THE ARCHITECTURE OF A WEBSERVER FOR TIMETABLE PROBLEM

Author: Ion Cozac
'Petru Maior' University Tirgu-Mures
e-mail: cozac@upm.ro

Abstract

We described in a recent paper [7] how to determine near optimum path in a given timetable graph. This paper argues why we should use the graph theory to solve the timetable problem, showing the main advantages of using graph theory. Next we describe the architecture of our web server that solves this problem, and we emphasize the advantages of the technology which has been used.

1 The timetable problem

This paper describes an approach for a time-table information system, which supports on-line queries to find quickly routes between two stations. This problem has been studied intensively in the last years; a rich survey paper was written by Müller-Hannemann et al [4]. Our study is based on the model designed by Schulz et al [5] [6].

A railroad net is modeled using a symmetric, weighted graph $G=(V,E,w)$:

– V is a set of railway stations;

– E is a set of direct links between stations: if $(x,y)\in E$ then there is no other station between x and y;

– w is the weight (length) function, $w : E \to \mathbf{N}^*$; $w(x,y) = w(y,x)$ for each $(x,y)\in E$.

For each train we know the complete time-table, that is, for each station we know the arrival time and the departure time. The first problem that must be solved is how to group these

informations into a complex structure (the so called ***time-table graph***), such that we can find (near) optimum paths between any two given stations, using only the train routes.

Suppose the railway net has three stations (*Sa*, *Sb* and *Sc*), and three trains have the following time-table:

– *T*1: (*Sa*, 8:00) → (*Sb*, 8:15, 8:20) → (*Sc*, 8:30);

– *T*2: (*Sb*, 12:00) → (*Sc*, 12:45, 12:50) → (*Sa*, 13:10);

– *T*3: (*Sc*, 14:00) → (*Sa*, 14:15, 14:20) → (*Sb*, 14:35).

For each station, and for each arrival and departure time, we define the set *V'* of nodes:

– nodes assigned to *Sa*: n_1(*T*1,8:00), n_2(*T*2,13:10), n_3(*T*3,14:15), n_4(*T*3,14:20);

– nodes assigned to *Sb*: n_5(*T*1,8:15), n_6(*T*1,8:20), n_7(*T*2,12:00), *n*8(*T*3,14:35);

– nodes assigned to *Sc*: n_9(*T*1,8:30), n_{10}(*T*2,12:45), n_{11}(*T*2,12:50), n_{12}(*T*3,14:00).

We also define the set *E'* of arcs:

– arcs that show successive moments of a particular train route: (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);

– arcs that show possible connections between different trains (transfer arcs): (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}).

A particular arc of the second type shows a connection between two nodes that are related to the same station and different trains. The first node is of arrival type and the second is of departure type.

The time-table graph *G'* = (*V'*,*E'*,*time*,*dist*) is directed and asymmetric (figure 1). A particular node *x'*∈*V'* is a triplet (*s*,*m*,*t*): *s* is the station, *m* the train (machine), *t* the departure or arrival time.

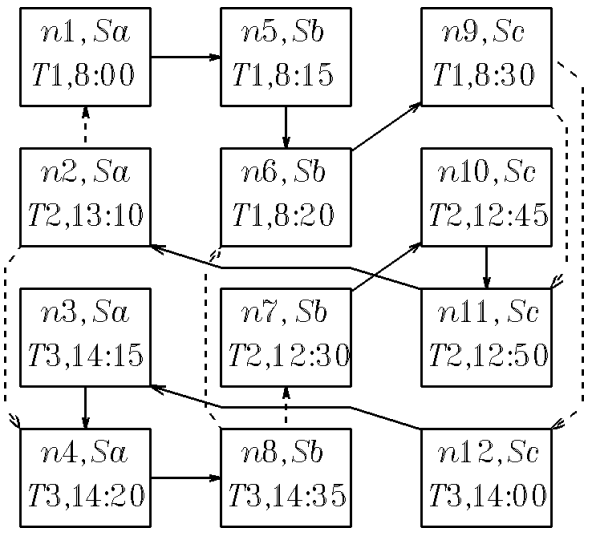

Figure 1. Time-table graph: example

The *time* and *dist* function are defined as follows:

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

$dist : E' \rightarrow \mathbf{N}$; $dist(x',y')$ = minimum distance between $s(x')$ and $s(y')$

where $t(x')$ is the *t* component of the vertex x', and $s(x')$ is the *s* component of the vertex x'.

The minimum distance is computed according to the railroad graph *G*. The values of the functions *time* and *dist* are computed once, in the preprocessing phase.

Our proposed model suggests that, for each station, one builds arcs between an arrival node (first vertex) and a departure node (second vertex). This model allows us to manage very easy any transfer.

2 What if we don't use the graph theory

Our previous paper [7] describes in detail how to determine optimum paths in this time-table graph. The searching process is divided into two steps, due to the huge size of the time-table graph. The first step determines the minimum distance from departure station to arrival station, which is denoted by *dm*. This step uses the grapg *G*, whiuch is very small, so the answer is found instantly. The second step uses the time-table graph *G'* to determine optimum paths from each departure vertex (wich is related to the departure station) to the first encountered arrival vertex (wich is related to the arrival station).

A recent paper [16] describes in detail the difficulties one may encounter if the application doesn't use the graph theory to solve the time-table problem.

First of all, if a new route is needed for a new train, the human operator must enter manually distances between consecutive stations. Due to human errors, the distance between two distinct stations may vary from one train to another, even if they cover the same route !

Example. What is the distance between Ciulnita CAM h and Baraganu: 30 km or 31 km ? From Ciulnita CAM h to Baraganu, the distance is 20 km. From Baraganu to Ciulnita CAM h, the distance is 31 km. These two distinct distances are related to distinct prices !

These inconsistencies are eliminated by using graph theory, because every distance is computed using very efficient implementation of Dijkstra's algorithm [2] [3] [4].

Non-using the graph theory causes another drawback: it is very difficult to determine complex routes, which are joints of more than two trains. Example: our website is able to determine routes from Tirgu Mures to Ionel; every route is joint of three or four trains. All the rest of romanian websites are incapable to determine such routes.

The difficulty in determining complex routes may be exploited by an attacker who may be

tempted to start distributed denial of services attacks. Because our webserver is written in C, it uses a special architecture that allow us to get the answer very quickly. This architecture will be detailed in the next section.

3 The architecture of our web server

The Internet address of our implementation is http://193.226.19.29:1026 . The webserver is written in C and runs on a Linux Fedora operating system.

This server is the visible component of a very complex application. The architecture of this application is similar to the architecture of a search engine [12].

The first component of this application is the data collector, which is similar to a crawler / spider / robot [13]. This component collects informations about updates on trains time-table from the official website. There is a significant difference between this collector and the search engine crawler (spider). The search engine robot gets informations from various websites which may be located anywhere in the world, but follows only explicit links. Our data collector gets informations from the official website, and sends queries for each train: it simulates the human operator that fulfills a particular form, in such a way that we finally obtain complete informations about all trains.

The functionality of this component will be extended in such a way that we will be able to modify manually the time-table and / or the route of a selected train.

The second component creates the time-table graph using the collected informations. This component is similar to the indexing component of a search engine [14]. First, we build the list of all vertices of the time-table graph. Each vertex is a triplet (s,t,h) (station, train, arrival / departure time); all these vertices are indexed. Second, we build all the arcs that shows the routes of all trains. Third, we identify all the stations that may be used for transfer (from one train to another), and in these cases we build supplementary arcs. Finally, we send an update request to the public web server.

The architecture of our webserver is of monolythic type [8] [89] [10] [11] [12], and is presented below.

Create socket on port 1026 for internet connections;

Load into main memory: railway net graph, time-table graph, html templates for web responses;

Repeat

Wait for internet connection;

Parse the request and build a list of type: Parameter = value

Case request of

Stop: stop the webserver;

Update: reload into memory all the important data;

Get / **Post**:

Case request of

Home page: send Home page;

Info about station *St*:

build the list of trains that are related to station *St*;

send the list of trains;

Info about train *Tr*:

send the route and the time-table of train *Tr*;

Routes from *Dep* to *Arr*:

determine routes from *Dep* to *Arr*;

send the routes;

end case

end case

until true;

Conclusions

The advantages of using the graph theory to solve the train's time-table problem have been presented recently:

- we avoid inconsistencies that are related to distances between two arbitrary stations;
- we are able to determine quickly very complex routes, which are joints of two or more trains.

The monolithic architecture of the public web server, described above, has an important advantage. All the important informations (railway net graph, time-table graph, html templates) are kept in main memory. If the web server receives simultaneously many requests, each request is processed by different processes (forks). All the important informations are stored only once in the main memory [8] [9] [10] [11], and are available for all processes. The consequence is a very high response speed, because there are no communications between different processes, as if it was the case of a PHP + MySQL application.

If we are concerned about security problems, this architecture has also an important advantage. Due to the high response speed, the webserver is not vulnerable to distributed attacks [11].

Bibliography

[1] Thomas H Cormen; Charles E Leiserson; Ronald L Rivest; Clifford Stein - Introduction to Algorithms, second edition, MIT Press and McGraw-Hill.?ISBN 0-262-53196-8

[2] Edsger Dijkstra - A note on two problems in connection with networks, Numerische Mathematik 1, 1959, pp 169-271

[3] D B Johnson - A note on Dijkstra's shortest path algorithm, Journal of ACM 20, 1973, pp 385-388

[4] Mathias Müller-Hannemann; Frank Schulz; Dorothea Wagner; Christos Zaroliagis - Time-table Information: Models and Algorithms (february 2006)

[5] Evangelia Pyrga; Frank Schulz; Dorothea Wagner; Christos Zaroliagis - Toward realistic modeling of time-table information through the time-dependent approach, Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003); Electronic Notes on Theoretical Computer Science, vol 92, pg 85-103, Elsevier, 2004

[6] Frank Schulz; Dorothea Wagner; Christos Zaroliagis - Using multilevel graph for time-table information on railway systems, Proceedings 4th Workshop on Algorithm Engineering and Experiments (ALENEX 2002), volume 2409 of LCNS, pp 43-59 (Editors: D. Mount and C. Steiri)

[7]http://www.upm.ro/facultati_departamente/stiinte_litere/conferinte/situl_integrare_europeana/Lucrari2/Ioan%20Cozac.pdf

[8] Mark Mitchell; Jeffrey Oldham; Alex Samuel - Advanced Linux programming, New Riders Publishing 2001

[9] Kurt Wall; Mark Watson – Linux programming unleashed, Sams Publisher 1999

[10]James Stanger; Patric T Lane - Hack proofing Linux: a guide to open source security, Syngress Publisher 2001

[11]Warren Gay – Linux socket programming by example, Que Publisher 2000

[12] http://en.wikipedia.org/wiki/Search_engine

[13] http://en.wikipedia.org/wiki/Web_crawling

[14] http://en.wikipedia.org/wiki/Index_(search_engine)

[15] http://en.wikipedia.org/wiki/Web_search_query

[16] Ion Cozac - Using graph theory to solve the time-table problem; Inter-eng, Interdisciplinarity in Engineering International Conference, November 2009, 'Petru Maior' University of Tirgu-Mures