Computer Science Journal of Moldova, vol.**5**, no.1(13), 1997

An algorithm and a program for finding the minimax path tree in weighted digraphs

R.Boliac

Abstract

An algorithm for finding the minimax path tree in a weighted digraph and a program in PASCAL, which implements this algorithm, are presented in this paper.

1 Introduction

The minimax path problem, which generalizes the well-known classical combinatorial problems of the shortest and the longest paths in a weighted directed graph is considered. This minimax problem arises as an auxiliary one when searching optimal stationary strategies in cyclic games [1, 2, 3] and solving some of network minimax transport problems [4]. The algorithm for finding all minimax paths with computational complexity $O(nm)$, where n is the vertex number and m is the edge number of G, and the program in PASCAL, implementing this algorithm, are presented further.

2 Problem formulation

Let $G = (V, E)$ be a directed graph with the vertex set V, $|V| = n$, and the edge set E, $|E| = m$. Let the non-negative cost function $c : E \to R^1_+$ be defined on the edge set. Asume that a vertex $v_0 \in V$ is

chosen so that for any vertex $v \in V$ there exists a path $P_G(v, v_0)$ from v to v_0. The vertex set V is divided into two disjoint subsets V_A and V_B ($V = V_A \cup V_B,\ V_A \cap V_B = \emptyset$).

Let s_A and s_B be two maps defined on V_A and V_B, respectively:

$$\begin{aligned} s_A &: v \to V(v) \qquad \text{for } v \in V_A, \\ s_B &: v \to V(v) \qquad \text{for } v \in V_B, \end{aligned}$$

where $V(v)$ is the set of extremities of edges $e = (v, u)$ originating in v, i.e. $V(v) = \{u \in V \mid e = (v, u) \in E\}$. Denote by $T_s = (V, E_s)$ the subgraph of G generated by the edges $e = (v, s_A(v))$ for $v \in V_A$ and by the edges $e = (v, s_B(v))$ for $v \in V_B$. Obviously, for an arbitrary $v \in V$ either a unique directed path $P_T(v, v_0)$ exists in T_s, or such a path does not exist in T_s. In the second case, if we pass through the edges from v, we get a unique directed cycle C_s.

For arbitrary (s_A, s_B) and $v \in V$ we define the quantity $\bar{c}(s_A, c_B, v)$ as the sum of edges in the path $P_T(v, v_0)$, if exists. If such a path does not exist in T_s , and the sum of edge costs in C_s is positive, then we assume $\bar{c}(s_A, s_B, v) = \infty$; otherwise, if the sum of edge costs in C_s is negative, $\bar{c}(s_A, s_B, v) = -\infty$. If the sum of edge costs in C_s is zero, then $\bar{c}(s_A, s_B, v)$ equals the sum of edge costs in the path which connects v and the cycle C_s.

The problem of finding the minimax path connecting the vertices w and v_0 is: to find

$$p(w) = \min_{s_A} \max_{s_B} \bar{c}(s_A, s_B, w)$$

and the maps $s_A^*,\ s_B^*$ for which

$$p(w) = \bar{c}(s_A^*, s_B^*, w) = \min_{s_A} \max_{s_B} \bar{c}(s_A, s_B, w).$$

We consider the problem of finding the minimax paths from all vertices of G to the vertex v_0. There exist such maps s_A^*, s_B^*, for which $T_{s^*} = (V, E_{s^*})$ is a tree with the root vertex v_0 [3]. So, the considered problem generalizes the well-known problem of the minimal path tree in a weighted graph.

The formulated problem can be interpreted as a dynamical game of two players with integral-time cost [3].

If $V = V_A$, then we have the shortest path problem in G (see [5]). If $V = V_B$, then we have the longest path problem in G [5].

3 The algorithm and the program

Further is presented an algorithm which generalizes Dijkstra's algorithm for finding the minimal path tree in a weighted graph [7]. The algorithm is based on the method of dynamical programming. We assign labels $l(v)$ to all vertices v of G. If a vertex $v \in V$ belongs to V_A, then its label is the upper bound of the length of the minimax path between v and v_0; otherwise the label of v gives the lower bound of the length of the minimax path between the vertices v and v_0. The labels are gradually changing (the labels of vertices from V_A are decreasing and the labels of vertices from V_B are increasing, respectively) by means of an iterative procedure, and at each step of this procedure exactly one label becomes constant. This means that this label gives the length of the minimax path between the considered vertex and the vertex v_0.

The algorithm

Let $l(v_i)$ denote the label of a vertex v_i.

Step 1. Set $l(v_0) = 0$ and consider this label constant. For all vertices v_i $(i \neq 0)$ set

$$l(v_i) = \begin{cases} +\infty, & \text{for } v_i \in V_A \\ 0, & \text{for } v_i \in V_B \end{cases}$$

and consider these labels temporary. Set $p = v_0,\ V_s = \{v_0\},\ E_s = \emptyset$.

Step 2. For all vertices $v_i \in V^-(p)$ ($V^-(p)$ is the set of origin vertices of edges which enter the vertex p) with temporary labels, change

the labels as follows:

$$l(v_i) = \begin{cases} \min\{l(v_i),\ c(v_i,p) + l(p)\}, & \text{for } v_i \in V_A \\ \max\{l(v_i),\ c(v_i,p) + l(p)\}, & \text{for } v_i \in V_B \end{cases} \tag{1}$$

Step 3. Find the vertex set

$$V^-(V_s) = \left(\bigcup_{v_i \in V_s} V^-(v_i) \right) \setminus V_s.$$

If $V^-(V_s) = \emptyset$, then go to step 7. Otherwise, find a vertex $v_i^* \in V^-(V_s)$ with the minimum label.

Step 4. If $v_i^* \in V_A$, then go to step 5. If $v_i^* \in V_B$ and $V_G(v_i^*) \subseteq V_s$, then go to step 5. Oterwise, delete from G all edges connecting the vertex v_i^* with the vertices of V_s and go to step 3.

Step 5. Consider the label of v_i^* constant and set $p = v_i^*$, $V_s = V_s \cup \{v_i^*\}$. Moreover, add to the set E_s the edge which originates in v_i^* and for which the minimum or the maximum was reached in (1), for $v_i^* \in V_A$ or $v_i^* \in V_B$, respectively.

Step 6. If $V_s = V$, then the directed tree $T_s = (V, E_s)$ is constructed. The vertex labels give us the lengths of the minimax paths, connecting these vertices with the vertex v_0. Stop. If $V_s \neq V$, then go to step 2.

Step 7. There exists a directed cycle in G, so that if we pass from any temporarily labeled vertex through the oriented edges of G, then we find ourselves on this cycle. In this case the minimax paths to v_0 exist only for the vertices with constant labels. For the vertices with temporary labels these paths have the lengths equal to $+\infty$. Set all the temporary labels equal to $+\infty$ and consider them constant. Stop.

Note that if $V = V_A$, then this algorithm becomes Dijkstra's algorithm for finding the shortest paths tree in the directed graph G [7].

It should be observed that the above algorithm has the computational complexity $O(nm)$. This can be easily deduced from the steps 2–6 of the algorithm.

The above algorithm was programmed in PASCAL. The obtained program works for graphs with up to 100 vertices. It has many possibilities, such as: the possibility of loading the graph G from a file, the possibility of drawing the graph G directly in the program, the possibility of editing the loaded or drawn graph, the possibility of saving the graph G to a file. The interface with the user is implemented in a pleasant manner. The user can communicate with the program using the menu and program's communications. As a result of this program's work the minimax path tree T_{s_*} is obtained. This tree is represented on the computer's screen and is also saved to a file.

Further we present the procedure `Min_Max`, which implements the algorithm. The vertices of G are numbered so that the vertex v_0 receives the number n. In this procedure the following types and variables are used:

`type vect=array[1..n] of real`;

`type vector=array[1..n] of integer`;

`type set1=set of 1..n`;

`var C:array[1..n,1..n] of real` – the edge cost matrix, i.e. if the edge $(i, j) \in E$, then `C[i,j]` is the cost of this edge; otherwise `C[i,j]=0`;

`var l:vect` – the vector of vertex labels, i.e. `l[i]` is the label of vertex i;

`var labeled:array[1..n] of boolean` – the vector containing the information about the vertices with temporary labels, i.e. if `labeled[i]=TRUE`, then the vertex i is temporarily labeled, otherwise this vertex is not labeled;

`var lab_c:array[1..n] of boolean` – the vector which contains the information about the vertices with constant labels, i.e. if `lab_c[i]=TRUE`, then the label of vertex i is constant and its label `l[i]` is the length of the minimax path connecting the vertex i with the vertex n;

`var next:vector` – the vector containing the information about the minimax paths in G, i.e. `next[i]` gives the vertex which follows the vertex i in the minimax path connecting vertices i and n.

The procedure `Min_Max`

```
procedure Min_Max( var next:vector; var l:vect);
   label S2,S3,S4,S5,S6,S7;
   var i,j,p,v_i,mn:integer;
       labeled,lab_c:array[1..n] of boolean;
       VV_s, V_s,nb:set1;
   begin
         for i:=1 to n-1 do
           begin labeled[i]:=FALSE;
                 lab_c:=FALSE;
           end;
         l[n]:=0;
         labeled[n]:=TRUE;
         lab_c[n]:=TRUE;
         p:=n;
         V_s:=[p];
    S2:  for i:=1 to n do
           if (C[i,p]<>0) and (not(lab_c[i])) then
             if not(labeled[i])
               then begin l[i]:= c[i,p]+l[p];
                          labeled[i]:=TRUE;
                    end
               else if (i in V_A)
                      then l[i]:=min(l[i],C[i,p]+l[p])
                      else l[i]:=max(l[i],C[i,p]+l[p]);
    S3:  VV_s:=[];
         for i:=1 to n do
           if (i in V_s) then
             begin nb:=[];
                   for j:=1 to n do
```

```
                  if C[j,i]<>0 then nb:=nb+[j];
                VV_s:=VV_s+nb;
          end;
      VV_s:=VV_s-V_s;
      if VV_s=[] then goto S7;
      v_i:=0;
      repeat inc(v_i);
      until (v_i in V_s);
      mn:=l[v_i];
      for i:=v_i+1 to n do
        if (i in VV_s) and (l[i]<mn) then
          begin v_i:=i;
                mn:=l[i];
          end;
S4:   if (v_i in [1..n]-V_A) then
        begin nb:=[];
              for i:=1 to n do
                if C[v_i,i]<>0 then nb:=nb+[i];
              if not(nb<=V_s) then
                begin for i:=1 to n do
                          if (i in V_s) then
                            C[v_i,i]:=0;
                      goto S3;
                end;
        end;
S5:   lab_c[v_i]:=TRUE;
      p:=v_i;
      V_s:=V_s+[p];
S6:   if V_s<>[1..n] then goto S2;
S7:   for i:=1 to n-1 do
        begin if lab_c
                then begin j:=0;
                           repeat inc(j);
                           until (i<>j) and
                                 (C[i,j]<>0) and
```

```
                                        (l[i]=l[j]+C[i,j]);
                                 next[i]:=j;
                            end
                       else begin j:=0;
                                 repeat inc(j);
                                 until not(lab_c[j]);
                                 next[i]:=j;
                            end;
               end;
     end;
```

There were provided experiments for different graphs and the minimax path trees in these graphs were obtained.

References

[1] Moulin H., Prolongement des jeux à deux jouers de somme nulle, *Bull. Soc. math.*, 1976, Mem. 45.

[2] Gurvitch V. A., Karzanov A. V., Khatchiyan L. G., Cyclic games: Finding minimax mean cycles in digraphs, *J. Comp. Mathem. and Math. Phys.*, 28 (1988), 1407–1417 (in Russian).

[3] Lozovanu D. D., *Extremal-combinatorial problems and the algorithms for their solving.* Ştiinţa, Chişinău, 1991 (in Russian).

[4] Lozovanu D. D., A strongly polynomial time algorithm for finding minimax paths in network and solving cyclic games, *Cybernetics and System Analysis*, 5 (1993), 145–151 (in Russian).

[5] Christofides N., *Graph Theory: An Algorithmic Approach.* Academic Press, London, 1975.

[6] Lozovanu D. D., Trubin V. A., Minimax path problem in network and the algorithm for its solving. *Discrete Mathematics* 6 (1994), 138–144 (in Russian).

[7] Dijkstra E. W., A Note on Two Problems in Connection with Graphs, *Num. Mathem.*, 1 (1959), 269–271.

Rodica Boliac,
Institute of Mathematics,
Academy of Sciences of Moldova,
5 Academiei str., Kishinev
2028, Moldova.

Received September 2, 1996