

DRUMURI OPTIME ÎN GRAFURI ORAR

ION COZAC

Abstract

We studied the optimum route problem in time-table graphs, in order to develop a web server supporting on-line queries. This paper describes how to solve the following problems:

- *separate train time-tables are given; how to build the entire time-table graph?*
- *how to use the maximum allowed distance as restriction criterion in the search process?*
- *how to split the time-table graph into two distinct graphs, to improve the efficiency?*

Our study is based on the model designed by Schulz, Wagner and Zaroliagis.

Keywords: *time-table graph, near optimum path, earliest arrival, maximum allowed distance*

1. Datele inițiale ale problemei

O rețea feroviară este modelată cu ajutorul unui graf simetric ponderat

$$G = (X, U, d):$$

- X este o mulțime finită și nevidă de noduri, asociată cu mulțimea stațiilor rețelei;
- U este o mulțime de arce care indică legături directe între stații: dacă $(x,y) \in U$ atunci între x și y nu există alte stații;
- d este o funcție distanță simetrică, $d: U \rightarrow \mathbb{N}^*$; $d(x,y) = d(y,x)$ pentru orice $(x,y) \in U$.

Prima problemă care trebuie rezolvată este cea a introducerii cât mai comode a datelor despre un orar nou. Pentru fiecare tren se precizează: care este traseul acestuia, prin ce stații trece, ora sosirii și plecării pentru fiecare stație.

Pentru un tren oarecare se introduc stația inițială de plecare și stația finală de sosire. În acest moment s-ar putea determina un drum de distanță minimă între cele două stații. Dar nu orice tren parcurge traseul de distanță minimă dintre cele două stații. Astfel că, dacă traseul trenului diferă de cel având distanța minimă, trebuie să precizăm un nod intermediar, sau poate chiar două. În continuare se determină un traseu de distanță minimă care să treacă prin toate nodurile în ordinea dată; în acest scop poate fi folosit algoritmul Dijkstra [1].

Având toate stațiile de pe parcursul unui tren, se pot preciza pentru fiecare stație ora sosirii și ora plecării. Stația inițială de plecare nu are asociată o oră de sosire, și stația finală de sosire nu are asociată o oră de plecare. De asemenea, este posibil ca o stație intermediară să nu aibă asociate aceste informații, dacă trenul nu are oprire în stația respectivă.

2. Modelare cu ajutorul unui graf expandat

Există modelări ale problemei mersului trenurilor disponibile pe Internet. Prezentăm în continuare un astfel de model (cf Schulz et al [4]), simplificat și completat pentru a modela cât mai bine unele particularități ale rețelei feroviare din România. Să presupunem că avem trei stații: Sa , Sb și Sc , și trei trenuri cu următorul program de circulație:

- T1: (Sa , 8:00) – (Sb , 8:15, 8:20) – (Sc , 8:30);
- T2: (Sb , 12:00) – (Sc , 12:45, 12:50) – (Sa , 13:10);
- T3: (Sc , 14:00) – (Sa , 14:15, 14:20) – (Sb , 14:35).

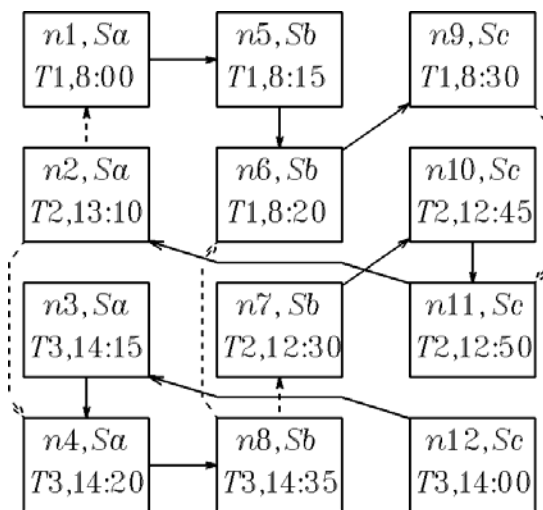
Pentru fiecare stație, și pentru fiecare moment de sosire și plecare, se definește câte un nod în graful expandat. Astfel vom avea următoarele noduri:

- cele asociate stației Sa : $n_1(T1,8:00)$, $n_2(T2,13:10)$, $n_3(T3,14:15)$, $n_4(T3,14:20)$;
- cele asociate stației Sb : $n_5(T1,8:15)$, $n_6(T1,8:20)$, $n_7(T2,12:00)$, $n_8(T3,14:35)$;
- cele asociate stației Sc : $n_9(T1,8:30)$, $n_{10}(T2,12:45)$, $n_{11}(T2,12:50)$, $n_{12}(T3,14:00)$.

Se definesc arce în graful expandat astfel:

- arce care indică momente succesive din ruta unui tren: (n_1, n_5) , (n_5, n_6) , (n_6, n_9) , (n_7, n_{10}) , (n_{10}, n_{11}) , (n_{11}, n_2) , (n_{12}, n_3) , (n_3, n_4) , (n_4, n_8) ;
- arce care indică legături cu alte trenuri: (n_8, n_7) , (n_8, n_6) , (n_5, n_7) , (n_2, n_1) , (n_2, n_4) , (n_3, n_1) , (n_9, n_{11}) , (n_9, n_{12}) , (n_{10}, n_{12}) .

În a doua listă se includ arcele care se referă la aceeași stație și unesc un vârf de sosire corespunzând unui tren cu un vârf de plecare corespunzând altui tren.



Un exemplu de graf orar

Uneori este necesar să introducem legături suplimentare în graful expandat, pentru a indica faptul că un tren de legătură trebuie luat dintr-o stație vecină. Exemple: (*Ploiesti*

Vest, Ploiesti Sud), (*Bucuresti Nord, Bucuresti Basarab*). În aceste situații legăturile trebuie să țină seama de timpul necesar pentru a ajunge dintr-o stație în cealaltă cu un alt mijloc de transport (de exemplu autobuz). Aceste arce suplimentare vor fi definite astfel:

- nodul inițial al arcului este asociat unui moment de sosire, și nodul final unui moment de plecare;

- se vor avea în vedere acele noduri care corespund unui interval de timp mai mare decât cel necesar pentru a parcurge distanța dintre cele două stații.

Graful expandat $G = (X, U, time, dist)$ asociat mersului trenurilor este un graf orientat unde:

- X este mulțimea de noduri definite cum s-a arătat mai sus; un nod $x' \in X$ este un triplet (s, t, b) cu semnificația: s – stație, t – tren, b – ora plecării sau sosirii;

- U este mulțimea de arce definite cum s-a arătat mai sus;

- $time : E \rightarrow \mathbf{N}$; $time(x', y') = (t(y') - t(x') + 1440) \bmod 1440$

- $dist : E \rightarrow \mathbf{N}$; $dist(x', y') = \text{distanța minimă de la } s(x') \text{ la } s(y')$

$t(x')$ este componenta t a nodului x' , $s(x')$ este componenta s a nodului x' .

Distanța minimă este determinată în graful G al rețelei feroviare. Valorile funcțiilor $time$ și $dist$ se calculează o singură dată, în faza de preprocesare.

Fiind date două stații pl și so , trebuie să se determine mai multe trasee optime (ca și timp de călătorie) de la pl la so . Trebuie să precizăm mai exact ce înseamnă traseu optim. Considerând stația pl și momentul de plecare b , să se determine un traseu care ne permite să ajungem în cel mai scurt timp în stația so . Dacă vom considera diferite ore de plecare, este posibil să avem trasee diferite cu durate diferite de călătorie, toate fiind optime conform precizării de mai sus. În literatura de specialitate se folosește termenul *the earliest arrival problem* (Pyrga et al [4]).

Criteriul de optim este relativ la momentul plecării. Dar dacă se identifică două trasee diferite, cu momente diferite de plecare, și momentul sosirii coincide, va fi luat în considerare doar acel traseu care are timpul total de călătorie minim.

Pentru a determina traseul optim se utilizează algoritmul Dijkstra cu câteva modificări.

Se determină o mulțime $S' \subset X'$ asociată stației p' care se referă doar la momente de plecare. Se determină de asemenea o mulțime $T' \subset X'$ asociată stației s_0 care se referă doar la momente de sosire. Pentru fiecare nod $s' \in S'$ se determină un traseu optim de la s' la primul nod t' care este depistat în T' . Pentru două noduri consecutive $m_1(s_1, t_1, h_1)$ și $m_2(s_2, t_2, h_2)$ timpul se determină astfel:

$$(1440 + h_2 - h_1) \bmod 1440$$

dacă momentele de timp sînt date în minute în intervalul $[0, 1439]$.

Autorii citați mai sus propun ca în a doua listă de arce să se includă acele arce care se referă la aceeași stație, și care unesc două momente consecutive de timp, indiferent de tipul acestora (plecare sau sosire), la care se adaugă un arc care unește ultimul moment de timp cu primul, pentru a da posibilitatea de a obține legături la momentul trecerii de la o zi la alta.

Avantajul acestei abordări este obținerea unui număr redus de arce în graful orar. Dezavantajul apare la gestionarea dificilă a situațiilor în care e nevoie să se schimbe un tren, dacă se urmărește în același timp și un număr minim de schimbări.

Deoarece modelul nostru cere să se construiască legături între orice vîrf de sosire și orice vîrf de plecare, numărul total numărul total de arce este de cîteva ori mai mare. În schimb se gestionează mult mai ușor situațiile în care e nevoie să se schimbe un tren.

Verificări suplimentare permit reducea numărului de arce, selectînd doar stațiile unde astfel de schimbări sînt într-adevăr necesare. De exemplu, arcul (n_5, n_7) nu e necesar deoarece se poate ajunge la stația S_c cu trenul T1 – direct, pe cînd folosirea acestui arc ar presupune schimbarea trenului. Un alt exemplu: avem un tren pe ruta $S_a \rightarrow S_b \rightarrow S_c$, dacă am avea un tren pe ruta $S_c \rightarrow S_b \rightarrow S_a$, nu s-ar justifica o schimbare între cele două trenuri în stația S_b .

Este necesară o selecție foarte atentă a arcelor care vor defini graful orar. De rezultatul acestei selecții va depinde într-o măsură foarte mare atît calitatea informațiilor furnizate de programul de căutare, cît și eficiența acestuia. Așadar, etapa de preprocesare a datelor este foarte importantă în astfel de situații.

3. Algoritm de căutare accelerată

Deoarece graful orar are un număr foarte mare de vîrfuri și arce, unii autori recomandă (Schulz et al [3]) construirea unor grafuri pe mai multe nivele, căutarea făcîndu-se mai întîi în graful cel mai mic, după aceea în graful de pe nivelul următor și așa mai departe. În graful de pe un nivel mai mare se face căutare restrictivă, doar pentru vîrfurile aflate în vecinătatea celor identificate la nivelul anterior.

Propunem în continuare un criteriu mai simplu de restricționare a căutărilor, care nu necesită memorie suplimentară. Inițial se determină distanța minimă de la stația finală la fiecare din celelalte stații, și se rețin doar stațiile care se află la o distanță care este cu cel mult 50% (de exemplu) față de distanța minimă. Această selecție se face foarte repede, deoarece graful rețelei feroviare este de zeci de ori mai mic față de graful orar.

În continuare, pentru fiecare vîrf extras (algoritmul Dijkstra modificat) căutarea continuă numai dacă distanța deja parcursă plus distanța minimă (rămasă de parcurs, determinată anterior) nu depășește limita stabilită.

Fie $p/$ stația de plecare și so stația de sosire. Trebuie să determinăm mai multe rute optime posibile de la $p/$ la so , criteriul de optim fiind timpul total de călătorie. Pentru a rezolva această problemă folosim o variantă a algoritmului Dijkstra. Selectăm o submulțime $S \subset X$ asociată stației $p/$ și orice nod este de plecare. Selectăm de asemenea o submulțime $A \subset V$ asociată stației so și orice nod este de sosire. Pentru fiecare nod $s \in S$ determinăm o rută optimă de la s la primul nod x' întîlnit în A .

Deoarece graful orar este destul de mare sînt necesare tehnici de accelerare a căutării. Propunem în continuare un criteriu care restrînge căutările fără a fi nevoie de memorie suplimentară. Mai întîi determinăm distanța minimă de la $p/$ la so , pe care o înmulțim cu un factor $f > 1$; obținem astfel md , distanța maximă admisă. De exemplu, dacă acceptăm o rută care acoperă o distanță cu 50% mai mare decît distanța minimă, atunci $f = 1,50$.

Algoritm MarkStations (variantă Dijkstra);

Input. $G = (X, U, d)$ graful rețelei feroviare;

pl , stația de plecare; so , stația de sosire;

f , factor de deviere maximă ($f > 1$).

Output. D , lista distanțelor de la so la toate celelalte stații;

md , distanța maximă admisă.

begin

for (each $x \in X$) do $D[x] := \infty$;

$D[so] := 0$; $md := 0$; $Z := \{so\}$;

while ($Z \neq \emptyset$) do begin

$x := \text{ExtractMin}(Z)$; // x are $D[x]$ distanța minimă dintre toate nodurile din Z

if ($x = pl$) then $md := D[x] * f$;

if ($md > 0$) and ($D[x] > md$) then break;

for (each $y \in \text{Succ}(x)$) do begin // pentru fiecare y , succesori al lui x

if ($y \notin Z$) then $Z := Z \cup \{y\}$;

$w := D[x] + d(x,y)$;

if ($D[y] > w$) then $D[y] := w$;

end;

end;

end (algorithm).

Acum sîntem pregătiți să determinăm o rută de la un nod oarecare din S la primul nod întîlnit din A .

Algoritm NearOptimumRoute (variantă Dijkstra);

Input. $G = (X, U, time, dist)$ graful orar;

s , nod de plecare; A , set de sosire;

D , lista distanțelor, determinată anterior;

md , distanța maximă admisă, determinată anterior.

Output. R , ruta găsită (lista nodurilor).

begin

```

for (each  $x' \in X^n$ ) do begin
     $P[x'] := \infty$ ;  $F[x'] := \infty$ ;  $T[x'] := \infty$ ;
end;
 $F[s'] := 0$ ;  $T[s'] := t(s')$ ;  $Z := \{s'\}$ ;  $P[s'] := s'$ ;
while ( $Z \neq \emptyset$ ) do begin
     $x' := \text{ExtractMin}(Z)$ ; //  $x'$  are costul minim dintre toate nodurile din  $Z$ 
    if ( $x' \in A^n$ ) then break;
    for (each  $y' \in \text{Succ}(x')$ ) do begin // pentru fiecare  $y'$ , succesori al lui  $x'$ 
        if ( $y' \notin Z$ ) then  $Z := Z \cup \{y'\}$ ;
         $wd := F[x'] + \text{dist}(x', y')$ ;
        if ( $D[s'(y')] + wd > md$ ) then continue; // ignorăm rute foarte lungi prin  $y'$ 
         $wt := T[x'] + \text{time}(x', y')$ ;
        if ( $(T(x', y'), F(x', y')) > (wt, wd)$ ) then begin // comparare lexicografică
             $T[y'] := wt$ ;  $F[y'] := wd$ ;  $P[y'] := x'$ ;
        end;
    end;
end;
 $k := 0$ ;  $R[0] := x'$ ; //  $x'$  este nodul final
while ( $x' \neq d^n$ ) do begin
     $x' := P[x']$ ;  $k := k + 1$ ;  $R[k] := x'$ ;
end; // Lista R va fi parcursă ulterior în ordine inversă
end (algorithm).

```

Algoritmul este numit *NearOptimumRoute* deoarece e posibil ca unele rute să nu fie neapărat optime (din punctul de vedere al timpului de călătorie), deoarece unele rute prea lungi (care dau o distanță mai mare decât md) sînt abandonate, chiar dacă ar putea da timpi de călătorie mai buni. Pare curios, dar România este țara tuturor posibilităților: există rute pe distanțe scurte unde circulă puține trenuri, și alte rute pe distanțe mai lungi dar mai circulă.

P se folosește pentru memora informații care vor fi folosite pentru a reconstitui o rută de la d' la nodul final. O rută de la d' la y' folosește x' ca nod precedent, deci $P[y'] = x'$.

T se folosește pentru a memora, pentru fiecare nod y' , timpul scurs de la $s(d')$ la $s(y')$.

F se folosește pentru a memora, pentru fiecare nod y' , distanța parcursă de la $s(d')$ la $s(y')$. Această listă se folosește pentru a verifica dacă distanța estimată de la p/l la s_0 depășește sau nu distanța maximă admisă:

$D[s(y')]$ este distanța minimă de la $s(y')$ la s_0 ;

wd este distanța parcursă de la p/l la $s(y')$; $wd \geq$ distanța minimă de la p/l la $s(y')$.

Criteriul de optim este dat de cuplul (T, F) astfel: în primul rînd contează timpul de călătorie minim, iar în caz de egalitate distanța minimă.

4. Implementare

Algoritmii descriși mai sus au fost incluși într-o aplicație web care poate fi găsită la adresa <http://193.226.19.29:1026>; pentru implementarea algoritmilor de drum optim am folosit cozi de prioritate (Johnson [2]).

Aplicația este scrisă în limbajul C și compilată cu GNU C version 4.1.0 . Sistemul de calcul are următoarele caracteristici: procesor Intel Celeron la 768 MHz, 128 Mb RAM memorie, și rulează un sistem de operare Linux Fedora.

Graful rețelei CFR are 1964 stații și 4020 arce. Graful orar are 47840 noduri și 224810 arce (1-Jan-2007, www.infofer.ro).

Pentru a evalua performanța serverului web am cerut să se determine rutele indicate mai jos. Tabelul de mai jos indică: stația de plecare, stația de sosire, distanța minimă, numărul nodurilor de plecare (numărul de trenuri care pleacă din stația de plecare), timpul total de răspuns, timpul mediu de răspuns.

În general, timpul total de răspuns depinde de numărul nodurilor de plecare, de distanța parcursă de la plecare la sosire, și de cât de repede poate serverul să găsească o direcție corectă spre stația de sosire. Tocmai de aceea am introdus ca și criteriu de restricție distanța maximă admisă: rutele prea lungi sînt abandonate.

Evaluarea performanțelor serverului web

Rute	Timp de răspuns
<i>Bucuresti Nord → Tirgu Mures</i> 448 km, 105 noduri de plecare	0,479 secunde (timp total) 0,004562 secunde (medie)
<i>Tirgu Mures → Bucuresti Nord</i> 448 km, 24 noduri de plecare	0,120 secunde (timp total) 0,005 secunde (medie)
<i>Bucuresti Nord → Timisoara Nord</i> 531 km, 105 noduri de plecare	0,520 secunde (timp total) 0,004952 secunde (medie)
<i>Timisoara Nord → Bucuresti Nord</i> 531 km, 76 noduri de plecare	0,328 secunde (timp total) 0,004316 secunde (medie)
<i>Tirgu Mures → Dej Calatori</i> 150 km, 24 noduri de plecare	0,098 secunde (timp total) 0,004083 secunde (medie)
<i>Dej Calatori → Tirgu Mures</i> 150 km, 54 noduri de plecare	0,191 secunde (timp total) 0,003537 secunde (medie)
<i>Dej Calatori → Cluj Napoca</i> 59 km, 54 noduri de plecare	0,186 secunde (timp total) 0,003444 secunde (medie)
<i>Cluj Napoca → Dej Calatori</i> 59 km, 62 noduri de plecare	0,220 secunde (timp total) 0,003548 secunde (medie)

BIBLIOGRAFIE:

- [1] E. Dijkstra – *A note on two problems in connection with networks* / Numerische Mathematik 1, 1959, pp 169-271
- [2] D. B. Johnson – *A note on Dijkstra's shortest path algorithm* / Journal of ACM 20, 1973, pp 385-388
- [3] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, Christos Zaroliagis – *Toward realistic modeling of time-table information through the time-dependent approach* / Electronic Notes on Theoretical Computer Science 92 no 1, 2003
- [4] Frank Schulz, Dorothea Wagner, Christos Zaroliagis – *Using multilevel graph for time-table information on railway systems* / Editors: D. Mount and C. Steiri, ALENEX 2002, LNCS 2409, pp 43-59.