Intelligent Railway Information System

Nupur Kothari, 98254
Nupur.Kothari@iitk.ac.in

under the supervision of
Dr. Dheeraj Sanghi
dheeraj@cse.iitk.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology
Kanpur 208 016, INDIA

April, 2002

Abstract

Due to the vastness and complexity of railway networks in India, it is a tough problem to find routes connecting two stations. This paper describes RIS, a project aimed at developing an Intelligent Railway Information system which finds direct as well as indirect rail routes between stations and displays the best ones based on a quality metric obtained from various user preferences.

1 Introduction

Due to the vastness and complexity of railway networks in India, it is a tough problem to find train routes connecting two stations. The complexity of the problem increases manifold when the two stations are not connected by direct trains but however have indirect routes connecting them, i.e. have trains connecting each to a common station. This complexity is further raised if the users choices, i.e. choices of timing, intermediate stations, cost etc. are taken into account and only those routes are listed which satisfy these constraints.

There is already online software existing to find routes consisting of direct trains between two stations and look up vacancies on those trains [1]. However, most of the stations in India are not connected by direct trains but have a huge number of indirect routes between each other. These are not given by the existing system. Also this system does not take into account the users preferences of time etc in mind while generating routes. Thus this software is not really of much use in route finding except maybe as an automated railway timetable and enquiry system for vacancy information and travel rates.

This project has been aimed at developing RIS, an intelligent railway information system which can be accessed online via the internet by users. It not only gives direct routes but also indirect routes and based on the users input tries to convert the users choices of time of arrival and departure, class of travel, interval between connecting trains, number of intermediate stations etc. into a quality metric and based on this metric generates the routes that satisfy it best. The routes are generated by either considering the direct trains, or by considering trains that connect the two stations to a common station. A list of important stations is maintained for each station which is search for such common stations. This reduces the search space greatly which would otherwise be the rest of the stations.

Also in the process, a code for hindi words

written in English similar to phonix has been developed which is basically used to correct user mistakes while translating hindi station names etc to English to provide user friendly behavior in the RIS.

2 Design Overview

RIS has been designed as consisting of the following major modules:

- Database Management System: This module provides facilities to the Administrator to maintain the RIS database and keep it up-to-date. It has been kept offline to ensure security of data and disallow unauthorized changes.

- Querying system: This module provides facilities of route search etc. using data obtained from the RIS database to the users. It has been implemented as a web based application so that users can access the route search service from the Internet itself and do not need to store the database and install this software.

These will be discussed in detail in later sections. iRIs has been completely designed in JAVA. JAVA was chosen because of its Object Oriented features, platform independence and also because of the ease with which Graphical User Interfaces can be designed in it.

2.1 Database Design

The RIS database stores information about cities, stations and trains, which is used to generate routes between stations and also various cost and distance information. Earlier, a database built by [2] was used for testing purposes but finally, due to outdated and incorrect information, typographical errors and inconsistencies, a new database was built from scratch.

For each city the information stored is its name, code, name soundex, and stations located in it. The city name is the primary key here.

For each station, the information stored is its

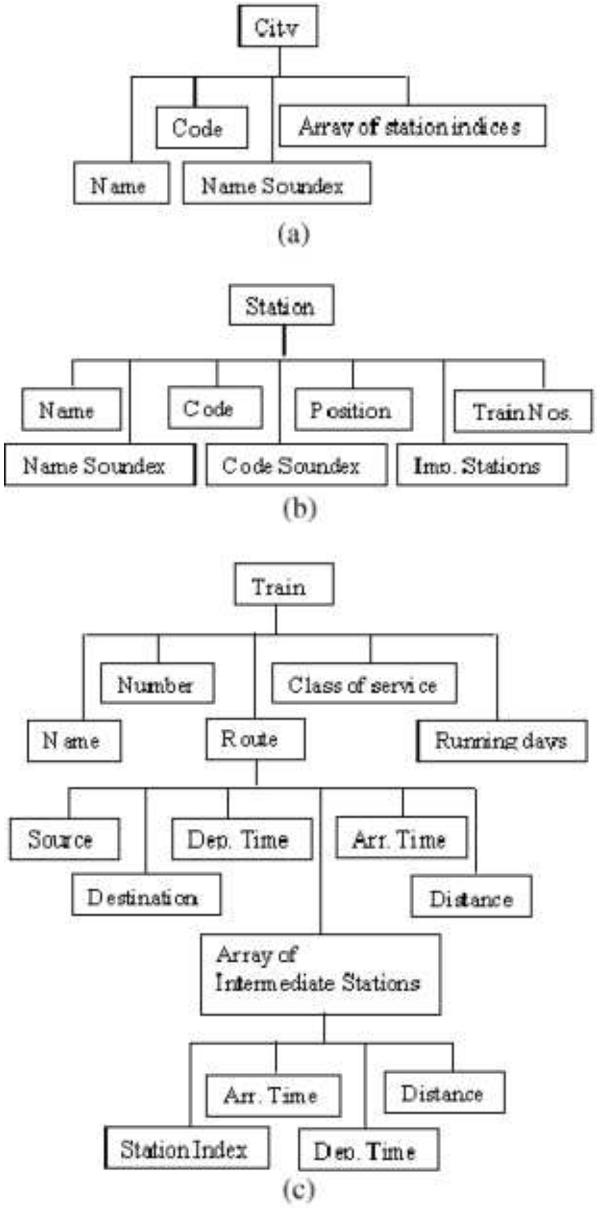

Figure 1: Database Design: Information stored for each (a) city (b) station (c) train

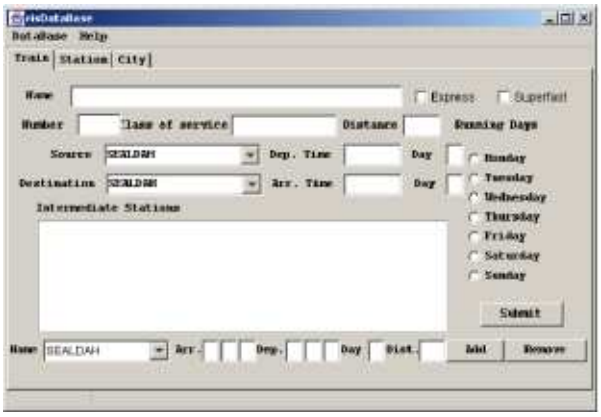

Figure 2: The RIS Database Management Application

Figure 3: Menus of the RIS DB Mgmnt System

name, name soundex, short name, latitude, longitude, list of trains passing through it, list of important stations.

The station name is the primary key here.

For each train, the information stored is its name, number, source, destination, intermediate stations, arrival and departure time at each station, distance of each intermediate station from source, days of running, classes of service available etc. The train number is the primary key here.

The database does not use any standard Database Management package; instead, the data is stored in binary files, using the object serialization facility of Java. The database contains information in the form of arrays of cities, stations, and trains. These arrays are simply written into separate files and every time the system is restarted, the arrays are loaded into memory.

3 DB Management System

The Database Management System is offline to ensure the security of the database. It provides facilities for the administrator to retrieve information, add, modify and delete information at will with all the dependencies taken care of. Another facility provided is that of updating train information about any train from the Internet [1]. Since this system is off line, no security like passwords etc. has been provided. However, the proxy password is required while updating train information (assuming the existence of a proxy server).

3.1 Options

Various options are provided in the system for manipulating the database. Figure 3 shows the basic menus and Graphical User Interface of the system.

3.1.1 Add

The administrator can add stations, trains, and cities to the database, provided no conflicts with the previously existing database occur (Figure 4).

Figure 4: Addition of Trains

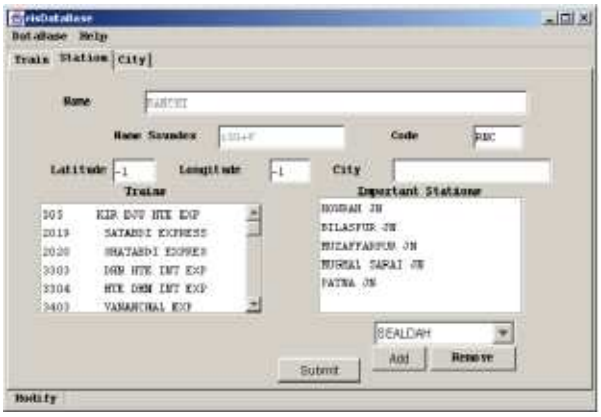

Figure 5: Modification of Stations

Figure 6: Search for Cities

3.1.2 Modify

The administrator can modify existing stations (Figure 5), cities and trains in the database. The required city/station/train can be retrieved by entering in the city name/station name/train number. After making the required changes, the information can be saved.

3.1.3 Delete

The administrator can delete existing stations, cities and trains from the database by entering the station name, city name or train number of the station, city or train to be removed.

A city cannot be deleted if it still has stations and a station cannot be deleted if there are still trains running through it.

If a train is deleted, then it is removed from the train list of all the stations, which were on its route. If a station is deleted, then it is removed from the station list of the city to which it belongs and from the important station list of all the existing stations.

3.1.4 Search

The administrator can search in the database for any train, station or city (Figure 6).

3.1.5 Commit Changes

After making all the changes, the administrator can commit the changes made to the database files. The arrays stored in memory are written into the files. Now, the changes made are irrevocable and cannot be reversed. If the changes are not committed into the database, they will be lost once the program is closed.

3.1.6 Undo Changes

The administrator can undo the changes made to the database that have not already been committed. The arrays are refreshed from the database files resulting in a loss of the changes that were not committed to the files.

3.2 Automated Database updating

The database needs to be updated every time the schedule of a train is changed or a new train is introduced. In this case, the administrator has to keep adding/modifying the database on a regular basis to keep it up to date. To reduce his effort, a tool has been designed which accesses the Indian Railway database itself through its internet portal [1] and updates the schedule of the trains asked for.

The advantage of this system is that there is no need for human intervention anywhere. Even the entering of train numbers can be automated, relieving the administrator of any responsibility. At present this tool is separate from the main

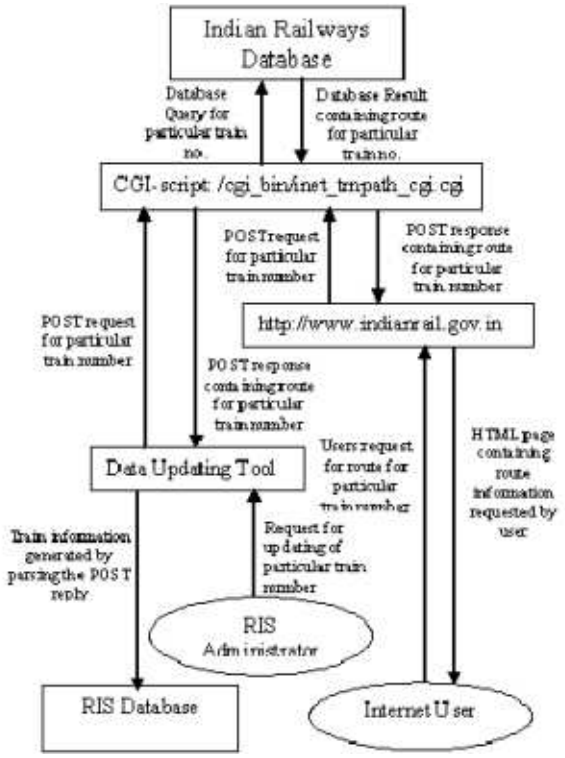

Figure 7: Basic structure and functioning of the Database updating tool

database management system but it can be integrated very easily with it, as an option for updating the database.

3.2.1 Implementation Details

The basic structure of the updating system and its functioning is shown in Figure 7.

3.3 Calculation of important stations

As already mentioned, for two stations, if no common train is discovered, then the important stations of both are checked to see if there is any train connecting them to the other station. This method does not ensure that an indirect route will be found if one exists, as we are checking only the important stations and it is quite possible that the common station may not belong to this set of stations.

However, we may assign the set of important stations in such a way so as to make sure that there is a high possibility of finding a common station in it. Here the concern is to restrict the size of this set otherwise the whole exercise of using important stations would be pointless. In RIS, the set of important stations for each station is determined in the following manner.

A station is said to be connected to another station if the other station can be reached from it by direct/indirect routes. The measure of direct connectivity of a station is defined as the number of stations that can be reached from it by the trains that run through it. distance(x, y) is the geographical distance between the two stations x and y. The following algorithm describes the procedure of obtaining the set of important stations for a station s.

1. The set of directly connected stations of s, CS is calculated.
2. For each station s' ϵ CS, first the set of directly connected stations CS' is calculated.
3. If n(CS'-CS)>t and distance(s', s)>r then add s' to FS. Go to 2 if stations left to be examined in CS
4. Add an element a ϵ FS to IS
5. For each f ϵ FS, calculate set of directly connected stations CF
6. For each i ϵ IS, calculate the set of directly connected stations CI.
7. If n(CF-CI) > t' then add f to CI. Go to 6 if stations left to be examined in CI.
8. Go to 5 if stations left to be examined in f.
9. IS is the set of important stations for the station s.

According to this algorithm, only those stations are considered for the set of important stations which increase the overall connectivity of the station s by a number greater than a certain threshold t. This means that only those stations are considered, which are connected to a certain no. of stations to which the station s is not connected and which

are within a certain geographical radius of s to ensure that the routes do not become extra long. In the case of RIS the threshold t and r have been determined empirically and are equal to the direct connectivity of s divided by a factor of 5, and 700 kms. respectively. Also before these stations are added to the set of important stations, it is made sure that their connected stations do not have a substantial overlap with the important stations already there. Here t is the threshold considered and in the case of iRIs, this is determined empirically to be number of extra connections provided by the important station being considered, divided by a factor of three.

Hence we try to restrict the set of important stations in two ways. Firstly, we only consider those stations which provide extra connections above a certain threshold. Secondly, we try to omit those stations, which provide extra connections to almost same set of stations.

This method provides an excellent set of important stations, which provide a route to almost the whole set of stations. One drawback may be that we do not look at the quality of the connections provided by these important stations so it is quite possible that we overlook quality of the connections for the quantity. However, in the case of indirect routes, quality has already been compromised by having a changeover, so this can be overlooked in favor of the fact that this method greatly restricts our search space while finding common stations for indirect routes, at the same time giving a high possibility of find a route if there exists one.

4 Querying System

The Querying System is hosted on-line and provides a number of facilities for all users. This querying system has been implemented using JAVA Servlets. The system provides facilities for printing out the routes of trains as given in the Indian Railways Time Table, for searching optimal routes given the source, destination and some user preferences, and for calculating the fare between two stations.

Figure 8: Online Querying system

4.1 Timetable facility

The querying system takes in either the train name or train no and prints out the complete time table of the particular train.

4.2 Route Search Facility

The Querying system takes in the source, destination, and a few other user preferences as well as the weightages given by the user for the criteria by which to evaluate the optimality of a route, to generate a list of the best routes as well as the cost of each.

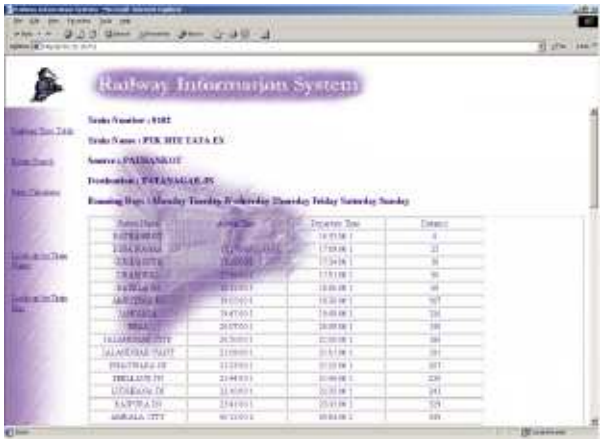

Figure 9: Time Table Generated by Querying System

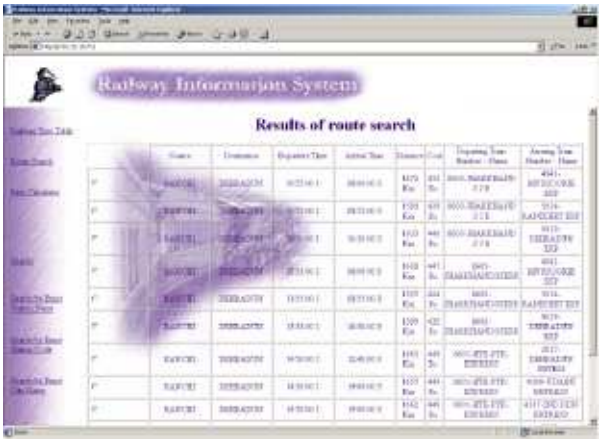

Figure 10: Routes Generated

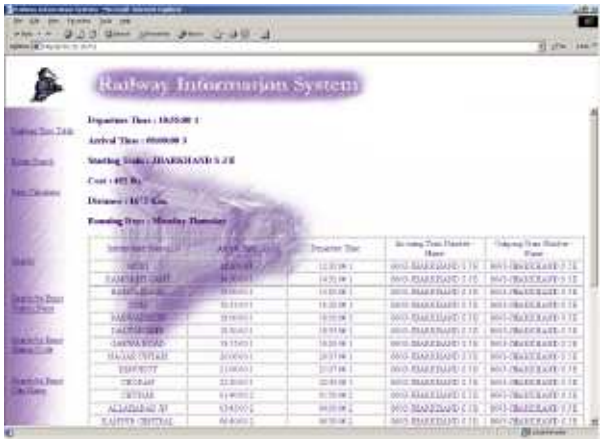

Figure 11: Route Detail as given by Querying System

4.2.1 Approach for Route Search

The approach followed for route search is to first find all the direct trains if any passing through the two stations. Then we have to search for indirect routes. Now for this,the simplest method would be to represent the connectivity of the trains in the form of a weighted directed graph with the stations as nodes and the trains between two stations as links between them. Then to find a route between two stations a search for a shortest path between the two corresponding nodes in the graph would be performed. However, this turns out to be computationally complex due to the intricacies of the graph and also a lot of storage would be wasted. Since this is supposed to be an enquiry system, the major requirement is that the queries should be quickly answered although the route may not be optimal. Therefore another strategy is used to handle indirect routes, as already mentioned, which may not give all the routes possible but gives most of the routes within a decent interval of time.

For every station a list of important stations that are well connected to the railway network and also are connected to the station itself is stored, as determined earlier. Thus while searching for an indirect route between two stations; the important stations of both are looked at in case they are directly connected to the other station or its important stations. At the next level, the important stations of the important stations are looked at and so on. This is done till a certain threshold after which it is declared that no practical route exists between the two stations. This process is done at more than 1 level as it is quite possible that a route may have 2 or more connecting stations as well. However this is not done beyond a certain threshold because it would be impractical to have too many connecting stations in the middle, as that would require hopping too many trains and also that would really increase the search time, whereas this whole exercise is being carried out to reduce exactly that. In RIS this is done at only one level as it was found experimentally that good results are obtained even if this process is carried out at only the first level.

4.2.2 User Preferences

For the route search, a number of user preferences are taken into account. They are:

1. Via Stations: The route should pass through these stations

2. Stopovers: The route should have change of trains at these stations

3. Minimum Time at Stop: There should be at least this interval between the two trains at a stopover

4. Maximum Time at Stop: There should be at maximum this interval between the arrival and departure at a stopover

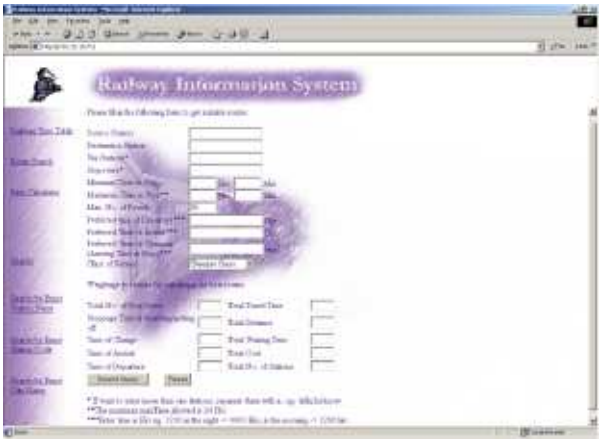

Figure 12: User Preferences for Route Search

5. Maximum Number of results: At maximum this number of best possible routes should be displayed after the route search
6. preferred time of departure: The time of day at which the user prfers to leave from the source
7. preferred time of arrival: the time of day at which the user prefers to arrive at the destination
8. preferred time of change: the time of day at which the user prefers to arrive at a stopover where he is supposed to disembark and wait for the connecting train
9. weightages to criteria for calculating best routes: the users preferences regarding the optimality of routes are taken in the form of weightages assigned to various criteria which are used to decide the optimality of a route

The routes found strictly satisfy the first 5 preferences. However in the case of preferences 6-8, the times may deviate slightly from the given time and one factor while deciding optimality of routes will be this deviation from the users preferences.

4.2.3 Quality Metric

The quality metric based on which the routes are ranked is determined from the various optimality criteria and the weightages assigned to them by the user. The optimality criteria being used currently in the RIS are:

1. Total Travel Time: this should obviously be minimum in the optimal case
2. Total Distance Travelled: this should be minimum in the optimal case
3. Total Waiting Time (at the stopovers): this should be minimal in the optimal case, as this signifies the extra time spent at the stopover waiting for the connecting train. However there si a minimum limit to this time as specified by the user to ensure ease in changing trains.
4. Total Cost: this should be minimum
5. Total No. of Stopovers: this should be minimum in the optimal case as a high number of stopovers signifies a large number of changes and hence wasted time and effort
6. Total No. of Stations en route: this should be minimal in the optimal case as the larger the number of stations en route, the higher the probability of delay of the train at any one of them and also larger the amount of time spent by the train at various stations
7. Stoppage Time at boarding/getting off: this should be maximum to ensure the safety and ease of the passenger while embarking or disembarking
8. deviation from preferred time of change: this should be minimum to satisfy the users preferences
9. deviation from preferred time of arrival: this should be minimum for the same reason as above
10. deviation from preferred time of departure: this should be minimum for the same reason as above

New optimality criteria can be added in the system very easily due to the modularity and object oriented quality of the code.

To obtain the quality of a particular route, the following procedure is followed in RIS:

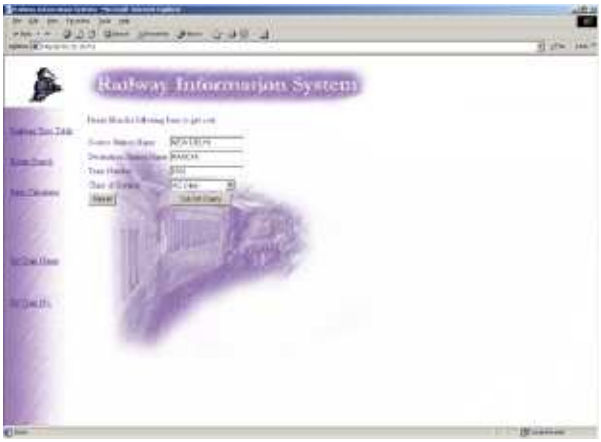

Figure 13: Cost Calculation Facility

Figure 14: Various options to ease user effort

1. For each optimality criterion, obtain the average as well as maximum values over the entire set of routes to be evaluated. In the case of the criteria of stoppage time and the deviations from user preferences, consider the maximum values over a route.

2. Calculate the value of deviation of the route's value for the criterion, divide it by the deviation of maximum from the average and multiply it by the users preferred weightage for that criterion.

3. add the above obtained value to the quality value in case the criterion is that of stoppage time else subtract it from the quality value.

The higher the quality value, the better is the quality of the route. This is a linear quality metric. However, other quality metrics can as well be used here.

4.3 Cost Calculation Facility

The Querying system also provides the facility of finding out the cost of traveling by a particular train in a particular class from a specified source to a destination. It takes the user input of train number/name, source station name, destination station name, and class of travel. The cost is calculated by using the 1999 Indian Railway distance - fare chart (available from http://gidduk.webhostme.com/farelist.html).

4.4 User Friendly Behavior

The RIS is designed to very user friendly and easy to use. A number of features have been incorporated into it to increase the ease with which the users can operate it. The web interface is aesthetically pleasing and easy to operate. The user is given the option to search by train numbers or names, and by stations names, codes, or city names. In the case of cities, all the stations within a particular city are considered. Also an option has been provided to search by names which are not complete or which have been misspelled or mistyped.

4.5 Auto Correction of Names

If a user enters in a wrong/incomplete name, it should be corrected to the closest name or set of names in the database. There are three ways in which incorrect names can be entered.

- Incomplete names may be entered. For example, if Vadodara Jn. is the name of the station in the database and the user enters Vadodara, Vadodara Jn should be taken as input.

- Another possibility is that of typing errors. For example, the user may enter Bomby instead of Bombay.

- A third possibility is that the user may not know the actual spelling of a station/train/city name and may try and guess the spelling from

its pronunciation. For example, some ignorant user may refer to Delhi as Dilli (from the Hindi pronunciation).

4.5.1 Incomplete Names

This is taken care of in the RIS by matching the name entered by the user to the names in the database and listing all such names for which it is an initial substring

4.5.2 Mistyped Names

This is taken care of in the RIS by using skeleton keys [4]. The skeleton keys are formed for all the names in the database and then sorted alphabetically. Now, for a misspelling, the skeleton key for that is generated and matched to the keys in the database, the words whose skeleton keys are closest to the skeleton key of the misspelling are retrieved and the user can choose the name she actually meant.

The skeleton key is constructed by concatenating the following features of the string (name, misspelled name): the first letter, the remaining unique consonants in the order of occurrence, and the unique vowels in the order of occurrence. For example, the skeleton key for the name RANCHI is RNCHAI and the skeleton key for the name VISHAKHAPATNAM is VSHKPTNMIA. The rationale for this key is that

1. The first letter keyed is likely to be correct
2. Consonants carry more information than vowels
3. The original consonant order is mostly preserved
4. The key is not altered by the doubling or undoubling of letters or most transpositions.

The skeleton key reflects the misspellings collected and intuitively it can be said that strings that "look" similar produce closely related keys. The major drawback of this key is its emphasis on the early consonants. The closer an incorrect consonant is to the start of a word, the greater is the lexicographic distance between the keys of the word and misspelling.

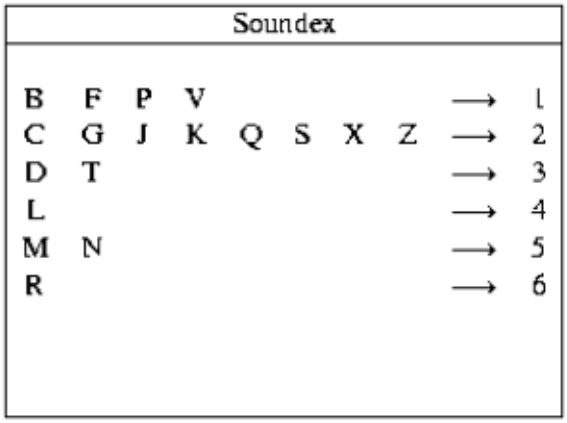

Figure 15: Soundex Code

4.5.3 Misspelled Names

In order to search for names in the database which had been misspelled, it was decided to use a phonetic code to capture the pronunciation of the word. A number of codes were considered as alternatives.

Soundex code: Soundex's phonetic property is restricted to the collecting of similar sounding consonants into different classes.

Soundex works as follows:

1. Remove all vowels, the consonants H, W, Y and all duplicate consecutive characters. The first letter is always left unaltered.
2. Create the Soundex code by concatenating the first letter with the following 3 letters replaced by their numeric code according to Figure 15.

PHONIX Code: PHONIX is far more complex than Soundex. While Soundex only removes vowels, some consonants and duplicate letters and carries out the numerical substitution, the work of PHONIX is more extensive:

1. Perform the phonetic substitution, i.e., replace certain letter groups by other letter groups.
2. Replace the first letter by V if it is a vowel or the consonant Y.
3. Strip the ending-sound from the word (roughly the part after the last vowel or Y).

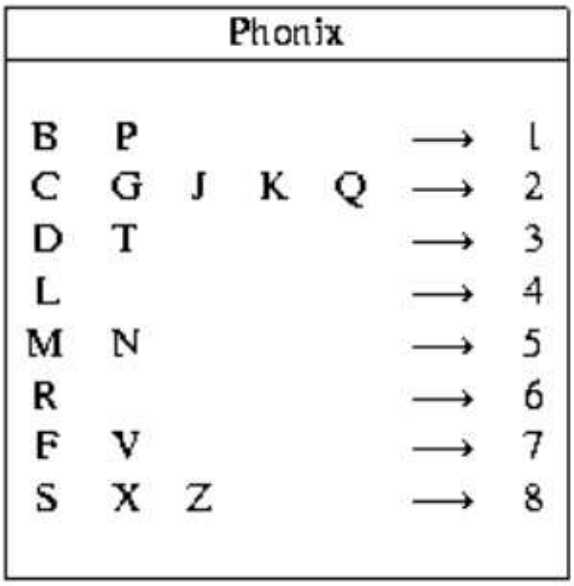

Figure 16: PHONIX Code

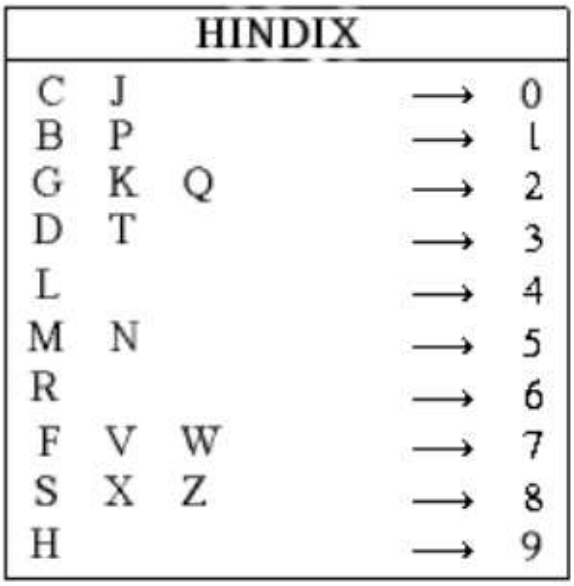

Figure 17: HINDIX Code

4. Remove all the vowels, the consonants H, W, Y and all duplicate consecutive characters.

5. Create the Phonix code of the word without its ending-sound by replacing every but the first remaining letter by its numerical values according to Figure 16. The maximum length of a Phonix code is restricted to 8 characters.

6. Create the PHONIX code of the ending-sound by replacing every letter by its numerical value. The maximum length of a PHONIX code for an ending-sound is restricted to 8 characters.

However, both the above codes are designed to be applied to English words and the pronunciation of many of the Hindi words (written in English) does not tally with their soundex/phonix codes. Thus there was a need to design a phonetic code for the station and train names to enable auto correction of misspelled names. Hence a code HINDIX similar to PHONIX which implemented rules pertaining to hindi words was designed.

HINDIX Code: This code follows the basic algorithm of the phonix code but the rules and letter manipulations have been changed with a few new rules added.

Thus the sequence of actions for obtaining hindix codes is:

1. Perform the phonetic substitution. Replace 'ph' by 'f'. If 'ee' occurs at the end of the string, replace by 'i'. If 'w' occurs at the end and is preceded by 'e', replace 'w' by 'u'. If 'w' is not succeeded by a vowel, replace by 'v'. If 'c' is not followed by 'h' replace 'c' by 'k'.

2. Replace the first letter by 'V' if it is a vowel or the consonant 'y'.

3. Strip the ending-sound from the word (roughly the part after the last vowel or 'y'). If the last vowel occurs at the last place in the string, strip after the second last vowel.

4. Remove all the vowels, the consonants 'h', 'w', 'y' and all duplicate consecutive characters, except for the first letter of the string.

5. Create the HINDIX code of the word without its ending-sound by replacing every but the first remaining letter by its numerical values according to Figure 17.

6. Create the HINDIX code of the ending-sound by replacing every letter by its numerical value. In the case of vowels and 'y' replace by 'V'.

The HINDIX code has been designed by looking at the general spelling tendencies of Indians while writing Hindi words in the Roman font. It has been designed empirically and gives good results for the RIS database. For example, the HINDIX code of 'Delhi' is d4+V and the HINDIX code for 'Dilli', as commonly misspelt by Indians is d4+V which matches. In many cases, the HINDIX code produces better results than either PHONIX or Soundex codes for Indian words. For example, the code for the misspelt name 'bhavanipur' is b751+6, the same as for the correct 'bhawanipur'. However the Soundex and PHONIX codes for both are different. Thus the HINDIX code is used in RIS to correct misspelt words.

5 Future Possibilities

The RIS project, although complete in itself, can be extended in a number of ways.

The route finding facility can be extended to airplanes as well, making it a complete travel information system, finding routes covering both ir and train travel. This is a very simple task due to the object oriented nature of the code and the modularity of design, the added advantages of using JAVA. The only need is to create a class for air flights containing their complete routes and timings. The RIS system therefore is very easily extendible.

RIS can be speech enabled by adding a speech recognition module and a parser module which parses the text converted from the speech and obtains the query. Such modules are easily available commercially and can be integrated with minimal effort.

6 Conclusion

Although the route search problem is not a tough one by itself, it becomes complex when combined with the vastness of the Indian Railways Network, the necessity for the on-line system to answer queries in real-time and also the need to take into account user preferences while finding routes to yield greater user benefits. RIS is a system which tries to incorporate all of the above while at the same time being user friendly and simple to use. The hindix codes developed for use in RIS go a long way in achieving this goal of user friendly behavior and can be used for retrieval of Indian words other than station names in other systems. Although at present RIS supports only rail routes, it is quite easy to generalize this system to include all types of routes and hence convert it into a comprehensive travel planning system.

7 References

1. http://www.indianrail.gov.in.
2. Leeladhar Bokade, "Computerized Railway Enquiry System", B. Tech. Project Report, 1992.
3. U. Pfeifer, T. Poersch, N. Fuhr, "Retrieval Effectiveness of Proper Name Search Methods", Information Processing and Management, 1996.
4. U. Pfeifer, T. Poersch, N. Fuhr, "Searching Proper Names in Databases", HIMS, 1996.
5. T. Gadd, "PHONIX: The Algorithm", Program 22(3), 1990.
6. J. Pollock, A. Zamora, "Automatic Spelling Correction in Scientific and Scholarly Text", CACM(27), 1984.