\n Graful G are %d arce \n\n",m);
for(i=1;i<=m;i++)
printf(" Arc %d:\t ( x%d , x%d ) with value %d \n",i,mu[i].x,mu[i].y,mu[i].val);
}
// Function for reading the input file and building the matrix values
int citire(char *nume,float **a)
{
int i,j,x,y,n,m;
FILE *f;
if((f=fopen(nume,"r")) != NULL)
{
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(i==j)
a[i][j]=0;
else
a[i][j]=INFINIT;
for(i=1;i<=m;i++)
{
fscanf(f,"%d %d %d",&mu[i].x,&mu[i].y,&mu[i].val);
a[mu[i].x][mu[i].y]=log(mu[i].val);
```

```
}
fclose(f);
return 1;
}
else
return 0;
}
// Function to print the optimal path
void drum(int *p,int i)
{
if(p[i]) drum(p,p[i]);
printf(" x%d ",i);
}
// Function to print the n x n matrix
void afism(float **a,int n)
{
int i,j;
printf("\n\n Adjacency matrix \n\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf(" %5.2f ",a[i][j]);
printf("\n");
}
}
// Function to print the x array with n integer components
void afisv_int(int *x, int n, char *s)
{
int i;
printf(" %s : ( ",s);
for(i=1;i<n;i++)
printf(" %d , ",x[i]);
printf(" %d ) \n",x[i]);
}
// Function to print the x array with n float components
void afisv_float(float *x, int n, char *s)
{
int i;
printf(" %s : ( ",s);
for(i=1;i<n;i++)
printf(" %5.2f , ",x[i]);
printf(" %5.2f ) \n",x[i]);
}
// Function to display the selected vertices
void afiss(int n)
{
int i;
printf(" Set S with selected vertices has %d elements \n",start.nelem);
for(i=1;i<=n;i++)
if(start.S[i]!=0)
printf(" x%d ",i);
printf("\n");
}
// Function for transforming the optimal values log-exp
void transform(float *d, int *dprim, int n)
{
int i;
for(i=1;i<=n;i++)
dprim[i]=(int)(round(exp(d[i])));
}
// Function that implements Dijkstra algorithm
void Dijkstra(float **a,int *s,float *d,int *dprim, int *p,int n,int r)
{
int i,j,poz;
float min;
afisv_int(p,n," Predecessors array ");
getch();
for(i=1;i<=n;i++)
start.S[i]=0;
start.S[r]=1;
start.nelem=1;
```

```
afiss(n);
getch();
for(i=1;i<=n;i++)
{
d[i]=a[r][i];
if(i!=r)
if(d[i]<INFINIT)
p[i]=r;
}
afisv_float(d,n," Array of values ");
getch();
for(i=1;i<=n-1;i++)
{
min=(float)INFINIT;
for(j=1;j<=n;j++)
if(start.S[j]==0)
if(d[j]<min)
{
min=d[j];
poz=j;
}
afisv_int(p,n," Predecessors array ");
getch();
printf("Minimum value %5.2f \n",min);
start.S[poz]=1;
start.nelem++;
afiss(n);
getch();
for(j=1;j<=n;j++)
if(s[j]==0)
if(d[j]>d[poz]+a[poz][j])
{
d[j]=d[poz]+a[poz][j];
p[j]=poz;
}
afisv_float(d,n," Array of values ");
getch();
}
afisv_float(d,n," Array of optimal values ");
transform(d,dprim,n);
afisv_int(dprim,n," Array of optimal values of the values multiplication ");
getch();
for(i=1;i<=n;i++)
if(i!=r)
if(p[i])
{
printf("\n The minimum value of the multiplication path values between x%d vertex and x%d vertex is %d \n",r,i,dprim[i]);
printf(" The path with minimum multiplication between x%d vertex and  x%d vertex is  {",r,i);
drum(p,i);
printf("}\n");
}
else printf(" There are no paths between x%d vertex and x%d vertex \n",r,i);
}
// n is the number of vertices of the graph
// m is the current number of arcs of the graph
// a is the adjacency matrix
// d the array of minimal optimal values of the paths
// s the array of selected vertices
// p the predecessors array
// nume the name of the file associated with the graph
// r is the root vertex, arbitrarily from the set {1,2,...,n}
int main()
{
int *s,*p,n,m,i,j,r,*dprim;
float **a,*d;
char nume[30];
printf(" Dijkstra algorithm for directed graphs \n");
printf("\n File name");
gets(nume);
if(citire_n_m(n,m,nume))
{
```

```
printf("Number of vertices %d \n",n);
printf("Number of  arcs %d \n",m);
getch();
printf(" Start vertex ");
scanf("%d",&r);
if(1<=r && r<=n)
{
a=alocmat(n);
s=alocvect_int(n);
d=alocvect_float(n);
dprim=alocvect_int(n);
p=alocvect_int(n);
if(citire(nume,a))
{
afism(a,n);
getch();
afis_arce(m);
getch();
Dijkstra(a,s,d,dprim,p,n,r);
}
free(a);
free(s);
free(d);
free(dprim);
free(p);
}
else
printf(" Vertex %d does not exist \n",r);
}
else
printf(" Error reading file %s \n",nume);
getch();
}
```

5. REFERENCES

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Dijkstra's algorithm. Introduction to Algorithms (Second ed.). MIT Press and McGraw-Hill. pp. 595–601, 2001. ISBN 0-262-03293-7.

[2] Dijkstra, E. W. A note on two problems in connexion with graphs. Numerische Mathematik 1: 269–271, 1959. http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf.