Transportation Planning and Technology, February 2006
Vol. 29, No. 1, pp. 25–42

Routledge
Taylor & Francis Group

ARTICLE

Adapting a Search Algorithm for the Spanish Railway Network

J. MACÍAS-GUARASA, R. SAN-SEGUNDO, J. M. MONTERO, J. FERREIROS, R. CÓRDOBA, F. FERNÁNDEZ, L. F. D'HARO & J. M. PARDO

Grupo de Tecnología del Habla, Departamento de Ingeniería Electrónica, Universidad Politécnica de Madrid, Spain

(Received 20 December 2004; Revised 21 December 2005; In final form 17 January 2006)

ABSTRACT This article describes a search algorithm adapted to the Spanish Railway Network for generating as many traveling options as possible between two railway stations. This algorithm (Warshall's algorithm) uses connecting matrices to find all possible railway journeys. The Spanish Railway Company has imposed severe restrictions: less than 1 second per query in a 600Mhz processor PC with 32Mb RAM and 150Mb hard disk free memory. The final average time for a simple query is around 0.25 seconds and the whole memory consumption is 127Mb. The final implementation has been divided into 3 modules. In the first module, we store additional information in the connecting matrices to accelerate the later search, proposing several strategies for reducing thier size. The journey option calculation module accesses the matrix information and composes the traveling options. Finally, in the filtering module we describe the selection criteria considering the algorithm embedded in a general information service.

KEY WORDS: Search algorithm; railway; journey option search; connecting matrices; train routing

Correspondence Address: R. San-Segundo, Grupo de Tecnología del Habla, Departamento de Ingeniería Electrónica, E.T.S.I. de Telecomunicación, Universidad Politécnica de Madrid, Ciudad Universitaria sn, 2804 Madrid, Spain. E-mail: lapiz@die.upm.es

ISSN 0308-1060 print: ISSN 1029-0354 online
DOI: 10.1080/03081060600585145

Introduction

The Spanish railway company, RENFE, is investing significantly in renewing its infrastructure. As part of this process, one particular issue has been the improvement of the journey search algorithm in order to offer better automatic information services. Currently, RENFE provides information on direct journeys (without connections) (Infotren, 2000) and they want to offer information on all possibilities: journeys with and without connections. Not only is it necessary to obtain journey alternatives, but also the algorithm should sort them from the 'best' to the 'worst' suitability. This aspect requires a journey quality measure to be defined. This measure will depend on the final service that RENFE will develop.

The search algorithm should have the following characteristics (San-Segundo *et al*., 2000):

1. The algorithm should generate as many possibilities as possible for traveling by train between any two stations in Spain (with at least three connections). The idea behind generating as many alternatives as possible is necessary because the algorithm must be independent of the final automatic service. After searching for all alternatives, the automatic system applies a filter to select the most interesting journey possibilities for the service provided. For example, if RENFE wants to develop a service for tourist train information, the criteria will be different from that of a general journey information service. In a general service, the duration of the journey could be an important factor while, for the tourist service, train comfort could have more relevance.
2. The search algorithm should work in real time. That means that all alternatives should be generated in less than 1 s. This requirement is necessary to avoid the user abandoning the service. This is of particular importance when the information service is provided by telephone where the user pays an additional telephone fee for this service. In our case, this extra fee depends on the duration of the call: RENFE provides the information service through a special telephone number.
3. For each alternative, the algorithm should provide all of the characteristics: departure and arrival times for all legs, connecting stations, type of train, whole journey and individual leg durations.
4. With this detailed description for each alternative, the algorithm should permit the sorting and filtering criteria to be defined easily. The journey quality measure must be easy to modify.

In the literature, there are many papers addressing the problem of optimized train routing and scheduling (Cordeau *et al.*, 1998, 2001;

Lindner, 2000; Liebchen & Peeters, 2002). These consider that train timetables are modifiable. In our case, the problem is different: we have to find the journey options given train routing and scheduling information.

This article is organized as follows: the next section describes the different algorithms considered and the one to be implemented. In the third section, the main concepts concerning the Spanish railway network are described. In the fourth section, we describe the algorithm implementation: the connecting matrices and the travel option generation. Finally, the fifth section summarizes the main conclusions.

Search Algorithm

In order to select the search algorithm, we analyzed several algorithms used for routing information packets in telecommunication networks (Sedgewick, 1990; Bertsekas & Gallager, 1992; Tanenbaum, 1996). These algorithms are: Dijkstra's, Floyd–Warshall's, Bellman–Ford's, Distributed Bellman–Ford's as well as Warshall's for directed graphs.

Dijkstra's algorithm is an iterative algorithm applied over all network nodes. For each node, the algorithm computes the minimum distance from this node to the rest of the nodes. In order to calculate the distance for any pair of nodes, the algorithm must be applied to all the nodes. In this case, the algorithm complexity is N^3. During the calculation process, it is possible to store the sequence of nodes that make up the minimum distance path for each pair of nodes. This algorithm has the advantage that all calculations can be carried out in a previous process before the search but it has the problem that it only obtains the 'best' path instead of all possible ones. Second, we considered the Floyd–Warshall algorithm. Similar to Dijkstra's, it computes only the 'best' path between two nodes and has a complexity of N^3. The algorithm generates several connecting matrices considering a greater number of nodes as intermediate nodes. The algorithm ends when the matrix considers all possible nodes as intermediate nodes. The final matrix has the minimum distance between any pair of nodes. During the process, it is possible to store the intermediate nodes for each 'best' path between any pair of nodes. The Bellman–Ford algorithms compute the 'best' path considering, in each iteration, a greater number of intermediate nodes. The algorithms generates several possible paths but the complexity is higher – N^4. This type of algorithm is very useful when there is no knowledge about the network.

Warshall's algorithm was used for the directed graphs. This algorithm has a complexity of N^3 and it permits all the possible paths between any pair of nodes to be calculated (with any number of

intermediate nodes). It is based on the idea 'if there is a path from A to B and another path from B to C, then there is a path from A to C'. For obtaining a path through several intermediate nodes, it is necessary to iterate the algorithm as many times as intermediate nodes considered. The algorithm details are shown in the following steps:

STEP 0: Initialization. We generate a matrix for direct paths (without connections) between nodes, M^0. Each element in the matrix, $M^0(x,y)$, represents the number of alternatives for going from node 'x' to node 'y' directly. The row i provides information on the direct paths from node i to all the nodes. On the other hand, column j represents the direct paths from all the nodes to node j.

STEP i: Matrix for paths with 'i' connections. The connection matrix M^i is obtained by multiplying the matrix M^0 i times: $M^i = (M^0)^{i+1}$. Each element in the matrix, $M^i(x,y)$, represents the number of alternatives for going from node 'x' to node 'y' with 'i' connections. Figure 1 shows a network example with the connecting matrices for direct paths, 1 and 2 connections.

In this example, for going from node B to node G, there is one direct path, one path with one connection and two paths with two connections. But not only is it necessary to know if there is a path between two nodes, we also need to know which nodes are the connections. In Figure 2 we show the algorithm to calculate the connection node in the one connection path from B to G. The algorithm carries out an 'AND' operation with the column considering G as final

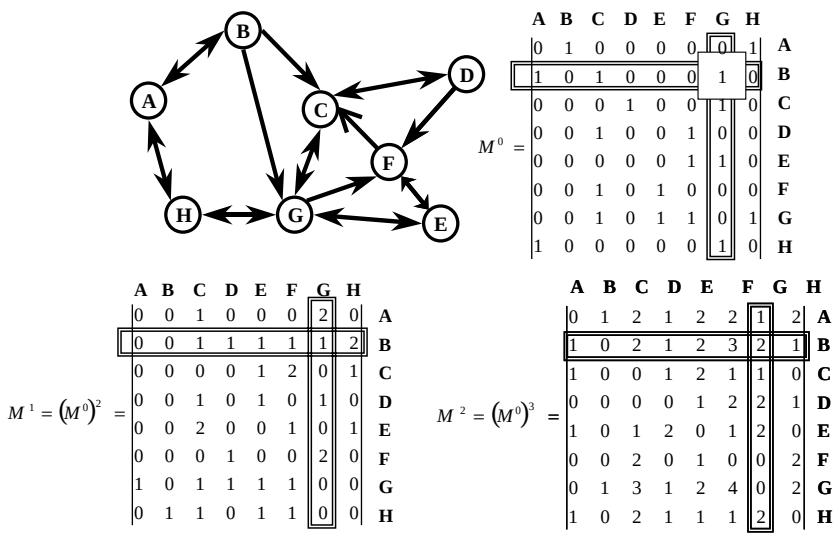

Figure 1. Network example with connecting matrices

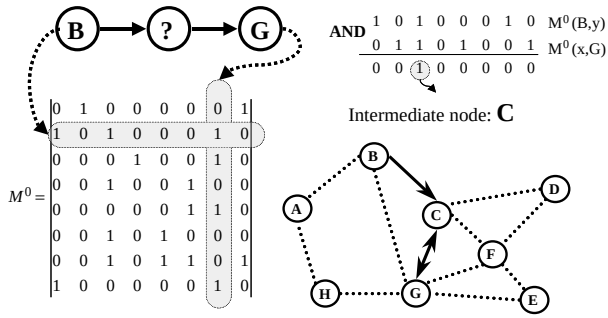

Figure 2. Intermediate node calculation for the one connection path between B and G

node M^0(x,G), and the row in which it considers B as initial node M^0(B,y). This strategy provides only one possible connection node C. When there are several intermediate nodes the 'AND' process must be repeated as many times as there are intermediate nodes to be computed. In Figure 3 we show the algorithm for calculating the connection nodes in the two connection path from B to G. In this case, the 'AND' processes involve columns and rows from different connecting matrices.

In this case, to calculate the first intermediate node, we carry out an 'AND' process between the column that it considers G as final node in a one connection path M^1(x,G), and the row that it considers B as initial node in direct paths M^0(B,y). The second 'AND' is similar to the process shown in Figure 2. This 'AND' offers a second alternative as a second intermediate node: B. This solution is not valid because initial and end nodes can never behave as intermediate nodes. This algorithm permits us to calculate all possible paths between any two nodes and the intermediate nodes.

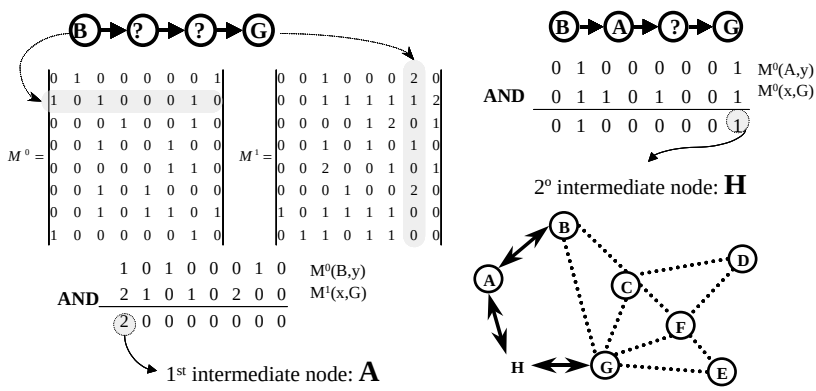

Figure 3. Intermediate node calculation for the two connection path between B and G

Previous Concepts and Definitions

In this section, we describe the Spanish railway network organization and the connecting data presentation. The main concept is the 'train working' (or 'train march'). Any train journey involves one or several train workings. A train working is a complete section of the railway network covered by a train and starting at a specific time. A train working description contains the followings fields (Figure 4):

1. Working code. This is a numerical code that identifies the train working in the working (or march) database. There are 1412 different workings.
2. Schedule code. This number defines the working schedule. A working schedule is based on day planning that specifies the days in which the train works: days when there is a train covering the stretch proposed. Not all workings are activated every day. In our database, there are 275 schedules and each working is associated with one of these schedules.
3. List of stations. In this list, there are four types of station: 'first station' (symbol >), 'end station' (symbol <), 'normal station' where passengers can get off/onto the train (symbol +) and 'train stop station' where passengers cannot get off/onto the train (symbol .).

For each station, its description contains four fields: station code, departure time, arrival time and logical distance from the previous station (the logical distance is not a real distance and is only used for ticket price calculation). The first station has no arrival time and the final station has no departure time. One point we must keep in mind for the following sections is that if we want to travel from station A to station B using a train working, station A must precede station B in the working description.

TRAIN WORKING CODE:	#50001
SCHEDULE CODE:	0026
FIRST STATION:	>17000,0800
DOWN/UP STATION:	+18000,0813,0815,0
DOWN/UP STATION:	+60400,0924,0925,152
DOWN/UP STATION:	+60600,1019,1020,135
STOP STATION:	.60800,1053,1054,80
DOWN/UP STATION:	+60902,1113,1114,37
DOWN/UP STATION:	+60905,1127,1128,18
END STATION:	<60911, 1155,41

Figure 4. Example of a train working

Another important concept we must comment on is the 'generic station'. In our geography, there are some cities with more than one train station. This fact would be no problem but RENFE wants to offer journey options with connections even between different stations sited in the same city. This means that passengers would have to find another means of transport for going from one station to the other. In order to consider this type of journey, a new concept has been defined: the generic station. A generic station is a group with all train stations sited in the same city. In the geography of Spain there are 999 train stations and eight generic stations (Alicante, Barcelona, Bilbao, Gijón, Madrid, Malaga, Seville & Valencia).

Algorithm Implementation

This section describes the search algorithm implementation for the Spanish railway network. In the search algorithm, we consider as nodes all train stations independently and the generic stations (i.e. a train station included in a generic station also counts as an independent node). So we have a total of 1007 nodes (999+8). This network is very complex: there are more than a million possible queries every day. In this description, we comment on the implementation details focusing on those that permit the algorithm complexity to be reduced. In Figure 5, we show a diagram of the search algorithm.

This algorithm is made up of three main processes: matrix generation; journey option calculation; and option filtering.

In the first process, the connecting matrices are calculated. These matrices contain not only information related to inter-nodal connectivity but they also store additional information that will help the

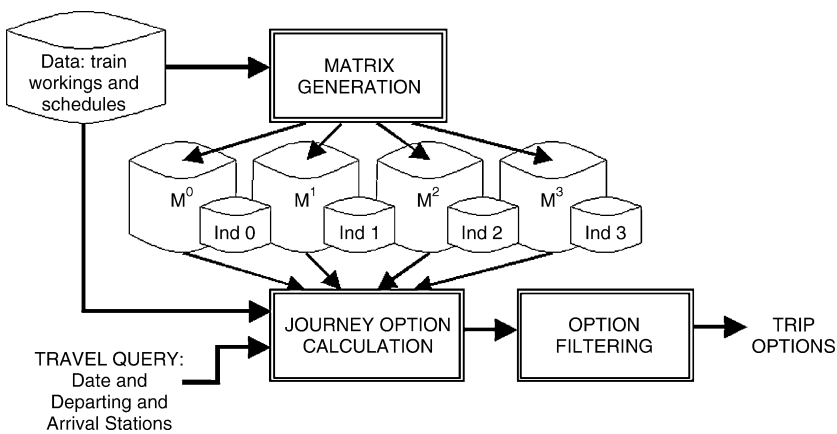

Figure 5. Diagram of the search algorithm

journey option calculation go faster. Apart from the connectivity matrices, the algorithm generates the same number of index matrices (Ind 0, Ind 1, Ind 2 and Ind 3). These matrices facilitate access to the previous ones.

The matrix generation is a background process and it is carried out only when the train workings or schedules are modified. Because of this, processing time is not a critical factor. In this case, the main problem is the storage memory for these matrices. The memory problem appears because, as we commented on above, these matrices contain additional data necessary to make the journey option calculation work in real-time. In this paper we present several strategies and solutions for selecting the additional data in order to keep the matrix sizes under a reasonable value.

The journey option calculation accesses the matrices information and puts together all the journey options between two train stations for a specific date. In this case the computing time is the main issue and the option calculation must be made in real time. This process uses the schedules to identify the activated train workings. The last module filters the journey options (in real-time) and presents the 'best' options sorted by a measure of quality. In this module, several measurements are possible. Although the final criteria depend on the final application, we will describe in below the criteria implemented for a telephone-based information service as an example.

Concerning the matrix sizes, RENFE imposed the following limitation: the hard disk space taken up by the algorithm must be less than 150 Mb. With respect to the option calculation and filtering modules, RENFE established that the time taken up by both modules for any query cannot exceed 1 s with a 600 Mhz processor with 32 Mb RAM. These limitations have oriented the solutions and strategies that were finally implemented.

In Figure 5 we have represented four connecting matrices: M^0, M^1, M^2, M^3. With this information, we can obtain (considering the algorithm described abvove) journey options with less than four connections. This number of connections is sufficient to obtain a high connectivity. In Table 1 we can see the connectivity depending on the maximum number of connections considered. As we can see, with three connections we can provide journey options for more than 99% of the queries. This has been another characteristic imposed by RENFE.

Matrix Generation

The matrix generation module computes the four connecting matrices (M^0, M^1, M^2 and M^3) and their respective index matrices (Ind 0, Ind 1,

Table 1. Connectivity depending on the maximum number of connections.

Maximum number of connections	0	1	2	3
Queries with connections	37 504	375 067	948 077	1 004139
Total number of queries (1007^2)	1 014 049	1 014 049	1 014 049	1 014 049
Percentage of queries with connections (%)	4	37	93	99

Ind 2 and Ind 3). As we commented before, in order to calculate the journey options in real time (less than 1 s), the connecting matrices must store additional information. This information occupies a large amount of memory so we cannot store it in RAM for the journey option calculation (we have a limit of 32 Mb). Because of this, we have created the index matrices. These matrices store pointers to the information stored in the connecting matrices. This way we can load the index matrices into RAM and provide fast access to the connecting matrices.

The index matrices are made up of 1 014 049 pointers (1007×1007). For each pointer we need a 32 bit number (byte position in M^i), so the memory taken up by each index matrix is around 4 Mb. For all the index matrices, we take up 16 Mb RAM. For the algorithm, the total RAM used is around 20 Mb, smaller than the limit imposed.

In the following sections, we describe the information stored in the connecting matrices. These data have been one of the most important issues. As we commented above, connectivity depends on the traveling date (not all the train workings are active every day). As the matrix generation is a very time consuming process and must be carried out only when the workings or schedules change. In this situation, the connecting matrices must contain information that does not depend on travel date. This characteristic increases significantly the amount of data. In the following sections, we describe strategies for reducing its size.

Direct journey matrix (M^0). In Figure 6 we show the information stored in M^0 for each possible journey. For each journey, we store all the train workings (and their respective schedules) connecting departure and arrival stations. At the end, we also include the minimum time required for a direct journey (minimum journey duration). This value will be useful for generating the rest of the matrices.

The first number is the number of workings that directly connect both stations. There are stations in Madrid and Barcelona where the

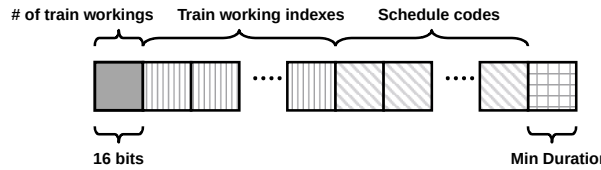

Figure 6. Data stored in M^0

number of train workings is very high. Because of this, we require a 16-bit number. After this, we include the workings. Instead of including the working codes (each code requires 32 bits), we define and store working indices associated with the codes. We have 1412 train workings so we only need a 16-bit integer per index. During all the algorithm modules, we use the workings index instead of the workings code. The next piece of data is the minimum journey duration. It is the minimum duration (in minutes) for any direct journey between both stations. As any train journey takes less than 48 h (2880 min), we include a 16-bit integer. Finally, we include the schedule codes associated with all workings. Each schedule code consumes 16 bits. These codes are not strictly necessary but storing the schedule codes has permitted us to save computing time and RAM use: it is not necessary to maintain a table linking working and schedule codes. This fact duplicates the M^0 size but it is irrelevant because the M^0 size is very small compared to the rest of matrices (see below).

As we showed in Table 1, the percentage of queries with direct journeys is very low (4%). This fact means that the M^0 size is very small: 725 Kb. For 96% of queries where there are no direct journeys, we do not store any information. An invalid pointer value is assigned to them in the index matrix (Ind 0). In our case, all the pointers are positive so we use −1 to indicate no direct journeys. As we can see, the Ind 0 matrix shows the connectivity and M^0 contains the information associated to this connectivity.

Matrices for journeys with connections (M^1, M^2, M^3). In these cases, the most interesting information is the itinerary: the train stations where the passengers have to change train (sequence of connecting stations). In order to reduce the time taken up in the next module, we decided to include itinerary information in the matrices. The itinerary calculation from the connectivity matrices consists, basically, of an 'AND' process between rows and columns from different matrices. This process is not very costly but is a real problem; additional checking is necessary that requires significant computing time. The additional processes are commented on as follows:In our system, we work with train workings that are not activated every day. Although the matrices show a journey

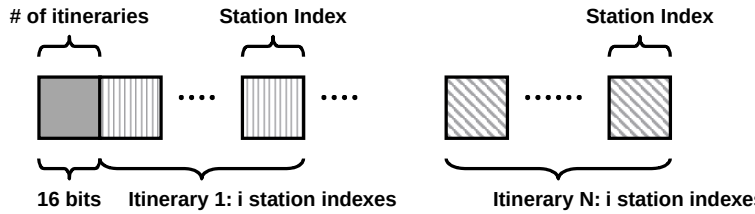

Figure 7. First version of the data stored in M^i with $i = 1$, 2 and 3

option through several connecting stations, there can be a mismatch between the train working schedules covering each leg that prevents the generation of a valid journey option for any particular date. These itineraries should be detected and eliminated.This algorithm can also produce itineraries with station repetition: one station can be considered as a connecting station twice in the same itinerary, or the departure/arrival stations are included as connecting stations. These itinerary options must be detected and discarded.This checking process must be carried out during the matrix generation where time consumption is not an important requirement. In the matrices, only validated itineraries must be stored. For the first implementation we decided to consider the following data format (Figure 7): The first number is the number of itineraries that connect both stations with 'i' connections. This number can be higher than 256 so we require a 16-bit number. We then include the 'i' station indices for each itinerary. Instead of including the station codes (each station code requires 32 bits), we define and store those station indices associated with the codes. We have 1007 station codes so we only need a 16-bit integer per index. During each of the algorithm modules we use the station index instead of the station code.

Even after the itinerary selection, the matrix sizes were extremely high, especially M^3 (see Table 2). Because of this, we decided to change the matrix format. Instead of storing the station indices, we decided to store an itinerary index (Figure 8). As we have 1007 stations, with three connections, we can obtain as much as $1007^3 = 1\,021\,147\,343$ different itineraries. We create an itinerary table and we assign a 32-bit index for

Table 2. Matrix size comparison.

	First format (Kb)	New format for M^3 (Kb)
M^1	6014	6014
M^2	130 044	130 044
M^3	529 011	354 636
Total	665 069	490 694

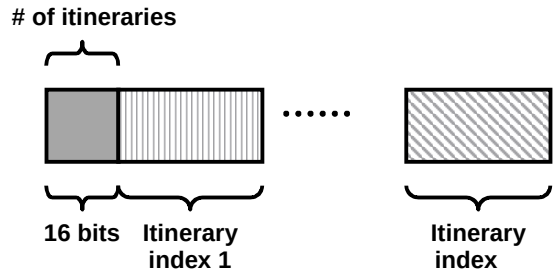

Figure 8. New format for M^3

each. The itinerary table is stored in a binary file. We only use this solution for M^3 (where the new format produces a 30% size reduction) but not for M^2 or M^1: in these cases, we get no size reduction. Table 2 shows the matrix sizes obtained.

As we show in Table 2, it is necessary to further reduce the matrix sizes. The following actions focus on reducing the number of itineraries stored. This fact can discard valid itineraries discarded and lose journey options. We tried to discard the worst itineraries. These actions are as follows.

The first action is to select connection stations. Instead of considering all train stations as possible connecting nodes, we make a selection. Our criterion has been the number of train workings that pass through the station. This is a measure of station connectivity. From the station list, we have selected the stations with highest connectivity. After this selection, the list was revised by RENFE who proposed to add/delete some of them. The final list contains 152 possible intermediate stations.

The aim of this action is to solve a problem we observed during implementation of the algorithm. In the Spanish network, there are large overlaps between different train workings: two workings can have up to 10 common train stations. This fact means that these 10 stations are all valid connecting stations: we obtain 10 different itineraries but the calculated journeys are equivalent (the same workings are involved). With this action, we drastically reduce this effect and the lost journey option is irrelevant.

This solution has made a significant reduction in the matrix sizes – from a total of 490 Mb (Table 2) to a total of 186 Mb. The lost journey option has been less than 1%. This fact reveals that the station selection has been carried out successfully: the selected stations really are important connecting nodes.

The second action consisted of sorting the itineraries and storing only the N 'best' ones in the matrices. For sorting the itineraries, we

considered two criteria: minimum journey duration and itinerary hierarchy. The minimum journey duration is the sum of the minimum leg duration for all the legs that make up the journey. The minimum leg duration is obtained from M^0 where we have stored this value in minutes at the end of each data. This measure provides an ideal journey duration throughout this itinerary. With these criteria we try to avoid extremely long itineraries through the Spanish countryside. For example, it is possible to obtain a journey from Madrid to Seville passing through Barcelona but this is absurd. The second measure is the itinerary hierarchy – the geometric average throughout the hierarchies of the intermediate stations that make up the itinerary. For the 152 possible intermediate stations, we have defined a hierarchy value. This is an integer number between 0 and 6 (0 for lowest hierarchy and 6 for highest hierarchy). This assignment has been carried out by attending to the number of train workings that pass through each station. Just as in the previous case, the station hierarchy was referred to and revised by RENFE.

To sort and select the itineraries we combined both criteria. This process consists of two steps carried out independently for each matrix: M^1, M^2 and M^3. First, we sort the itineraries (with i connections) using the minimum duration. We obtain the best itinerary and select those that have a minimum duration less that 300% of the 'best' itinerary duration. In the second step we sort the selected itineraries by their hierarchy, storing the N 'best' in the matrix. In Table 3, we show the journey loss and matrix size depending on the number of itineraries stored.

For the journey lost calculation, we have evaluated all journeys between all two stations for two days (Wednesday and Saturday): around 2 million queries (Chequea, 2000). We consider two journeys as equal when the train workings for covering the legs are the same in both cases. With these criteria, we permit different stations for the itinerary connections but they are equivalent journeys: of the same duration and quality.

Table 3. Journey lost and matrix sizes depending on the number of itineraries stored.

	N = 5		*N* = 10		*N* = 15		*N* = 20	
	Journey loss (%)	Matrix size (Mb)	Journey loss (%)	Matrix size (Mb)	Journey loss (%)	Matrix size (Mb)	Journey loss (%)	Matrix size (Mb)
M1	15	2	5	3	2	3	1	3
M2	22	19	9	31	5	41	3	49
M3	40	21	20	40	15	59	13	79
Total	34	42	18	74	9	103	8	131

Considering the results presented in Table 3, we decided to consider $N = 15$. In this case, we have a total matrix size of 103 Mb with a journey lost of 9%. RENFE imposed a limit size of 150 Mb for all the application files: executable file, table and additional files, index matrices (16 Mb) and connecting matrices (103 Mb). Considering $N = 20$ the total matrix size increases around 30% but journey lost reduction is very low.

Before finishing this section, it is necessary to comment that for queries where there are no journeys with i connections, we do not store any information in M^i. An invalid pointer value is assigned to them in the index matrix (Ind i). As all the valid pointers are positive, we use the -1 value. The Ind i matrix shows the connectivity and M^i contains the itineraries associated to this connectivity.

Journey Option Calculation

This module obtains the journey options for a traveling query: departure station, arrival station and departure date. It uses the information contained in the connecting matrices. Now, the computing time is a critical factor.

The calculation process is divided into two parts: direct journey calculation and journey with connections computing. The first step is to calculate the direct journeys for the required query. This module accesses matrices Ind 0 and M^0, and obtains the train workings (with their schedules) that connect both stations directly. After this, we check which workings are active for the specified date. These workings define the possible direct journeys. We obtain the departure and arrival times from the train working descriptions.

In the train working checking process we find a special characteristic that increases the difficulty. In the Spanish railway network, there are train workings taking one or two days but the activating information is associated only with the first one. Solving this problem has been possible because all train workings have a duration of less than 24 h (this characteristic is imposed by working conditions at RENFE). Knowing the journey departure time (TDT) and the train working start time (MBT), it is easy to decide if the journey is included in the first or second working day (Figure 9).

The second step is to compute the journeys with one, two or three connections. From M^i matrices, we load the different itineraries. For each itinerary, we obtain the train workings that connect the intermediate nodes using M^0. Just as in the previous step, we have to calculate the train working which activates all the legs that make up the whole journey. In this process, it is necessary to keep two effects in mind: a train working can cover two days but the activating

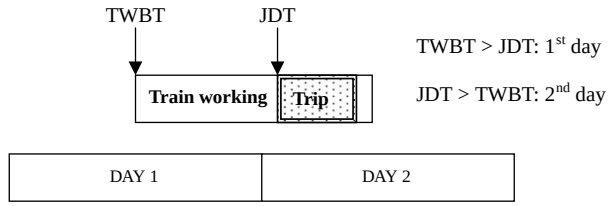

Figure 9. Calculation of the train working activating day

information refers to the first day (Figure 9), and the whole journey (that means several train workings) can also cover several days. These two effects are considered when obtaining the train working activating dates respective to the query date.

To combine the sections of train workings that form the whole journey, we concatenate them by considering two details. The first is that we have to guarantee passengers have a minimum connecting time. This time is different if the connections are made at the same station or the connection implies different train stations (see the generic station concept in above). In our program, the minimum time depends on the connecting station and it is modifiable by the user. The second detail is how to choose a train working in which there are several workings that cover a leg. In our case, for the first leg we sort train workings by departure time but for the remaining legs we sort them by arrival time. This way we obtain the journey with the lowest time duration (an example in presented in Figure 10).

Journey Option Filtering

In this process, we filter and sort the journey options using several criteria. These criteria depend on the final application of the algorithm. This section describes the criteria used in our case for a general information service over the telephone.

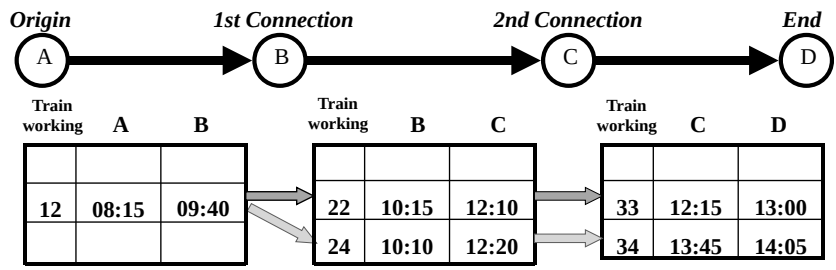

Figure 10. Journey calculation from sections of train workings

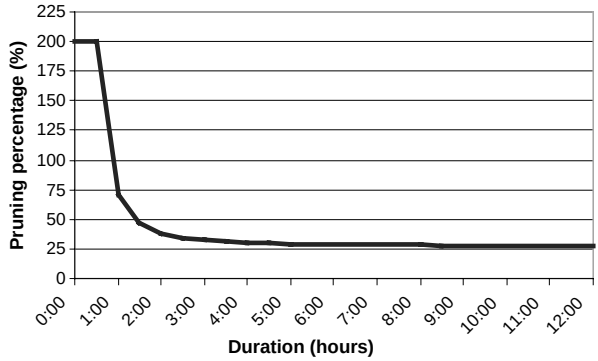

Figure 11. Pruning percentage vs. minimum journey duration

The filtering process is carried out by attending to the journey duration. All the options exceeding a percentage of the minimum duration journey are discarded directly. This percentage is different depending on the minimum duration. In Figure 11, we show the relationship between minimum duration and the percentage considered to prune the journey options. As we can see, when the minimum duration is lower, the percentage increases.

Finally, we sort the journey options by considering the measurement of cost. Table 4 shows the description for the digits in this cost measurement. First we discriminate between direct journeys and journeys with connections, then we consider journey duration and, finally, the number of connections.

As we commented before, the journey option calculation and filtering must be carried out in real time (less than 1 s). In our case, thanks to the data stored in the matrices, these two processes take an average time of 0.25 s (with a 600 Mhz processor). This value is four times smaller than the limit imposed by RENFE. Our algorithm consumes a large amount of memory but it is very fast independent of the query.

Finally, we want to comment on another utility implemented in our system. It consists of giving the possibility to define a desired intermediate station (IS) for the journey. In our algorithm, the connecting matrices (M^0, M^1, M^2, M^3) contain a limited number of

Table 4. Digits for the cost measure.

1st digit	0 for direct journeys and 1 for journeys with.
2nd and 3rd digits	Number of duration hours (04:15).
4th and 5th digits	Number of extra duration minutes (04:15).
6th digit	Number of connections.

train workings or itineraries so it can transpire that the intermediate station is not included in any of these workings or itineraries. In these cases we have implemented a utility that concatenates the journey options obtained with two queries: Origin to IS and IS to End. From these two queries, we only use direct journeys and journeys with one connection. This way the final journey has as many as three connections. The time consumption for this utility remains less than one second.

Conclusion

In this article we have described how we have adapted a search algorithm for the Spanish railway network: using the Warshall algorithm for directed graphs. This algorithm uses connecting matrices to find all possible journeys that go from one station to another. The requirements imposed by RENFE (the Spanish railway company) forced us to implement a very fast algorithm but with a high memory consumption. For any query, the time is less than 1 s with a 600 Mhz processor: the average time for a simple query is around 0.25 s. Total memory consumption was 127 Mb (less than the limit imposed: 150 Mb).

This implementation has been divided into three modules: matrix generation, journey option calculation and journey option filtering. In the first module, we have focused on the additional information stored in the connecting matrices. The main problem has been the memory taken up by these matrices. Several strategies for reducing this memory have been presented. The greatest reduction has been obtained by limiting the intermediate stations: the matrix sizes were reduced by more than 60% (from 490 to 186 Mb). This effect has been possible as a result of the high overlap between train workings. Further strategies have been implemented but they produced journey option losses – in the final implementation, we lost 9% of journey options. This loss is mainly focused on the three connection journeys (15%). For two and one connection journeys the loss was 5 and 2%, respectively. For direct journeys there was no loss.

Conversely, due to the RAM limit (32 Mb), we have defined index matrices associated with the connecting matrices that permit quick access to them. The index matrices consume 16 Mb and can be loaded into RAM. These matrices not only provide quick access, but also contain the connectivity information.

The journey option calculation module accesses the matrix information and puts together the traveling options. Special attention must be paid to train working activation and concatenation. The selecting criteria are applied in the filtering module. In this paper, we also

described the criteria for a general information service over the telephone.

Acknowledgements

This work has been partially supported by the grant 2FD1997-1062-C02 (EU and Spanish CICYT). Authors want to thank the contributions of colleagues at GTH and RENFE company. Authors also want to thank Mark Hallett for the final revision.

References

Bertsekas, D. & Gallager, R. (1992) *Data Networks* (Upper Saddle River, NJ: Prentice-Hall).

Chequea (2000) Software for journey evaluation. Version 3.

Cordeau, J. F., Toth, P. & Vigo, D. (1998) A survey of optimization models for train routing and scheduling, *Transportation Science*, 32, pp. 380–404.

Cordeau, J. F., Soumis, F. & Desrosiers, J. (2001) Simultaneous assignment of locomotives and cars to passenger trains, *Operations Research*, 49, pp. 531–548.

Estaciones Comerciales (2000) *Anuario de RENFE*. Maps of the Spanish Railway Network. Annual Revision (Madrid: RENFE).

Infotren (2000) Software for direct journey queries developed by RENFE, version 75.

Liebchen, C. & Peeters, L. (2002) Some practical aspects of periodic timetabling, In: P. Chamoni, R. Leisten, A. Martin, J. Minnemann & H. Stadtler (Eds) *Operations Research 2001,* pp. 25–32 (Berlin: Springer).

Lindner, T. (2000) Train schedule optimization in public rail transport, PhD dissertation, TU Braunschweig.

San-Segundo, R., Macías-Guarasa, J. & Salido, M. A. (2000) Especificaciones del nuevo motor de búsqueda para la nueva aplicación infotren, RENFE-UPM report, RENFE, Madrid.

Sedgewick, R. (1990) *Algorithms in C* (Reading, MA: Addison–Wesley).

Tanenbaum, A. S. (1996) *Computer Networks* (Upper Saddle River, NJ: Prentice-Hall).