Computer Science Journal of Moldova, vol.**7**, no.3(21), 1999

Algorithm and program for finding the optimal paths in dynamic networks

A. Leconiuc

Abstract

The program of finding the optimal paths in dynamic networks is studied. Some aspects concerning the implementation of an earlier proposed algorithm, as well as the description of the program that actualy reprezents the algorithm implementation are prezented.

1 Introduction

The problem of finding the optimal paths in dynamic networks, which was studied in [1-4], is considered in this paper. Some aspects concerning the implementation of the algorithm, proposed in [4], as well as the description of the program that actualy reprezents the algorithm implementation, are described here in a more detailed way .

2 The algorithm and problem formulation

The algorithm and problem formulation for finding the optimal paths in dynamic networks is given here. The problem is studied in the case when the cost functions on the edges of the network are positive and non-decreasing functions. We use the algorithm from [4] and propose some modifications of the algorithm for the program implementation.

2.1 Problem formulation

Let L be a dynamic system with the finite set of states V, $|V| = n$, and at every time moment $t = 0, 1, 2, \ldots$ the state of system L is $v(t) \in V$. Two states v_0 and w are chosen in V, where v_0 is the initial state ($v_0 = v(0)$) and w is the final state of the system, i.e. w is the state in which the system must be brought. The dynamics of system is described by the directed graph of passages $G = (V, E)$, a edge $e = (u, v)$ of which signifies the possibility of passage of system L from the state $u = v(t)$ to the state $v = v(t + 1)$ at any time moment $t = 0, 1, 2, \ldots$. To each edge $e = (u, v)$ a function $c_e(t)$ is assigned, which reflects the cost of system's passage from the state $v(t) = u \in V$ to the state $v(t + 1) = v \in V$ at any time moment t.

So, we assume that the set of states V is divided into two disjoint subsets V_A and V_B ($V = V_A \cup V_B$, $V_A \cap V_B = \emptyset$), where V_A is the position set of player A, and V_B is the position set of player B. The first player is choosing the passages in order to reach the vertex w, and the second player chooses the passages so that system L could not reach the state w. The first player has the aim to reach the final state with the minimal game cost, while the second player has the aim to maximise the cost of the game.

Define strategies of players as the maps

$$s_A : u \to V^+(u), \quad \text{for } u \in V_A \setminus \{w\},$$

$$s_B : u \to V^+(u), \quad \text{for } u \in V_B \setminus \{w\}.$$

Denote by $G_s = (V, E_s)$ the digraph generated by the set of edges $(v, s_A(v))$ and $(v, s_B(v))$. Obviously that for fixed s_A and s_B there exists a oriented path $P_s(v_0, w)$ connecting v_0 and w, or such a path does not exist. If exists, this path is unique and passing through its edges from v_0 to w, numbers $0, 1, \ldots, k_s$ can be assigned to these edges. These numbers characterize the moments of time $t_e(s_A, s_B)$ when the system passes from a state to another if players A and B fix their corresponding strategies s_A and s_B. If there are no paths from v_0 to w in G_s, then starting in v_0 and passing through the directed edges, we get a unique directed cycle C_s.

For a fixed pair of strategies s_A, s_B and fixed v_0 define the quantity $f_{v_0}(s_A, s_B)$ equal to $\sum_{e \in P_s(v_0,w)} c_e(t_e(s_A, s_B))$, if there exists a path $P_s(v_0, w)$ from v_0 to w in G_s; otherwise set $f_{v_0}(s_A, s_B) = +\infty$. Consider the problem of finding strategies s_A^*, s_B^*, so that

$$f_{v_0}(s_A^*, s_B^*) = \min_{s_A} \max_{s_B} f_{v_0}(s_A, s_B).$$

2.2 Algorithm

Step 1. To every vertex $v \in V$ assign two labels: $l(v)$ and $t(v)$. $l(v)$ is the length of the optimal path from $v \in V$ to w, $t(v)$ is the time moment at which this path passes through vertex v.

$$l(w) = 0, \quad t(w) = t^*, \text{ and consider these labels constant;}$$

$$l(v) = \begin{cases} +\infty, & v \in V_A, \\ 0, & v \in V_B, \end{cases} \qquad t(v) = +\infty \ \ (v \neq w),$$

and consider these labels temporary.

Consider $p = w$, $V_s = \{w\}$, $E_s = \emptyset$.

Step 2. For vertices $v \in V^-(p)$ ($V^-(p) = \{v \in V | (v, p) \in E\}$) with temporary labels modify labels by formulas:

- if $v \in V_A$ and $l(v) > c_{(v,p)}(t(p) - 1) + l(p)$ then $l(v) = c_{(v,p)}(t(p) - 1) + l(p)$, $t(v) = t(p) - 1$;
- if $v \in V_B$ and $l(v) < c_{(v,p)}(t(p) - 1) + l(p)$ then $l(v) = c_{(v,p)}(t(p) - 1) + l(p)$, $t(v) = t(p) - 1$.
- If $t(v) = 0$, then delete from G' the edges (u, v), $u \in V$.

Step 3. Find the vertex set $V^-(V_s) = \Big(\bigcup_{v \in V_s} V^-(v) \Big) \setminus V_s$. In $V^-(V_s)$ find the vertex v^* with the minimal temporary label $l(v^*)$.

Step 4. If $v^* \in V_A$, then go to **Step 5**. If $v^* \in V_B$ and $V(v^*) \subseteq V_s$, then go to **Step 5**. If $v^* \in V_B$ and $V(v^*) \not\subseteq V_s$, then delete from G all edges connecting v^* with V_s and go to **Step 3**.

Step 5. Consider the labels of v^* constant and put $p = v^*$, $V_s = V_s \cup \{v^*\}$. Add to set E_s the edge $(v^*, u) \in E$ for which $l(v^*) = l(u) + c_{(v^*,u)}(t(u) - 1)$ and $t(v^*) = t(u) - 1$.

Step 6. If $p = v_0$ then STOP. The optimal path from v_0 yo w can be obtained by passing from v_0 to w through the directed edges in E_s. If $p \neq v_0$, go to **Step 2**.

3 Program description and examples

3.1 Program description

This program was designed in Delphi 3.0 programming environment. It uses dynamic arrays for storing algorithm data. The memory is allocated and freed dynamically at runtime as needed. That's why the program is able to process any dynamic network, as far as data describing the graph do not exceed your operating system memory. This program has also another characteristic, built into the Microsoft operating systems, called – multithreading. What does this mean? When a process(program) starts, a primary thread is created and launched. However, it is possible to have more than one thread per process, and each of them performing different tasks. The central process unit can execute only one thread at a time. Due to its great speed, it executes a thread for a short time, then switches to another thread and so on, thus creating the impression that all of the threads are running in parallel. The benefit of the end user from multithreading is that you can build a new graph (or anything else) while another graph is currently processing, or you can process two or more graphs at the same time. As mentioned in the algorithm, the graph edges represent functions. This program may identify the following functions or any combination of them : $sin(t)$, $cos(t)$, e^t – exponential +, -, /, * ^ – power. All these functions are of the same parameter t. The capacity of determining the value of a user defined function at runtime is implemented quite straightforward. After the user has specified all the functions for a dynamic network, the program creates a file with as much procedures as the number of the dynamic network edges. Further, the file is compiled

as a dll. When a function value is needed, the corresponding dll procedure is called. Another characteristic of this program is that any graph can be saved to, or loaded from a file. This is especially useful when you are dealing with large graphs. Once saved into a file, you can load it in no time and use it, instead of introducing it every time. The program has a graphical, easy to use interface, which enables you to draw graphs (put vertexes where you like them to be) and then modify graph matrix as needed. All the commands the program can execute have intuitive names, so anyone who knows the problem formulation would be able to use them. Note that the program performance depends greatly on your processor's frequency as well as on the available physical memory. The program works under Windows 95/98/NT operating systems.

3.2 Examples

First example

$$\begin{pmatrix} 0 & 10 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 & 4 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

First player vertexes : 1, 4, 5
Second player vertexces : 2, 3, 6
Initial vertex : 1
Final vertex : 6
Optimal path : 1, 3, 4, 6
Path lenght : 14

Second example

$$\begin{pmatrix} 0 & 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 3 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

First player vertexes : 1, 3, 4, 7
Second player vertexces : 2, 5, 6
Initial vertex : 1
Final vertex : 7
Optimal path : 1, 4, 3, 5, 6, 7
Path lenght : 11

References

[1] D.D. Lozovanu. *Extremal-combinatorial problems and algorithms for their solving.* Stiinta, Kishinev, 1991. (in Russian)

[2] D.D. Lozovanu, *Algorithms for solving some network minimax problem and their applications.* Cybernetics, No.1, 1991, pp.70-75. (in Russian)

[3] R. Boliac, *An algorithm and a program for finding the minimax path tree in weighted digraphs.* Comput. Sci. Journal of Moldova, Vol.5, No.1, 1997. pp.55-63

[4] R. Boliac, D. Lozovanu, A. Leconiuc, *Optimal paths in Dinamic Networks.* Bul. Acad. of Sci. of R.M. 1999, No.1.

A.Leconiuc,
Institute of Mathematics
and Computer Science,
Academy of Sciences of Moldova
5, Academiei str., Kishinev,
MD2028, Moldova
phone: 73–83–35
e-mail: *lozovanu@math.md*

Received October 30, 1999