IMPLEMENTATION OF PRESUMPTIONS USING DFS_h IN A CASE STUDY

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Gaurav Rathi

(801132031)

Under the supervision of:

Dr. Shivani Goel

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

June 2013

CERTIFICATE

This is certified that the thesis entitled **"Implementation of presumptions using DFS_h in a case study"** is an authentic record of my own work carried out as requirements for the award of the degree of M.E. (Computer science & Engineering) at Thapar University, Patiala , under the guidance of Dr. Shivani Goel (Assistant Professor, CSED) during July 2012 to June 2013.

Date:-

Gaurav Rathi

Roll No. 801132031

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Dr. Shivani Goel
Assistant Professor
Department of Computer Science and Engineering

Counter signed by:

Dr. Maninder Singh
Head of the Department
Department of Computer Science and Engineering

Dr. S. K. Mohapatra
Dean of Academic Affairs
Thapar University, Patiala

Abstract

The research work in this thesis provides an idea to make presumptions and take a case study of railway reservation system to apply idea of presumptions using hMetis and DFS_h algorithm. These are used because these can very efficiently divide and traverse the graph. First divide the graph with the help of hMetis and then traverse the graph with the help of DFS_h Algorithm .With the help of few modifications, we can restrict the traversing for adjacent node each adjacent of node traverse up to two nodes only and then return to original node. With the help of this process, also we can reduce the database size.

After dividing and traversing the graph, the root or path from source to destination node can be easily found .Then, how many trains are available on particular path are available can be found and the availability of the seats can also be checked. If there is waiting in the availability of seats, then we analyse all five scenarios which are given in thesis and with the help of ***presumptions*** make reservation system very efficient.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

No volume of words is enough to express my gratitude towards my guide **Dr. Shivani Goel**, Assistant Professor, Department of Computer Science & Engineering, Thapar University, Patiala, who has been very concerned and has aided for all the materials essentials for the preparation of this thesis report. She helped me to explore this vast topic in an organized manner and provided me all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. Maninder Singh**, Head of Computer Science & Engineering Department and **Mr. Karun Verma**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there at the need of hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Gaurav Rathi

(801132031)

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives a basic information about the Railway Reservation System, DFS, DFS-H, DFS Applications, SQL, Dynamic Query and Dot Net.

1.1 Online Reservation system

The Indian Railways (IR) carries about 5.5 lakh passengers in reserved accommodation every day. The computerised Passenger Reservation System (PRS) facilitates booking and cancelling of tickets from any of the 4000 terminals (i.e. PRS booking windows) all over the country. These tickets can be booked or cancelled for journeys commencing in any part of India and ending in any other part, with travel times as long as 72 hours and distances up to several thousand kilometres[3] .

The pilot project of PRS was launched on 15 November 1985, over Northern Railway with the setting up of the Integrated Multiple Train Passenger Reservation System (IMPRESS), an online transaction processing system developed by the Indian Railways in association with Computer Maintenance Corporation (CMC) Ltd., at New Delhi. The objective was to provide reserved accommodation on any train from any counteract, preparation of train charts and accounting of the money collected. This application was subsequently implement in 1987, at Mumbai, Chennai, Kolkata and Secunderabad. With the addition of new areas and many redefinitions, the IMPRESS system fell squat of growing hope of the travelling public. Hence a fresh application software, i.e., Country Wide Network for Computerised Enhanced Reservation and Ticketing (CONCERT) was developed by the Centre for Railway Information Systems (CRIS), New Delhi primarily using 'C' and also using 'FORTRAN'. The application was first implemented at the Secunderabad PRS site in September 1994 and subsequently at the other four PRS locations. Currently, the PRS servers are maintain at the five areas in Delhi, Mumbai, Kolkata, Chennai and Secunderabad and work in a circulated database process surroundings [3].

Communication of all the terminals with their server was established using Railway/Department of Telecommunication (DOT) channel line, fibre optic wire/microwave channel, switches, modem, multiplexers etc. The inter-networking of five PRS nodes was completed in April 1999. Interconnectivity is recognized between the five PRS centres over 2 mbps leased Bharat Sanchar Nigam Limited (BSNL) lines. The system has the competence of

issuing engaged tickets from anywhere to anywhere, in any train, date or group linking any pair of stations from any booking workstation of the PRS.

This thesis aims at development of an Online Railway Reservation Utility which facilitates the Railway customers to manage their reservations online, and the Railway administrators to modify the backend databases in a User-Friendly manner.

The customers are necessary to register on the server for getting right to use to the database and query result reclamation. Upon registration, each client has an account which is fundamentally the 'view level' for the customer. The account contains comprehensive information of the user entered during registration and permits the customer to get entrée to his long-ago reservations, ask about travel fare and accessibility of seats, make anew reservations, modernize his account details, etc.

The Railway Administrator is the second party in the connections. The administrator is necessary to login using a master password, once suitable as an administrator, one has right to use and right of adaption to all the information stored in the database at the server. This include the account information of the clients, attributes and figures of stations, explanation of the train stoppages and physical explanation of coaches, all the reservation that have been made, etc. The railway administrator has the right to modify any information stored at the server record.

1.2 DFS Graph Algorithm

We begin with some standard graph-theoretic definitions. A graph G is a set of vertices V and a set of edges E, written G = (V, E). We let $|V| = n$ and $|E| = m$. A ***directed edge*** of G is an element of V x V; an undirected edge of G is a subset of V of cardinality two. G is directed (undirected) if E consists entirely of directed (undirected) edges. The indegree of a vertex $\bar{\alpha}$ in a directed graph G is the number of edges of the form $(v, \bar{\alpha})$; the outdegree of a vertex $\bar{v}$ is the number of edges of the form $(\bar{v}, \alpha)$. A directed (rooted) tree T is a directed graph in which every node except one has indegree 1; the remaining node has indegree 0 and is called the root of T. A graph G' = (V', E') is a subgraph of a graph G = (V, E) if V'~V and E'~ E. A directed tree T is a spanning tree of a graph G if T is a subgraph of G and T contains all the vertices of G. A spanning tree T is a depth-first search tree of G iff, for all non-tree edges {v, w} (or (v, w) if G is directed), v and w lie on the same branch of T.

The depth-first search problem is: given a graph G and a vertex ***T*** in G, make a depth-first Search tree T of G rooted at T**.** T is called a search tree because the standard way to construct it is to search G in a depth-first method, as follows. We begin our search at r; T is initially empty, and all the vertices of G are marked unvisited. We mark the existing vertex v as visit and sequentially examine all unexplored edges leaving v (in any order we wish). If there is an unvisited vertex w adjoining to v, we make w the existing vertex and reiterate. If there is no such vertex, we backtrack to the last vertex we visited that has at least one such vertex and repeat. We halt when we have explored every edge (and visited every vertex) in G. This algorithm has a simple recursive description

procedure DFS (v)
begin
mark v as visited;
while there is an unmarked vertex w adjacent to v **do**
add (v, w) to T;
DFS (w)
end {while }
end { DFS }

The running time of DFS is easy to analyse; it is just O(n + m), since we visit every vertex and explore every edge exactly one time. Given what we are trying to achieve, then, this algorithm is optimal to within a steady factor. For this reason, we will regularly equivocate between the terms "parallel DFS algorithms " and "sub-linear DFS algorithms", as just similar DFS algorithms can run in sub-linear time. It should be clear that the decision we make at each vertex (i.e., which edge to explore next) has a drastic effect on the decisions we make after that. This is why so many people conjecture that DFS was naturally sequential when the question first arose. The reason why they were mistaken is that, loosely speaking, their understanding of DFS as a search problem was not declarative enough.

1.2.1 DFS Applications

- **Detecting Cycle**

For this application first we should know about the types of edges in DFS algorithm,

DFS algorithm edges classify in four types.

- **Tree Edges** edge (*u; v*) is a tree edge if *v* was first discovered by exploring edge (*u; v*). The tree edges form a spanning forest of *G* (no cycles).
- **Back Edges** an edge (*u; v*) is a back edge if it connects vertex *u* to an ancestor *v* in a DFS tree.
- **Forward edges** an edge (*u; v*) is a forward edge if it connects a vertex *u* to a descendant *v* in a DFS tree.
- **Cross edges** all other edges.

We already know that all edges of *G* will be classified as either tree edges or back edges. In addition, if *G* is acyclic, then only tree edges will arise. This is because if *G* is acyclic, then it is a forest (disjointed trees). Therefore, we have the following product: An undirected graph is acyclic if DFS yields no back edges. Detecting cycle can be done in O(n + m) time by running DFS. We use a Stack s to keep track of the path between the start vertex and current vertex, as soon as a back edge encountered, we return the cycle as the portion of the stack from the top to vertex w.

- **Topological Sorting**

A **topological sort** (sometimes abbreviated **topsort** or **toposort**) or **topological ordering** of a directed graph is a linear ordering of its vertices such that for every directed edge *uv* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering. For example, the vertices of the graph may signify tasks to be performed, and the edges may represent constraint that one task must be performed before another. In this purpose, a topological ordering is just a valid chain for the tasks. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a direct acyclic graph (DAG). Any DAG has at least one topological ordering, and algorithms are identified for constructing a topological ordering of any DAG in linear time

- **Strongly Connected Component**

Decomposing a directed graph into its powerfully connected components is a standard application of depth-first search. The difficulty of searching connected components is at the heart of many graph applications. Generally speaking, the connected components of the graph correspond to altered classes of objects. The first linear-time algorithm for strongly connected components is due to Tarjan (1972).

Given digraph or directed graph G = (V, E), a strongly connected component (SCC) of G is a maximal set of vertices C subset of V, such that for all u, v in C, both $u \rightarrow v$ and $v \rightarrow u$, that is, both u and v are reachable from each other. In other words, two vertices of directed graph are in the similar component if and only if they are reachable from each other.

- **Cut Vertex or Articulation Point**

In an undirected graph, a cut vertex is a vertex and if we removing it then the graph splits into two disconnected components. As an example, consider the following figure 1.1

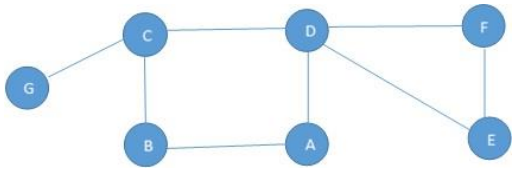

Figure1.1 Graph for showing cut vertex

Removal of "D" vertex divides the graph in to two connected components {E , F}and {A, B, C, G }. Similarly, removal C vertex divides the graph into {G} and {A, B, C, D , E, F}. For this graph A and C are the cut vertices.

1.2.2 DFS-H Algorithm

VLSI, web applications, network, and data mining are represented by using a massive graph. These applications need graph search: checking all nodes in a graph with the goal of finding a specific node with a specified property. Different techniques are adopted for graph finding such as Breadth first Search (BFS) and Depth First Search (DFS). It is easy to apply the depth first search on the small graph since it fits in the memory. However, large graphs, DFS poses many challenges related with its storage in memory and the I/O operations.

With the help of enhancing DFS computation (DFS-H) on a large dense graph based on the RAM model. The main idea is to divide the large graph into the set of sub-graphs that fit in

the memory by using hMetis. The separation techniques reduce the number of crossing edges between the sub-graphs, which reduce the number of I/O operations.

1.2.3 hMetis

The hMetis; software package was developed for partitioning the VLSI circuits by G. Karypis and V. Kumar (G. Karypis, 1999; G. Karypis, 1997; G. Karypis, 1996). It permits the user to specify a parameter K, which represent the required number of fragments. It partitions the graph into K fragments such that each fragment contains approximately the same number of vertices, while minimizing the number of edges that cross the fragments. Figure 1.2 illustrates a graph that has been partitioned. There are other methods to partition the graph that depends on spatial proximity.

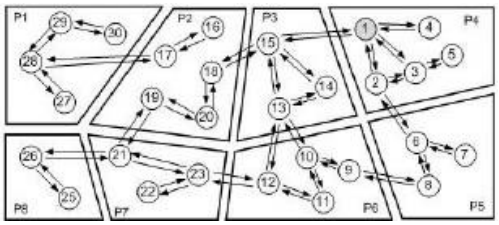

Fig. 1.2 Divide graph in eight parts by hMetis

1.3 Network Scheduling

1.3.1 Genetic Algorithm

Genetic algorithms (GAs) have been successfully applied to combinatorial problems and are able to handle huge search spaces as those arising in real life scheduling problems. GAs perform a multidirectional stochastic search on the complete search space that is intensified in the most promising areas.

1.3.2 Fuzzy Logic

Using fuzzy logic, two-stage fuzzy optimization model is obtained to a robust rescheduling plan under irregular traffic conditions, and a scenario-based depiction is adapted to describe fuzzy recovery time durations on a double-track railway line. It minimizes the expected total delay time in the rescheduled train schedule with respect to the original timetable.

1.3.3 Ant Colony

One of the most surprising behavioural patterns exhibited by ants is the ability of certain ant species to find what computer scientists call direct paths. Biologists have given away experimentally that this is achievable by exploiting communication based only on

pheromones, a stinking chemical substance that ants may deposit and smell. It is this behavioural pattern that inspired computer scientists to develop algorithms for the solution of optimization problems [2].

1.3.4 Constraint Logic Programming(CLP)

If a problem is very large (more than a few hundred variables), CLP is a good framework to use because it is relatively easy to implement problem specific information. Even a simple model of the cane train system has several thousand variables. Also, if it is required that the program be frequently updated, or that the user be able to interactively affect the outcome of the search, CLP is an appropriate methodology.

1.4 Dynamic Query

A *Dynamic Query* is a pre-configured search against either an LDAP directory or a DBMS that returns, at a least, a sequence of e-mail address, and may also return extra information, such as the recipient's full name or phone number. Queries can be used either for admittance manage purposes (for instance, to permit all members of the CORP_HR Windows Security Group authorization to post to the HR-NEWS list), or to give additional recipients for a relocation to a mailing list.

All active queries are define forward of time by the LISTSERV administrator. List owner cannot create their own queries, but they can make use of pre-defined queries and provide parameters to these queries (see below for more information on security limits). The LISTSERV administrator determines how these parameters are used. For example, the administrator could define a query that matches all members of a particular Windows Security Group, specified as a parameter. List owners could then give the group name when they demote to the query. Queries whose description contains no parameters cannot be changed by list owners.

Dynamic queries are executed when, and only when, LISTSERV needs entrée to the query results to complete a work. An undeveloped query consume no resources. Note that LISTSERV does carry out queries when verifying the syntax of list header alters. LISTSERV discards any invalid queries and issues a caution when the query is valid, but returns no data, as this usually means that a parameter was spelled wrongly. Queries can be inserted in most list header keywords used for access control or to specify e-mail addresses. Example:

- Owner= J.Smith@example.com (Joe Smith)
- Owner= Query(DEPT,HR) (HR Department)
- Review= Private, Query(DEPT,HR),*@example.com

1.5 SQL queries

SQL is a database computer language planned for the retrieval and administration of data in relational database. SQL stands for Structured Query Language.

SQL is structured Query Language which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server uses SQL as standard database language.
Also they are using different dialects, Such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

1.5.1 Why Sql ?

- Allow users to access data in relational database management systems.
- Allow users to explain the data.
- Allow users to describe the data in database and manipulate that data.
- Allow to set in within other languages using SQL modules, libraries & pre-compilers.
- Allow users to make and drop databases and tables.

1.5.2 SQL Process

When you are executing an SQL command for any RDBMS, the system determine the best way to fetch your request and SQL engine figures out how to read the job.

There are different parts included in the process. These parts are Query Dispatcher, Optimization engines, Classic Query Engine and SQL query engine etc. Classic query engine handle all non-SQL queries but SQL query engine won't handle logical files.

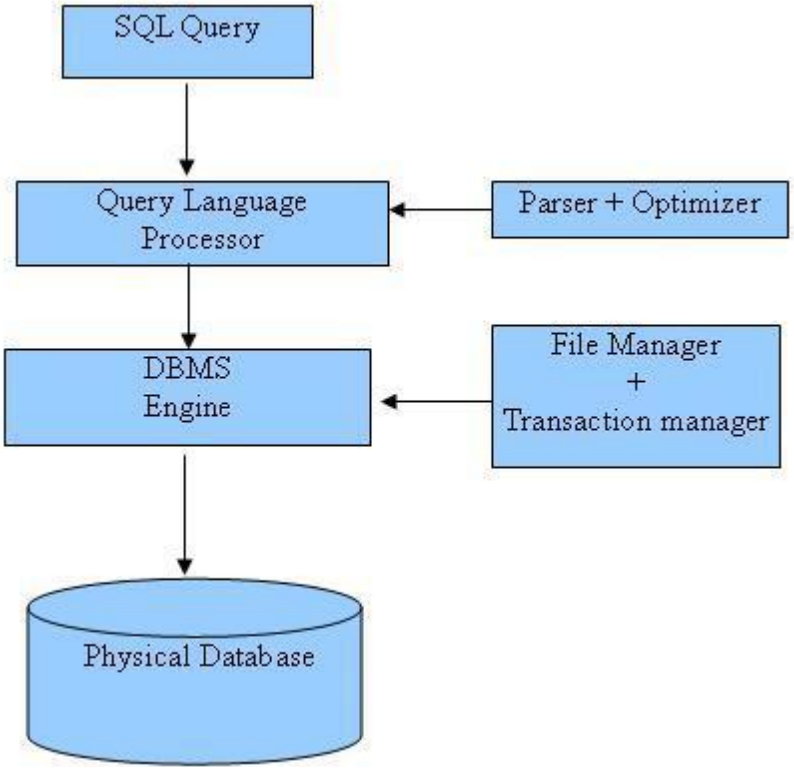

Fig. 1.3 Process of SQL

1.6 SQL Express

Microsoft SQL Express is the freely-downloadable and distributable version of Microsoft's SQL Express relational database management system. It offer a database solution specifically targeted for embedded and smaller-scale application. ? Unlike its antecedent, MSDE, there is no parallel workload chief which "restricts performance if the database engine receive more labour than is classic of a small number of users" it does, however, have a number of *technical limits which make it unwanted for large- scale deployments, including:*

- Maximum database size of 4 GB per database (compared to 2 GB in the former MSDE). The 4 GB limit is per database (log file excluded) and can be extended in some scenarios through the use of multiple interconnected databases.
- Hardware utilization limits:
    - Single physical CPU, multiple cores
    - 1 GB of RAM (runs on any size RAM system, but uses only 1 GB)

- Absence of SQL Server Agent Service

Although its predecessor, MSDE, was virtually devoid of basic GUI management tools, SQL server Express includes several GUI tools for database management. Among these tools are :

- SQL Server Management Studio Express
- SQL Server Configuration Manager
- SQL Server Surface Area Configuration tool
- SQL Server Business Intelligence Development Studio

A relatively late addition to the SQL Server Express product line is a reduced functionality version of SQL Server Reporting Services, but features such as Analysis Services and Notification Services are only available in the "standard" edition and higher editions.

1.7 ASP .Net

The .Net platform is a good development framework with a new programming interface to windows services and API's integrating a number of technologies that emerge from Microsoft during late 1990's .Net framework is the latest and the most powerful technology with the help of which windows services can be created very efficiently because .Net manages error reporting and system restore very nicely and easily as compared other languages. ASP.net is a web applications framework developed and marketed by Microsoft that programmers can use to build active web sites, web application and web services. It was first release in January 2002 with version 1.0 of the .Net framework, and is the descendant to Microsoft's Active server page (ASP) technology. ASP.Net is build on the common language runtime (CLR), allowing programmers to write ASP.Net code using any supported .Net language.

Thesis Outline

In chapter 1, introduction to problem is discussed. Chapter 2 contains the literature survey. Problem taken in this thesis is stated in chapter 3. Many scenarios taken up in the research problem are discussed in chapter 4. Chapter 5 contains experimental results and chapter 6 concludes the work done in thesis and provides the future scope.

Chapter 2

Literature Review

2.1 The Unreserved Ticketing System of Indian Railways

17 million people travel by Indian Railways in a day. And approximately 1 million travel on reserved tickets, which provide them a chair or berth. And the 16 million buy unreserved tickets at more than 6100 stations in the country, which give them the right to board and travel by coaches or trains nominated for unreserved passengers but do not guarantee a seat or sleeping berth. Around 9 million of the 16 million unreserved passengers are daily passengers commuting mainly for business, to work or to study, either within the same city or to nearby cities and towns. Earlier all ticketing was through manually issued pre-printed card tickets or paper tickets. As the numbers of trains run and the passengers carried increased phenomenally over the years, the logistics, of indenting, procuring, distributing, stocking, issuing and accounting for such a large number of tickets, posed a formidable challenge to the Indian Railways. In 1985, computerised ticketing for reserved tickets was introduced in Delhi. This system, known as the Passenger Reservation System (PRS), proved extremely successful, both from the administration and the public perspectives, and over the next decade was extended to cover most of the stations where trains with reserved accommodation stopped. PRS has kept pace with emerging technologies and today you can buy a reserved ticket through the internet and through mobile phones. Unreserved ticketing, which accounted for the bulk of the tickets issued but catered predominantly to the ordinary second class passenger – who paid the lowest fares and was perceived as a 'subsidised' customer, was by and large technologically neglected till 2002 when a computerised Unreserved Ticketing System for Indian Railways was conceived, sanctioned and introduced in Delhi as a pilot project. The system was designed, developed and implemented in-house by the Centre for Railway Information Systems – the IT arm of the Indian Railways.

2.1.1 Background

Most people, when they think of railway tickets, think of a reserved ticket. A reserved ticket is associated with a passenger journey by a particular train in an assigned coach on an assigned seat or berth. In some cases when the bookings are full at the time of purchase of a ticket, the passenger has an option of buying a unconfirmed ticket – either one which is a

RAC (reservation against cancellation) or a wait listed one. A RAC ticket assures the passenger of a nominated seat on the train with the hope of a berth if there is a cancellation or if a passenger with a confirmed ticket does not show up. The wait listed passenger either gets confirmed against cancellations or has to change travel plans if it remains unconfirmed. Some of the more adventurous ones, board the train in the hope of obtaining a place en route.

The story of reserved tickets and the romance associated with it is well documented. But that is only six percent of the total passenger traffic. The other 94% of the passengers buy tickets which are neither for a specific train nor for an assigned seat.

The unreserved passengers account for 16 of the 17 million passengers daily on the Indian Railways. They get tickets for a *destination* and are free to board either any train which has all unreserved accommodation or any coach nominated for unreserved passengers on trains that have both reserved and unreserved accommodation. The only limitation is that the journey should be completed within a specified time window. A seat is not guaranteed. The validity of the ticket is usually up to midnight of the day for which the ticket is issued except for the commuter services of Mumbai, Kolkata and Chennai where journeys must commence within 1-2 hours of ticket purchase. Unreserved passengers mainly travel in ordinary second class coaches, which have the lowest fares, but contribute over Rs. 8700 crores annually to railway revenues, which is around 53% of Indian Railways income from passenger traffic. Unlike the reserved passengers, who generally book their seats in advance, the unreserved passengers normally buy their tickets just before boarding their trains. So demand for tickets peaks just before departure of popular trains at all stations and during office opening and closing hours in big cities. In such a situation if a passenger cannot be sold a ticket quickly and conveniently, the railway is likely to lose a business opportunity because the passenger will either opt for an alternate mode of transport or, worse, travel without a ticket. Till the 1990s, tickets were manually issued. The ticket was normally a Printed Card Ticket (PCT) which had all the details like origin, destination, class, fare etc. pre-printed.

The booking clerk had to stamp the date and time of issue using a dating machine. The system worked well for decades but when there was a dramatic increase in traffic various problems began to surface:

1. Every station had to stock tickets for every possible destination, combined with every class of travel, every type of train service, route and concession, which could be accessed from that station, regardless of the frequency of demand. Since indenting, procuring and distributing

tickets for such a large number of stations involved elaborate, expensive and time consuming procedures, stations were required to stock 10-20 months requirement of tickets, depending on their consumption patterns. This meant that busy stations had to stock millions of tickets and zealously protect them from termites, rodents, leaky roofs etc. because each PCT had a monetary value. They would also have to guard it from human miscreants.

2. As the number of passengers increased it became increasingly difficult to ensure that every station always had its full requirement of tickets on time. This lead to a greater dependence on paper tickets, where the Booking Clerk filled in the ticket details in a pre-printed format. Paper tickets were prone to frauds by the ticket issuers and caused complaints of overcharging.

3. Since PCTs had the fare pre-printed on them, whenever a fare change was to be implemented the station staff at each station had to manually correct the fare on each ticket in stock. This not only led to wastage of manpower and errors but also increased the opportunities for malpractice leading to complaints. It was also difficult to ensure that such a large number of geographically disparate stations implemented the new fares on schedule, resulting either in loss of railway revenue or overcharging of the passenger–depending on whether the fares were going up or down.

4. PCTs were also prone to misuse in the form of fake tickets which couldn't easily be detected and re-used if the issue date impression was faint. At important junction stations the number of types of tickets to be sold became so high that one counter could not possibly handle tickets for all destinations. This lead to the introduction of direction or destination specific counters which often caused inequitable distribution of queues and passenger dissatisfaction.

5. Tickets had to be bought from the station of journey origination only and on the date of journey as providing more flexibility would have increased the type and number of tickets to be stocked beyond manageable limits.

6. The system required elaborate accountable processes which not only resulted in the creation of a large back office but consumed half an hour of each Booking Clerk's ticket issuing shift as each ticket in stock at the window had to be taken or handed over.

7. Accounting reports were sent manually to Zonal railway headquarters on a 10 day periodicity. These were often missed or incomplete so there was no reliable and up to date revenue accountable at the central level.

2.2 DFS Algorithms

There are many techniques for searching a graph. The DFS algorithm extends the current path as far as possible before backtracking to the last choice point and trying the next alternative path. Given a graph G = (V, E) where V stand for set of vertices and E stands for set of edges. A vertex u ϵ V, where we want to explore each vertex in graph. Let n = |V| and m = |E|. Basically a graph can be of two types: directed and undirected. Graph can be represented by two techniques : 1) by matrix, 2) by linked list. Now we assume graph is represented by a linked list. The advantages of such representation are:

i)It requires $\Theta(n+m)$ space to store the vertices and there corresponding list, as opposed to $\Theta(n^2)$ for the adjacency matrix

ii)It makes it possible to go through the neighbours of a vertex u in O(|adj[u]|) time, linear in the number of neighbours.

Graphs form a suitable abstraction for problems in many areas like chemistry, electrical engineering, sociology and many more. Thus it is important to have the most economical algorithms for answering graph-theoretical questions.

DFS will process the vertices first deep and then wide. After processing a vertex, it recursively processes all of its descendants. Backtracking and depth first search is a technique which has been widely used to finding a solution of combinatorial theory and artificial intelligence [6].

Suppose G is a graph and we want to explore it. Initially all the vertices are unexplored than we start from a random vertex of graph G and now follow adjacent edge, traversing edge and visit new vertex we select single adjacent vertex and we continue in this way. At each step, we select an unexplored edge leading from a vertex already visited and we traverse this edge. The edge leads to some vertex, either new or already visited. Whenever we run out of edges leading from old vertices, we choose some unvisited vertex, if any exists, and begin a new exploration from this point. Eventually we will traverse all the edges of G, each exactly once[7].

2.2.1 Overall Strategy of DFS Algorithm

Depth-first search selects a source vertex *s* in the graph and paint it as "visited." Now the vertex *s* becomes our current vertex. Then, we traverse the graph by considering an arbitrary edge (u, v) from the current vertex *u*. If the edge (u, v) takes us to a painted vertex *v*, then we

back down to the vertex *u*. On the other hand, if edge (u, v) takes us to an unpainted vertex, then we paint the vertex *v* and make it our current vertex, and repeat the above computation. Sooner or later, we will get to a "dead end," meaning all the edges from our current vertex *u* takes us to painted vertices. This is a deadlock. To get out of this, we back down along the edge that brought us here to vertex *u* and go back to a previously painted vertex *v*. We again make the vertex *v* our current vertex and start repeating the above computation for any edge that we missed earlier. If all of *v*'s edges take us to painted vertices, then we again back down to the vertex we came from to get to vertex *v*, and repeat the computation at that vertex. Thus, we continue to back down the path that we have traced so far until we find a vertex that has yet unexplored edges, at which point we take one such edge and continue the traversal. When the depth-first search has backtracked all the way back to the original source vertex, s, it has built a DFS tree of all vertices reachable from that source. If there still undiscovered vertices in the graph, then it selects one of them as the source for another DFS tree. The result is a forest of DFS-trees. Note that the edges lead to new vertices are called discovery or tree edges and the edges lead to already visited (painted) vertices are called back edges[9].

In order to keep track of progress, depth-first-search colours each vertex. Each vertex of the graph is in one of three states:

1. Undiscovered,
2.Discovered but not finished (not done exploring from it), and
3. Finished (have found everything reachable from it) i.e. fully explored.

The state of a vertex, *u*, is stored in a color variable as follows:

1. color[*u*] = White - for the "undiscovered" state,

2. color[*u*] = Grey - for the "discovered but not finished" state, and

3. color[*u*] = Black - for the "finished" state.

Like BFS, depth-first search uses $\pi[v]$ to record the parent of vertex *v*. We have $\pi[v]$ = NIL if and only if vertex *v* is the root of a depth-first tree. DFS time-stamps each vertex when its color is changed.

1. When vertex *v* is changed from white to grey the time is recorded in d[*v*].
2. When vertex *v* is changed from grey to black the time is recorded in f[*v*].

The discovery and the finish times are unique integers, where for each vertex the finish time is always after the discovery time. That is, each time-stamp is an unique integer in the range of 1 to 2|V| and for each vertex v, d[v] < f[v]. In other words, the following inequalities hold:

2.2.2 Algorithm Depth-First Search

The DFS forms a depth-first forest comprised of more than one depth-first trees. Each tree is made of edges (u, v) such that u is gray and v is white when edge (u, v) is explored. The following pseudo code for DFS uses a global timestamp time[8].

DFS (V, E)

```
1.  for each vertex u in V[G]
2.    do color[u] ← WHITE
3.       π[u] ← NIL
4.  time ← 0
5.  for each vertex u in V[G]
6.    do if color[u] ← WHITE
7.          then DFS-Visit(u)          ▷ build a new DFS-tree from u
```

DFS-Visit(u)

```
1.  color[u] ← GRAY                    ▷ discover u
2.  time ← time + 1
3.  d[u] ← time
4.  for each vertex v adjacent to u    ▷ explore (u, v)
5.    do if color[v] ← WHITE
6.          then π[v] ← u
7.               DFS-Visit(v)
8.  color[u] ← BLACK
9.  time ← time + 1

10. f[u] ← time                        ▷ we are done with u
```

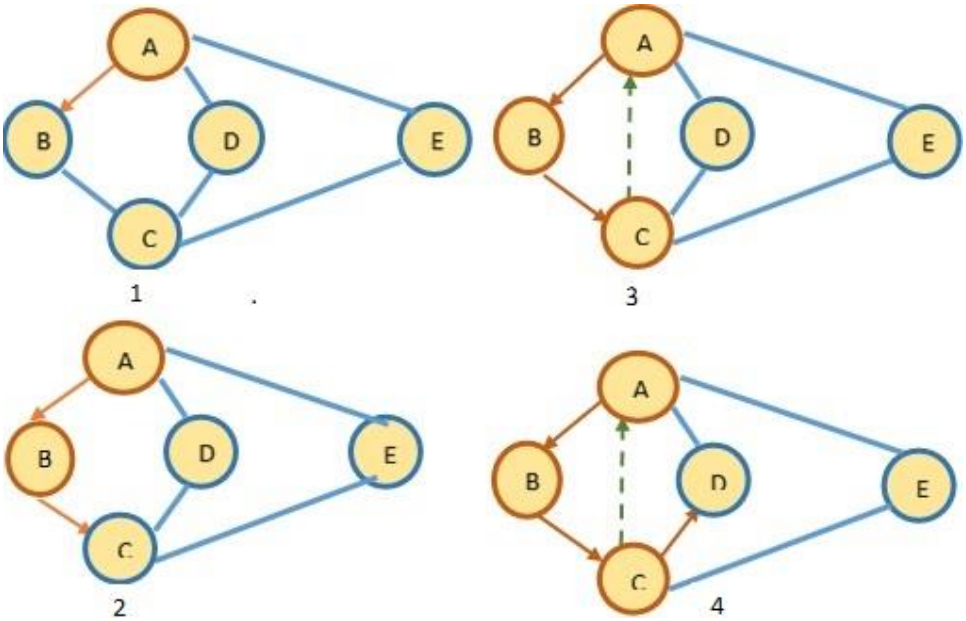

Figure 2.1 DFS Strategy first Four Phases

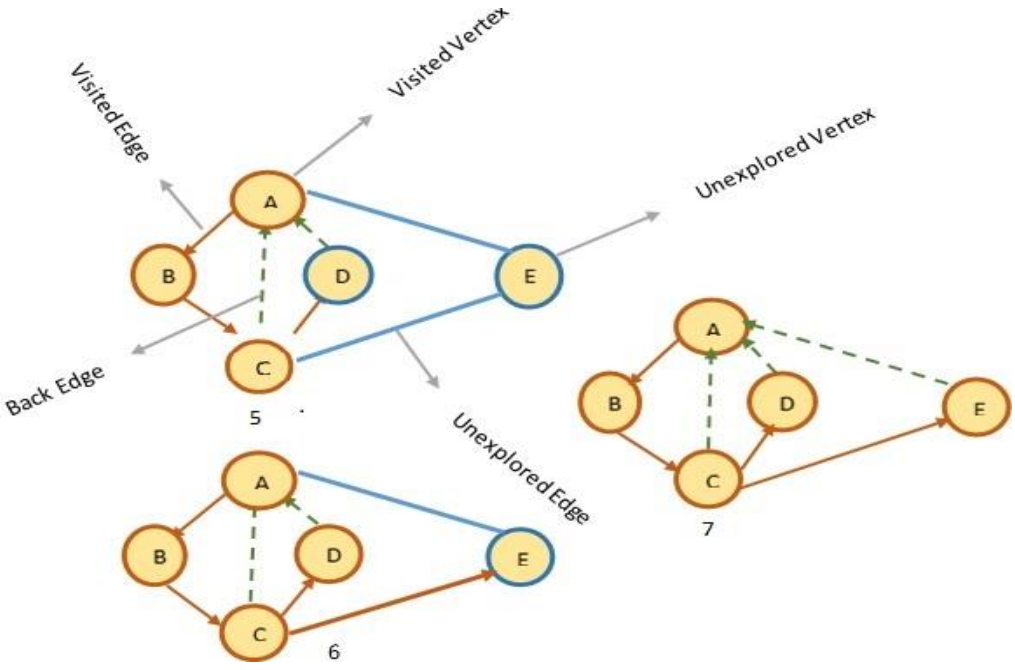

Figure 2.2 DFS Strategy last Phases with naming of Edge and Vertex

2.3 Work in DFS

With the help of DFS we can perform many real life applications:

2.3.1 A DFS Algorithm for Train Re-scheduling

In this application, we present a fast and effective approach for railway traffic re-scheduling which aims to minimize the delays during a disturbance by the use of heuristics and parallelization techniques. The approach is a parallel depth-first search branch-and-bound (B&B) algorithm based on a sequential greedy algorithm proposed by [5, 4].

A branch and bound (B&B) procedure is proposed for resource constrained project scheduling formulation by incorporating an exact lower bound rule and a beam search heuristic is used for tight upper bound [6]. A four step heuristic is proposed in [7], in which 0/1 integer linear programming are used to accept or reject the solution.

The Mixed-Integer Linear Program (MILP) model showed to be too time-consuming to solve using the existing solver for more severe disturbances. Therefore, a greedy depth-first search branch-and-bound algorithm was developed for addressing the re-scheduling problem [5]. The algorithm was further improved in [4] with a more efficient branching strategy.

The main objective of the sequential greedy algorithm is to quickly find a feasible and good-enough solution, and therefore it performs a depth-first search (DFS). It uses an evaluation function to prioritize when conflicts arise and branches according to a set of criteria. When a first feasible solution has been found the algorithm continues to search for improvements if the time limit permits it. In our experiments we have set the time limit to 30 seconds. A detailed description of the algorithm is given in [5].

2.3.2 Application for Road Networks Structure by Enhanced DFS

Road networks structure need fast algorithms to extract appropriate features at a reasonable time. One of the popular acceleration algorithms that achieve this objective is the Depth First Search computation in a large and dense graph. The algorithm is enhanced in by utilizing the hMetis partitioning which is a hyper graph partitioning algorithm that divides a massive graph into sub-graphs or fragments each containing nodes. The enhanced technique is called the DFS_h algorithm [10]. One of the features of the DFS_h algorithm is that it starts randomly from any node in any fragment and it processes the other fragments piece by piece based on the border nodes and edges. Taking into account time and the number of I/O operations, the results are compared with those of the traditional Depth first search where the

proposed DFS_h algorithm proves to be more effective on the Jordan road networks structure [9].

In this algorithm develop an acceleration algorithm for enhancing DFS computation (DFS_h) on a large dense graph based on the RAM model. The main idea is to divide the large graph into the set of sub-graphs that fit in the memory by using hMetis [8]. The partition techniques reduce the number of crossing edges between the sub-graphs, which reduce the number of I/O operations.

2.3.3 Depth First Search Algorithm for SPIHT in minimum memory usage

SPIHT and block-wise SPIHT algorithms where full Depth First Search algorithm is used to agglomerate significant bits at each bit plane. Search strategies used for SPIHT to date are more or less based on a Breadth First Search algorithm. The aim of this work is to minimize the final memory usage without paying additional overhead cost [11]. DFS also brings benefits such as resolution scalability and a random access decodable bitstream.

The aim of the SPIHT algorithm is to locate significant bits in each bitplane of the wavelet coefficients With1 the minimum tree cost. Therefore it is important to implement an efficient search algorithm. The search starts from the highest energy bit plane level and continues down to the lowest energy bit plane. The search path at every traversal of a bit plane is registered in three lists in terms of types of the branches and the importance of the pixels: lists of insignificant sets (LIS), list of insignificant (LIP) and significant (LSP) pixels. As the algorithm traverses a bit plane, LIS relates the roots of the same spatial orientation to each other with the type information, while 2x2 pixels of each block with a significant element are addressed in the lists of LIP and LIS.

2.3.4 Semi-External Depth First Search heuristics

Computing the strong components of a directed graph is an essential operation for a structural analysis of it. This problem can be resolved through twice running a depth-first search (DFS). In an external setting, in which all data can no longer be stored in the primary memory, the DFS problem is not solved so far. Suppose that node-related data can be stored internally, semi-external computing does not make the problem substantially easier. Considering the definite need to analyse very huge graphs, in which they are developed a set of heuristics which together allow the performance of semi-external DFS for directed graphs in practice. The heuristics have been applied to graphs with very different graph properties, including “web graphs” as described in the most recent literature and some large call graphs

from ATT [12]. Depending on the graph structure, the program is between 10 and 200 times faster than the best alternative, a factor that will further increase with future technological developments.

Depth-first search, *DFS*, is a basic and crucial operation on graphs. On undirected graphs it can be used for computing the biconnected components. On directed graphs DFS is the key routine to computing strongly connected components. DFS can also be used for determining whether a graph is acyclic, and, if yes, computing a topological sorting. For graphs with n nodes and m edges, sequential DFS can be performed in $O(n + m)$ time. This algorithm accesses the n adjacency lists of the nodes in an a priori unpredictable order which implies that it exploits random memory access in an essential way. In an *external* setting, where the data do not fit in the main memory [12], it performs extremely badly. Surprisingly, there are no substantially better external memory algorithms which means that in practice the above mentioned problems cannot be solved for graphs such as the web graph or the call graphs (in which the nodes represent customers and every directed edge represents a telephone call) of telephone companies.

2.3.5 DFS-BFS Technique for Reversible Logic Circuit

Logic synthesis with reversible circuits has received considerable interest in the light of advances recently made in quantum computation. In this paper, we propose an improved technique for synthesizing reversible circuits based on a combined depth-first search (DFS) and breadth-first search (BFS) algorithm. A method based on DFS alone may often take a long time to converge, whereas, a BFS based method requires a large amount of memory for designing a circuit of moderate complexity. To strike a balance between these two approaches, a hybrid DFS-BFS based synthesis algorithm that reduces the computation time compared to the DFS method and requires less space compared to the BFS method, while optimizing the cost of the circuit. Synthesis results on several reversible benchmark circuits have been reported [13].

2.4 Parallel Algorithms for Depth-First Search

The basic idea behind parallel algorithms is obvious: If we have more than one processor at our disposal, we can solve a problem more quickly by dividing it into independent sub-problems and solving them at the same time, one on each processor. The running time of the algorithm is then the longest running time of any of these processors; more specifically, given input n, the running time T(n) on input n is the elapsed time from when the first processor begins executing to when the last processor stops executing[15].

We say that a parallel algorithm for a given problem is optimal if its processor bound P(n) and its time bound T(n) are such that P(n)T(n) = O(S), where S is the running time of the best known sequential algorithm for the problem. Thus parallel algorithms should meet at least two criteria: T(n) should be as small as possible, and they should be optimal.

There are four popular models of parallel computation: shared memory models, boolean circuits, fixed connection networks, and parallel comparison trees [14].

Chapter 3

Problem Statement

This chapter define what is in existing system and what is proposed for improving the existing system.

3.1 Existing System

When we go for the reservation first we open the IRCT. Which required own account or required registration and then after we log in and now we can fill detail of our reservation for example Source, Destination and date. And now system gives the details of the Train according to which we fill. And then we select the train one of them which we prefer and then it will give the detail about seats availability.

Everything was fine till now but after that if seats available than everything is fine but if seats not available than we don't have any other choice except to change the train or date. But most of the time we didn't get the reservation. And all this happened because we don't have presumptions or detail about the how much seats available from station to station.

3.2 Problem Statement

The solution of this problem is given with the help of presumptions. With the help of DFS, DFS_h, hmetis and DBMS the solution of this problem is given. Two approaches to solve this problem have been taken: first by graph searching algorithm and other by a DBMS and dynamic queries. So problem is that when we go for the reservation we didn't find seat while some seats are available between the stations because each station have own quota for seats. We try to understand this problem by example. So scenario is "if Train's final source and Destination is Amritsar and Delhi respectively and we have a total 50 seats for reservation coach and in between of these stations, 30 stations exist" below a table mention of stations .

Table 3.1 Shows Route of a Train and availability of seats

No.	Station code	Name	No. of seats
1	ASR	Amritsar jn	50
2	JNL	Jandiala	50
3	BEAS	Beas	50
4	KRE	Kartarpur	50

5	JUC	Jalandhar City	50
6	JRC	Jalandhar Cant	50
7	PGW	Phagwara Jn	50
8	GRY	Goraya	50
9	PHR	Phillaur Jn	50
50	LDH	Ludhiana Jn	50
11	AHH	Ahmadgarh	50
12	MET	Malerkotla	50
13	DUI	Dhuri Jn	50
14	NBA	Nabha	50
15	PTA	Patiala	50
16	RPJ	Rajpura Jn	50
17	UBC	Ambala City	50
18	UMB	Ambala Cant Jn	50
19	SHDM	Shahbad Markndа	50
20	KKDE	Kurukshetra Jn	50
21	NLKR	Nilokheri	50
22	TRR	Taraori	50
23	KUN	Karnal	50
24	GRA	Gharaunda	50
25	PNP	Panipat Jn	50
26	SMK	Samalkha	50
27	GNU	Ganaur	50
28	SNP	Sonipat	50
29	RDDE	Rathdhana	50
30	NUR	Narela	50
31	SZM	Subzi Mandi	50
32	NDLS	New Delhi	50

And each station have 50 seats for Reservation we take some scenarios to proper understanding the problem.

1. Some people reserve the 10 seats form **Amritsar** to **Ludhiana** Jn than 10 seats are deduct from all the stations between Amritsar to Ludhiana.
2. Another one Reserve 20 form **Amritsar** to **Delhi** than 20 seats deduct from all the station between Amritsar to Delhi.
3. Reserve 10 seats from **Jalandhar** to **Dhuri jn** than 10 seats deduct from all the station between Jalandhar to Dhuri.
4. Reserve 10 seats from Ludhiana to Delhi than 10 seats deduct from all the stations between Ludhiana to Delhi.

But now if a person want to reserve the seats from **Malerkotla** to **Delhi** than it will show the waiting and doesn't provide any suggestion while if we see the chart than we find two station far seats are available but present reservation system provide no suggestions this suggestion method is also applied in a social networking sides like Facebook or Twitter etc. So for this suggestions we applied two method, one of them is implemented and theoretical concept is given about the another one.

Chapter 4

Implementation

4.1 Circuit Partitioning through hMetis

METIS and hMETIS is an algorithm which divide circuit in small-small parts. METIS is an algorithm for partitioning huge irregular graphs, partitioning huge meshes, and calculating fill-reducing orderings of matrices which is sparse, to partition graph in equal size METIS provides two programs called pmetis and kmetis.

The multilevel graph partitioning by Metis algorithm: pmetis is based on multilevel recursive bisectioning described in [26] and kmetis is based on multilevel k-way partitioning described in [27]. Multilevel partitioning algorithms first of all reduce the size of the circuit by rough graph's details. This takes form as collapsing adjacent vertices and edges.

The smaller graph is then dividend and refined into the original graph. As the partitioning algorithms operate with the reduced-size graph, which are extremely fast as compared to traditional partitioning algorithms that compute a partition directly on the original graph. Extensive testing has also concluded that the partitions provided by METIS are consistently better than those produced by spectral partitioning algorithms [28].

The hMETIS is an extension of METIS that uses the hypergraphs in place of graphs [29]. A hypergraph is a generalization of a graph that allows an edge | i.e., a hyperedge | to connect more than two vertices. A stand-alone program shmetis was used to partition the hypergraphs.
METIS and hMETIS both are designed to operate on large graphs with generally more than 1000 vertices.

4.1.1 Overview of the Algorithms

In the rest of this section, we briefly describe the various phases of the multilevel algorithm. The reader should refer to [10] for further details.

- **Coarsening Phase**

During the hypergraph coarsening phase, successively smaller hypergraphs in sequence are constructed. The aim of coarsening is to create a small hypergraph, in a manner that a good bisection of the small hypergraph is not significantly worse than the bisection directly obtained for the original hypergraph. In addition to that, hypergraph coarsening also helps in

successively reducing the size of the hyperedges. That is, after several levels of coarsening the large hyperedges are contracted to hyperedges connecting just a few vertices. This is particularly helpful, since refinement heuristics based on the Kernighan-Lin algorithm [30, 32] are very effective in refining small hyperedges but are quite ineffective in refining hyperedges with a large number of vertices belonging to different partitions. The group of vertices that are contracted together to form single vertices in the next level coarse hypergraph can be selected in different ways. hMETIS implements various such types of grouping schemes (also called *matching schemes*) some of which are described in [31].

- **Initial Partitioning phase**

During the initial partitioning phase, computation of a bisection of the coarsened hypergraph is done. Since this hypergraph has a very small number of vertices (usually less than 100 vertices) many different algorithms can be used without significantly affecting the overall runtime and quality of the algorithm. hMETIS uses multiple random bisections followed by the Fiduccia-Mattheyses(FM) refinement algorithm.

- **Uncoarsening and refinement phase**

During the uncoarsening phase, the partitioning of the coarsest hypergraph is used to obtain a partitioning for the finer hypergraph. Which has done by successively projecting the partitioning to the next level finer hypergraph and using a partitioning refinement algorithm to reduce the cut and thus improve the quality of the partitioning. As the next level finer hypergraph has more degrees of freedom, such types of refinement algorithms tend to improve the quality and hMETIS implements a variety of algorithms that are based on the FM algorithm [32]. The details of some of these schemes can be found in [31].

- ***V*-Cycle Refinement**

The idea behind this refinement algorithm is to use the power of the multilevel paradigm to further improve the quality of bisection. The *V*-cycle refinement algorithm has two phases, named as a coarsening and an uncoarsening phase. The initial partitioning that is input to the algorithm preserved by the coarsening phase. We will refer to this as *restricted coarsening* scheme. In restricted coarsening scheme, the groups of vertices that are combined to form the vertices of the coarse graphs correspond to vertices that belong only to one of the two partitions. So that, the original bisection is preserved throughout the coarsening process, and becomes the initial partition from which we start performing refinement during the

uncoarsening phase. The uncoarsening phase of the *V* -cycle refinement algorithm is identical to the uncoarsening phase of the multilevel hypergraph partitioning algorithm that have been described earlier. It moves vertices between partitions as long as such moves improve the quality of the bisection. Note that the various coarse representations of the original hypergraph, allow refinement to further improve the quality as it helps it climb out of local minima.

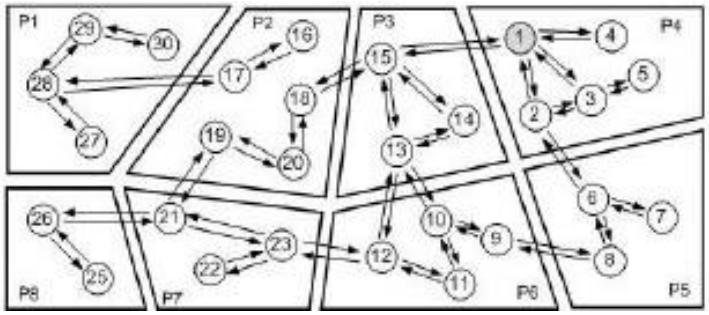

Figure 4.1 Partitioning the graph in the eight past

4.2 DFS-h Algorithm

Our algorithm has two main steps: the first step is the pre-processing step that uses hMetis. The second is when we apply (DFS_h) to compute the DFS for all fragments. The hMetis algorithm partitions the vertices of a graph into fragments and the edges of the graph are of two types: local and border. A local edge has vertices which belong to the same fragment, whereas a border edge has vertices of which belong to different fragments and the two fragments are said to be adjacent if they have edges with endpoints belonging to different fragments. The endpoints of the border edges are called border nodes. All other nodes are local nodes. For example, P3 and P4 are adjacent, but P1 and P3 are not. Figure 4.1 illustrates these concepts. The pre-processing step considers specifying the type of each node in the fragment.

The DFS_h algorithm is presented below. It starts by specifying the fragment number which contains the Node Id (step 1) and transferring the fragment to the memory. If it's colour was white then we call DFS_Visit_h function (step 2.1), otherwise we call Recover Point function (step 3). Recover Point function returns -1 only when we finished exploring all paths in all fragments, otherwise it changes the color of the remaining border nodes from Gary to Black (step 4.1.1). This is valid for all explored points (Recover Point function returns -1), for the other unexplored points we call DFS_Visit_h function (step 4.2) for another fragment.

DFS_h (NodeId) (compute the depth first search using DFS-h)

1. FragNo=FragmentNo (NodeId)
2. If (color (NodeId) = White) Then
2.1 DFS_Visit_h (NodeId);
3. P_id= RecoverPoint(FragNo);
4. If P_id= -1 Then
4.1 For each node RemId ∈ List of all Border nodes
4.1.1 Color (RemId) = Black
4.1.2 Finish //Finish traversing all nodes in all partitions
4.2 Else DFS_Visit_h (p_id);

The DFS_Visit_h function given below, changes the color of the node NodeId to Gray and explores all the adjacent nodes that belong to the same fragment whose colors are white. Recursively we call DFS_Visit_h function until we finish exploring all nodes in the same fragment. Then we select the latest discover border node whose color is gray and use it to switch to another fragment

- **The DFS_Visit_h algorithm**

DFS_Visit_h (NodeID)

1. Color (NodeId) = Gray
2. For each node j ∈ adj (NodeId)) do
2.1 If (fragment (j) = fragment (NodeId) and color (j) =White)
2.1.1 then DFS_Visit_h (j)
3. If (NodeID) is local node
3.1 Color (NodeId)= Black

The RecoverPoint function given below. Selects the most recent visited border node in the fragment whose color is gray and use it to explore more paths in different fragments using one of the border nodes. It returns -1 when all fragments have been visited.

- **The RecoverPoint Algorithm**

RecoverPoint (FragNo)

1. For each NodeIdk ∈ BorderList (FragNo)
1.1 If (NodeIdk the latest visited one)

1.1.1 For each node NodeIdj ∈ Adjacent (NodeIdk)

1.1.1.1 If FragmentNo (NodeIdj) ≠ FragmentNo (NodeIdk) and color(NodeIdj) =White

1.1.1.1.1 Return NodeIdj

1.1.1.1.2 Else return -1

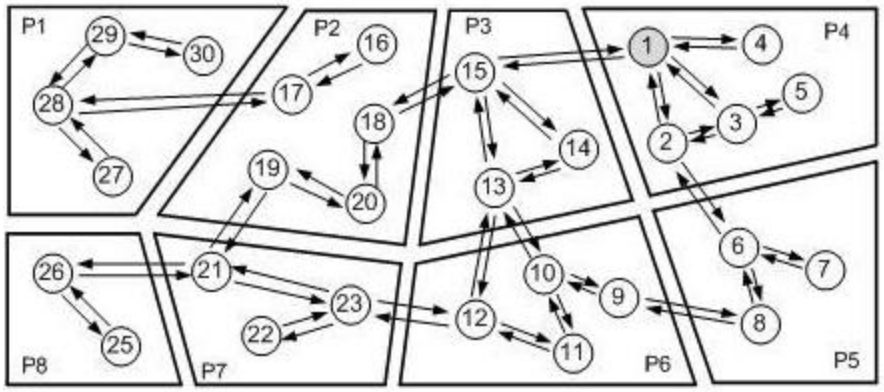

Figure 4.2 Apply DFS_h algorithm in graph which is divided by hMetis, First Step

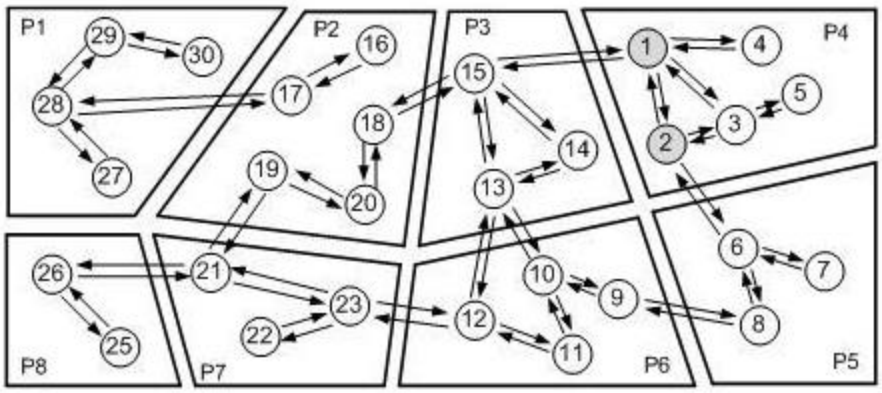

Figure 4.3 DFS_h Algorithm Second Step

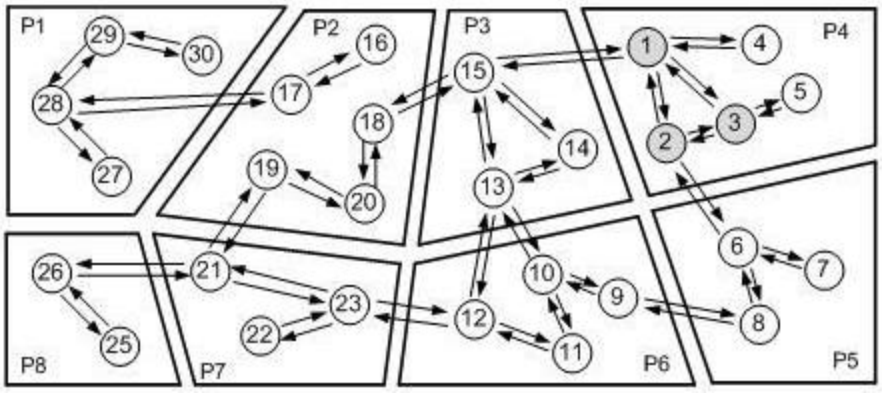

Figure 4.4 DFS_h Algorithm Third Step

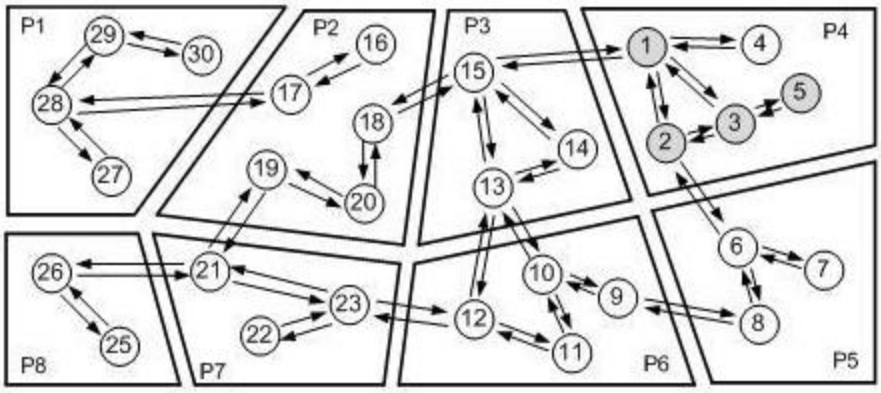

Figure 4.5 DFS_h Algorithm Fourth Step

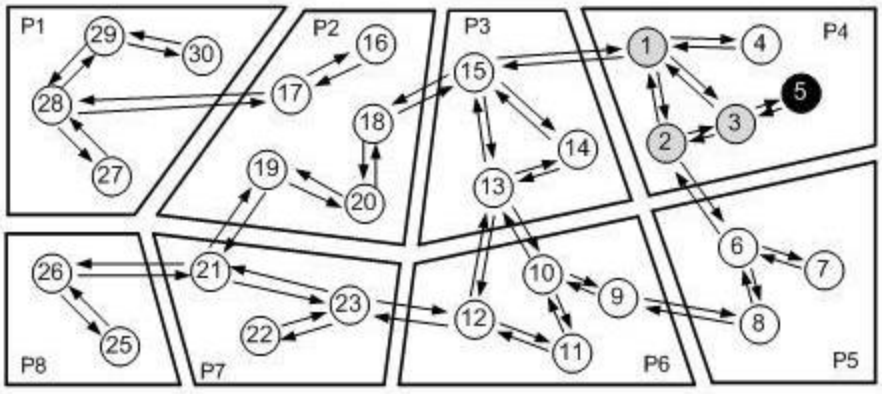

Figure 4.6 DFS_h Algorithm Fifth Step

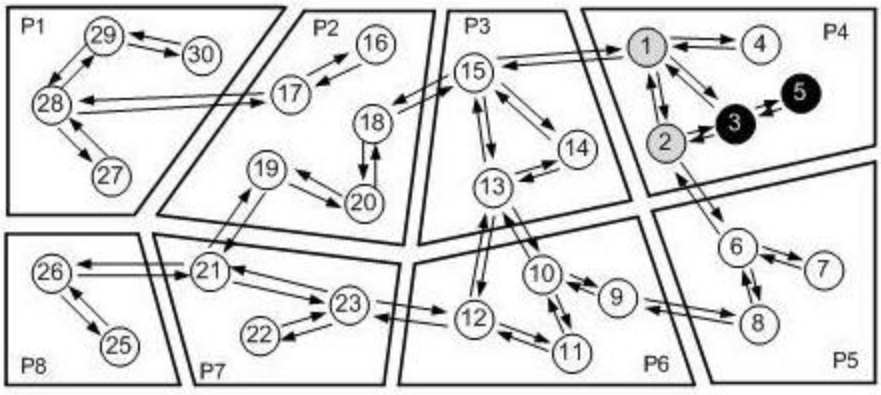

Figure 4.7 DFS_h Algorithm Sixth Step

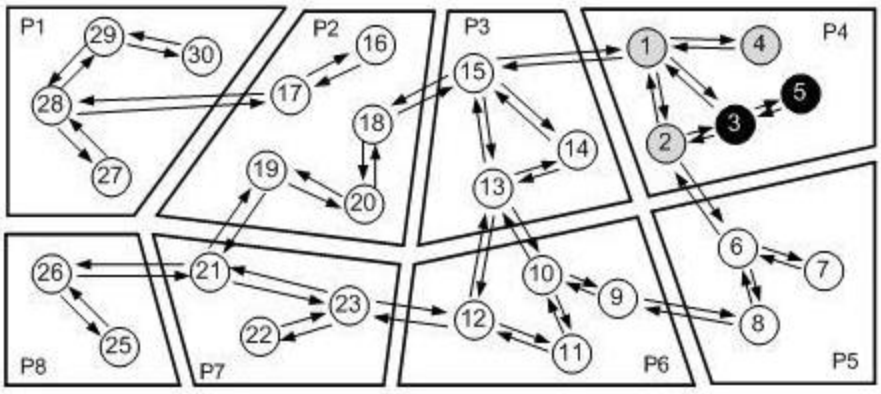

Figure 4.8 DFS_h Algorithm Seventh Step

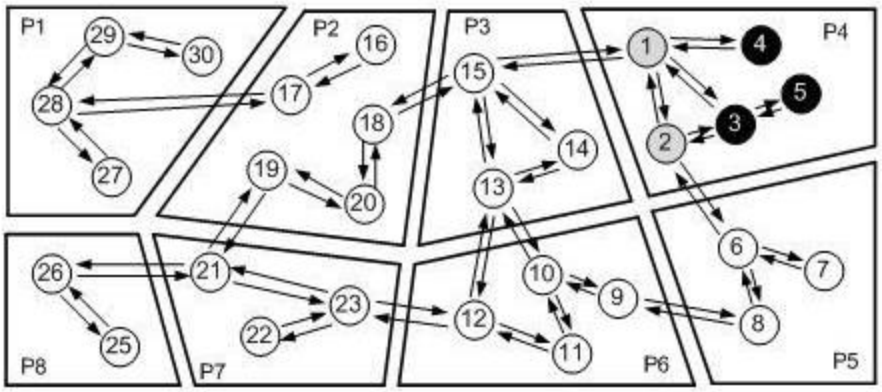

Figure 4.9 DFS_h Algorithm Eight Step

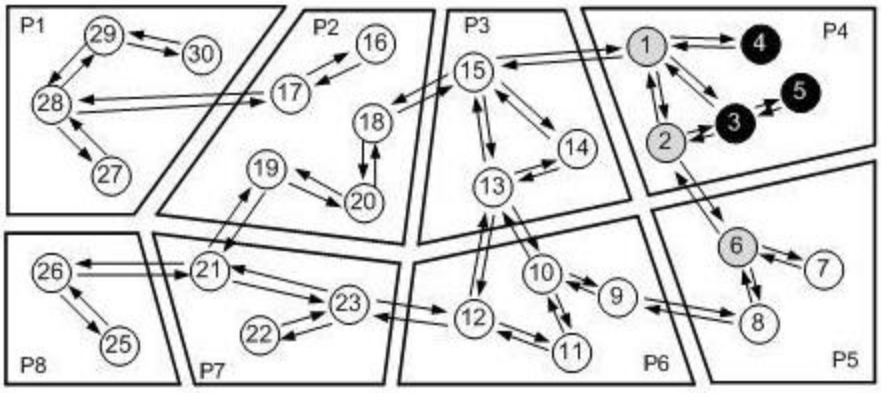

Figure 4.10 DFS_h Algorithm Ninth Step

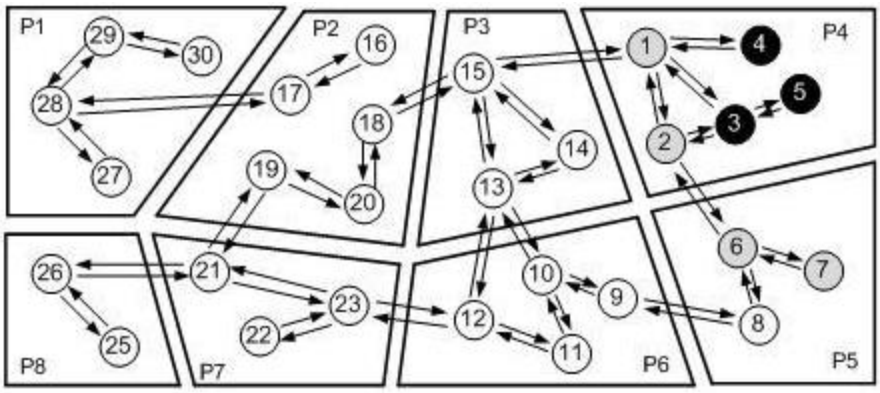

Figure 4.11 DFS_h Algorithm Tenth Step

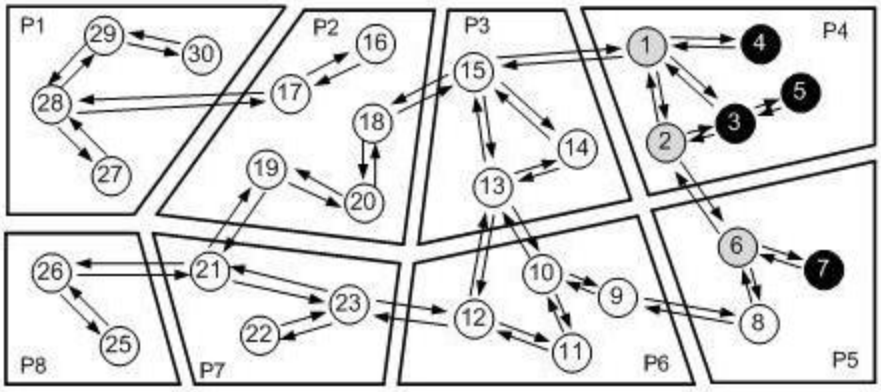

Figure 4.12 DFS_h Algorithm Eleventh Step

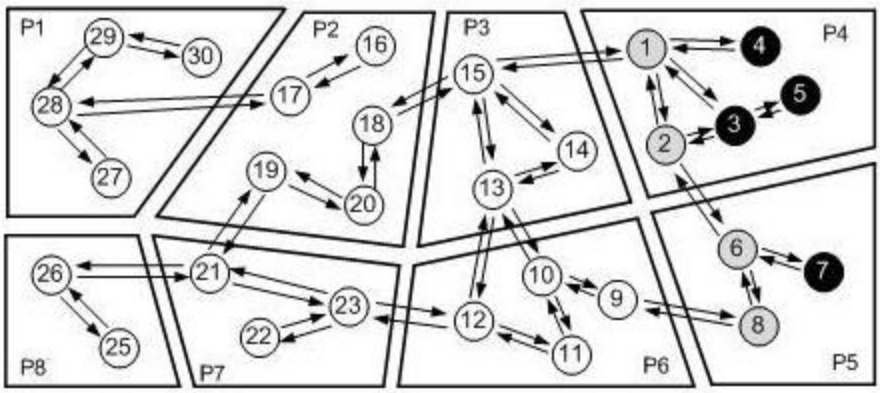

Figure 4.13 DFS_h Algorithm Twelth Step

Now we are able to understand how to divide the graph of Railway in our application stations are the node and the railway track are the edges with the help of hMetis and DFS_h algorithm

we can divide the our railway graph and this method is very fast to partitioning the graph and then traverse the graph with the help of DFS_h algorithm.

Now we are going to understand, how these algorithm use in our application. So first of all we divide the huge railway network with the help of hMetis algorithm and then we apply DFS_h algorithm on the graph or network to traverse the graph.

Throw some modification in DFS_h we can find the adjacent node or station of the particular node or station.

In our application first we select the source and destination and then we search all the adjacent of the source and destination and according to the source and destination we find the trains. And see how much train on this root. Select one train which is suitable for us. For proper understanding we take an example and with help of example we try to understand problem and solve it. So we take previous example which ix show in a third section table 3.1. in which we select the source and destination for example we take Ludhiana jn and Panipath jn Respectively so with the help of DFS_h algorithm we find all the adjacent of the source and destination we apply the DFS_h algorithm for two nodes and then return on original source .

When we find all the Adjacent of source and destination than check there is root available or not. If root is not available than drop the node and check another one if we find the root from source adjacent to destination or destination adjacent. Then we select the train and check for availability but there is six scenarios can be explained one by one.

First Scenario

In the first scenario we take ultimate source and destination which is original source and destination where our boarding and departure is same as our source and destination which is chosen. This scenario comes only when we get succeed in reservation. This scenario is a normal scenario which is also come in our old reservation system.

Second scenario

In the second scenario we deal with our original application's advantage, if we didn't get the reservation than with the help of our algorithm it will provide the next two station and give the details about the availability, for example, if we want to book the ticket from Ludhiana to Panipath and we didn't get the reservation or seat than we have a detail about the next two

stations for example Ahmadnagar and Malerkotla to our original destination sometimes it happened we didn't get the reservation by original destination but we find reservation from next station to destination . But we didn't get the presumption in our old system.

Third scenario

In the third Scenario if we didn't find the availability of seats in second scenario than we go for the third scenario and now we will take the next two sources and see the availability status in before and after of destination. For example if we boarding from Ahmadnagar or Malerkotla and destination is before two or one station from destination or after the destination. For example Karnal and Gharaunda or Samlkha and Ganour. If we take before the destination than we can follow the same train but if we choose station after the destination than we have another advantage to choose another root for example some trains is taken a different route from next source station to after the destination it may not include our original destination and that time we fail for find the train in our old system and in this situation we go manually for find the train which is very time consuming.

Fourth scenario

In the fourth scenario, if we fail to get the reservation in Third scenario than we come in a fourth scenario in which we just try just before from the original source to original destination in which we have less chance to make a different route, but it may happened sometimes we get reservation because of each station have own quota for seats in train and in this scenario it will also provide a others trains options. For example if our original source is Ludhiana than we search for previous station of Ludhiana like Phillaur junction or Goraya junction to original destination Panipat.

Fifth scenario

In this scenario, we will take a route from previous station of original source to before or after stations of original destination. It may provide the new route and seats also and if route is same then may be possible there is availability of seat. How seats are available? Its answer is already explained in section three but there is we take another example which is clear all five scenarios

This is the route of train no. 12716 "ASR NED EXPRESS"

Table 4.1 Shows Route of Train No 12716

Source	Destination	No. of seats
Amritsar jn	Jandiala	10
Jandiala	Beas	10
Beas	Kartarpur	10
Kartarpur	Jalandhar City	10
Jalandhar City	Jalandhar Cant	10
Jalandhar Cant	Phagwara Jn	10
Phagwara Jn	Goraya	10
Goraya	Phillaur Jn	10
Phillaur Jn	Ludhiana Jn	10
Ludhiana Jn	Ahmadgarh	10
Ahmadgarh	Malerkotla	10
Malerkotla	Dhuri Jn	10
Dhuri Jn	Nabha	10
Nabha	Patiala	10
Patiala	Rajpura Jn	10
Rajpura Jn	Ambala City	10
Ambala City	Ambala Cant Jn	10
Ambala Cant Jn	Shahbad Markndа	10
Shahbad Markndа	Kurukshetra Jn	10
Kurukshetra Jn	Nilokheri	10
Nilokheri	Taraori	10
Taraori	Karnal	10
Karnal	Gharaundа	10
Gharaundа	Panipat Jn	10
Panipat Jn	Samalkha	10
Samalkha	Ganaur	10
Ganaur	Sonipat	10
Sonipat	Rathdhana	10
Rathdhana	Narela	10
Narela	Subzi Mandi	10

Subzi Mandi	New Delhi	10
New Delhi	Null	10

This is the route of train no. 12926 "PASCHIM EXPRESS"

Table 4.2 Shows Route of Train No 12926

Source	**Destination**	**No. of seats**
Amritsar jn	Jalandhar City	10
Jalandhar City	Jalandhar Cant	10
Jalandhar Cant	Phagwara Jn	10
Phagwara Jn	Ludhiana Jn	10
Ludhiana Jn	Ahmadgarh	10
Ahmadgarh	Malerkotla	10
Malerkotla	Dhuri Jn	10
Dhuri Jn	Rajpura Jn	10
Rajpura Jn	Ambala City	10
Ambala City	Ambala Cant Jn	10
Ambala Cant Jn	Shahbad Markndа	10
Shahbad Markndа	Kurukshetra Jn	10
Kurukshetra Jn	Nilokheri	10
Nilokheri	Taraori	10
Taraori	Karnal	10
Karnal	Gharaunda	10
Gharaunda	Panipat Jn	10
New Delhi	Mathura	10
Mathura	Agra	10
Agra	Dholpur	10
Dholpur	Gwalior	10

Now we take an example in which we make reservation from Patiala to new Delhi and we assume only 10 seats are capability of the Train and we chose a train whose no is 12716 because there is only one train which is go from Patiala to Delhi. If we reserve 5 seats for

Patiala to Delhi, then we easily get the reservation and it is exist in first scenario. Now we have only 5 seats remaining which is show below in table 4.3

Table 4.3 Shows reduced Seats when Reserving the Seats and Shows First Scenario

Source	**Destination**	**No. of seats**
Amritsar jn	Jandiala	10
Jandiala	Beas	10
Beas	Kartarpur	10
Kartarpur	Jalandhar City	10
Jalandhar City	Jalandhar Cant	10
Jalandhar Cant	Phagwara Jn	10
Phagwara Jn	Goraya	10
Goraya	Phillaur Jn	10
Phillaur Jn	Ludhiana Jn	10
Ludhiana Jn	Ahmadgarh	10
Ahmadgarh	Malerkotla	10
Malerkotla	Dhuri Jn	10
Dhuri Jn	Nabha	10
Nabha	Patiala	10
Patiala	Rajpura Jn	5
Rajpura Jn	Ambala City	5
Ambala City	Ambala Cant Jn	5
Ambala Cant Jn	Shahbad Markndа	5
Shahbad Markndа	Kurukshetra Jn	5
Kurukshetra Jn	Nilokheri	5
Nilokheri	Taraori	5
Taraori	Karnal	5
Karnal	Gharaunda	5
Gharaunda	Panipat Jn	5
Panipat Jn	Samalkha	5
Samalkha	Ganaur	5
Ganaur	Sonipat	5
Sonipat	Rathdhana	5

Rathdhana	Narela	5
Narela	Subzi Mandi	5
Subzi Mandi	New Delhi	5
New Delhi	Null	5

We have only 5 seats from Patiala to Delhi but we see that before Patiala station still we have 10 seats but if want to book 10 seats from Dhuri to Panipath than we can't. Because we have only five seats from Patiala to Panipath in this situation we have only one choice which is just change the train but we see than from Patiala only one train. Now we required manual search in older system but proposed in proposed we have multiple choice it will provide another train also if we didn't get the reservation. We think about the second scenario in which if didn't get the reservation than it search from next station to original destination and next station is Ambala. And we find that there is another train where seats are available. But this is not exactly second scenario. For the second scenario we will take another example in which we reserve the seats from Amritsar to Rajpura if we reserve 5 seats from Amritsar to Rajpura and then if we want to reserve the 10 seats from Patiala to Delhi than we didn't get reserve whole 10 seats we only get 5 reservation successfully and 5 are waiting because only five seats are available so for this if we think second scenario we get success which is show below in table.

Table 4.4 Showing Second Scenario

Source	**Destination**	**No. of seats**
Amritsar jn	Jandiala	5
Jandiala	Beas	5
Beas	Kartarpur	5
Kartarpur	Jalandhar City	5
Jalandhar City	Jalandhar Cant	5
Jalandhar Cant	Phagwara Jn	5
Phagwara Jn	Goraya	5
Goraya	Phillaur Jn	5
Phillaur Jn	Ludhiana Jn	5
Ludhiana Jn	Ahmadgarh	5
Ahmadgarh	Malerkotla	5
Malerkotla	Dhuri Jn	5

Dhuri Jn	Nabha	5
Nabha	Patiala	5
Patiala	Rajpura Jn	5
Rajpura Jn	Ambala City	10
Ambala City	Ambala Cant Jn	10
Ambala Cant Jn	Shahbad Markndа	10
Shahbad Markndа	Kurukshetra Jn	10
Kurukshetra Jn	Nilokheri	10
Nilokheri	Taraori	10
Taraori	Karnal	10
Karnal	Gharaunda	10
Gharaunda	Panipat Jn	10
Panipat Jn	Samalkha	10
Samalkha	Ganaur	10
Ganaur	Sonipat	10
Sonipat	Rathdhana	10
Rathdhana	Narela	10
Narela	Subzi Mandi	10
Subzi Mandi	New Delhi	10
New Delhi	Null	10

Now if we see the table than we find remaining five seats are available if we reserve the seats from Rajpura to Delhi.

In third scenario it may happened some people reserve 5 seats from Gharaunda to Delhi and we want to reserve the 7 seats from Patiala to Panipat than we didn't get the reservation success fully and if we think about the third Scenario than we will find some seats are available from next station to before destination which is we see below in table

Table 4.5 Showing third Scenario

Source	**Destination**	**No. of seats**
Amritsar jn	Jandiala	5
Jandiala	Beas	5

Beas	Kartarpur	5
Kartarpur	Jalandhar City	5
Jalandhar City	Jalandhar Cant	5
Jalandhar Cant	Phagwara Jn	5
Phagwara Jn	Goraya	5
Goraya	Phillaur Jn	5
Phillaur Jn	Ludhiana Jn	5
Ludhiana Jn	Ahmadgarh	5
Ahmadgarh	Malerkotla	5
Malerkotla	Dhuri Jn	5
Dhuri Jn	Nabha	5
Nabha	Patiala	5
Patiala	Rajpura Jn	5
Rajpura Jn	Ambala City	10
Ambala City	Ambala Cant Jn	10
Ambala Cant Jn	Shahbad Marknda	10
Shahbad Marknda	Kurukshetra Jn	10
Kurukshetra Jn	Nilokheri	10
Nilokheri	Taraori	10
Taraori	Karnal	10
Karnal	Gharaunda	10
Gharaunda	Panipat Jn	5
Panipat Jn	Samalkha	5
Samalkha	Ganaur	5
Ganaur	Sonipat	5
Sonipat	Rathdhana	5
Rathdhana	Narela	5
Narela	Subzi Mandi	5
Subzi Mandi	New Delhi	5
New Delhi	Null	5

If we see table than we will find that we have 10 seats available from Rajpura to Gharaunda but Gharaunda to Panipat we have only five seats are available so we get only five

reservation successfully remaining 2 for waiting but if we have assumption than we can successfully reserve the seats just before the destination so now our scenario is like below table 4.6

Table 4.6 Showing Third And Fourth Scenario

Source	Destination	No. of seats
Amritsar jn	Jandiala	5
Jandiala	Beas	5
Beas	Kartarpur	5
Kartarpur	Jalandhar City	5
Jalandhar City	Jalandhar Cant	5
Jalandhar Cant	Phagwara Jn	5
Phagwara Jn	Goraya	5
Goraya	Phillaur Jn	5
Phillaur Jn	Ludhiana Jn	5
Ludhiana Jn	Ahmadgarh	5
Ahmadgarh	Malerkotla	5
Malerkotla	Dhuri Jn	5
Dhuri Jn	Nabha	5
Nabha	Patiala	5
Patiala	Rajpura Jn	5
Rajpura Jn	Ambala City	3
Ambala City	Ambala Cant Jn	3
Ambala Cant Jn	Shahbad Markndа	3
Shahbad Markndа	Kurukshetra Jn	3
Kurukshetra Jn	Nilokheri	3
Nilokheri	Taraori	3
Taraori	Karnal	3
Karnal	Gharaunda	3
Gharaunda	Panipat Jn	3
Panipat Jn	Samalkha	5
Samalkha	Ganaur	5

Ganaur	Sonipat	5
Sonipat	Rathdhana	5
Rathdhana	Narela	5
Narela	Subzi Mandi	5
Subzi Mandi	New Delhi	5
New Delhi	Null	5

Now we clearly understand the problem many times we see that seats are available between the stations but we didn't find the seats because of lake of presumptions. But this is not exactly third scenario some time we see that every station has own quota and they have own seats we can book seats till after the original destination for example if we book seats up to Delhi than may be because of station quota we can find the seats this is a different thing so this particular situation we can't show throw table so it will also be create so with the help of presumption also we can find this situation.

Now fourth Scenario in which we just try to reserve the seat just before (Last station) from our original station. It is similar thing as show in scenario third, only difference is that, in the third scenario we reserve the seat up to next station of destination but in the fourth scenario we are boarding from our original station but reserve the seat from previous station because each station has own quota, we are little bit confuse because of our previous examples, but for this scenario we take a different example where each station has own quota. And quotas decided by the population of the city. Now we take an example which is clearly define the scenario.

Table 4.7 Shows Quota of seats for a Station and Example for fourth and fifth scenario

Source	**Destination**	**No. of seats**
Amritsar jn	Jalandhar City	20
Jalandhar City	Jalandhar Cant	10
Jalandhar Cant	Phagwara Jn	10
Phagwara Jn	Ludhiana Jn	7
Ludhiana Jn	Ahmadgarh	10
Ahmadgarh	Malerkotla	7

Malerkotla	Dhuri Jn	5
Dhuri Jn	Rajpura Jn	6
Rajpura Jn	Ambala City	5
Ambala City	Ambala Cant Jn	5
Ambala Cant Jn	Shahbad Markndа	10
Shahbad Markndа	Kurukshetra Jn	4
Kurukshetra Jn	Nilokheri	10
Nilokheri	Taraori	6
Taraori	Karnal	4
Karnal	Gharaunda	6
Gharaunda	Panipat Jn	7
Panipat Jn	New Delhi	9
New Delhi	Mathura	30
Mathura	Agra	15
Agra	Dholpur	15
Dholpur	Gwalior	10

Here we see all the scenario where seats are available according to the city's population or the type of the city. Now we take scenario where some people book the seats from Malerkotla to Karnal and they booked 5 seats they successfully got the seats now our scenario is show below in table where we see how to deduct the seats from each station while each station has different capability so for this scenario we take actual scene of railway where we have a capability of train. If train can carry hundred people. Than railway distribute the whole seats according to the city. But if seats are not occupied by a particular station than it will transfer to those stations where waiting is more. And waiting is converted in confirm status.

Table 4.8 Showing Fifth Scenario

Source	**Destination**	**No. of seats**
Amritsar jn	Jalandhar City	20
Jalandhar City	Jalandhar Cant	10
Jalandhar Cant	Phagwara Jn	10
Phagwara Jn	Ludhiana Jn	7
Ludhiana Jn	Ahmadgarh	10

Ahmadgarh	Malerkotla	7
Malerkotla	Dhuri Jn	0
Dhuri Jn	Rajpura Jn	4
Rajpura Jn	Ambala City	3
Ambala City	Ambala Cant Jn	3
Ambala Cant Jn	Shahbad Markndа	7
Shahbad Markndа	Kurukshetra Jn	3
Kurukshetra Jn	Nilokheri	7
Nilokheri	Taraori	4
Taraori	Karnal	3
Karnal	Gharaunda	6
Gharaunda	Panipat Jn	7
Panipat Jn	New Delhi	9
New Delhi	Mathura	30
Mathura	Agra	15
Agra	Dholpur	15
Dholpur	Gwalior	10

Now we see that there is seats are changed according to the city and the quota but the station where our boarding reduce full seats how many u reserved. So with the help of presumption we can find how many seats are available at the last or next station. Fifth scenario also looks like it but the difference is that in the fifth scenario destination also can move. So with the help of all these scenario we can easily find the trains and this is also convenient for the passengers.

Chapter 5

Implementation Results

This chapter gives the detail of implementation of the methodology. The proposed methodologies have been implemented in ASP .Net programming language.

5.1 Implementation of methodology

The proposed methodologies in this thesis have been implemented in ASP .Net programming language. As we have shown that this Methodology is based on the two algorithms with the help of these algorithm we implement the whole methodology. The proposed methodology basically implement the railway reservation system. But this required the huge database system which is not efficient so for this we take a scenario of graphs. With the help of graph we can reduce the huge database size.

5.2 Snapshots

This is the snapshot of simple railway reservation in which we have to fill source, Destination and Date.

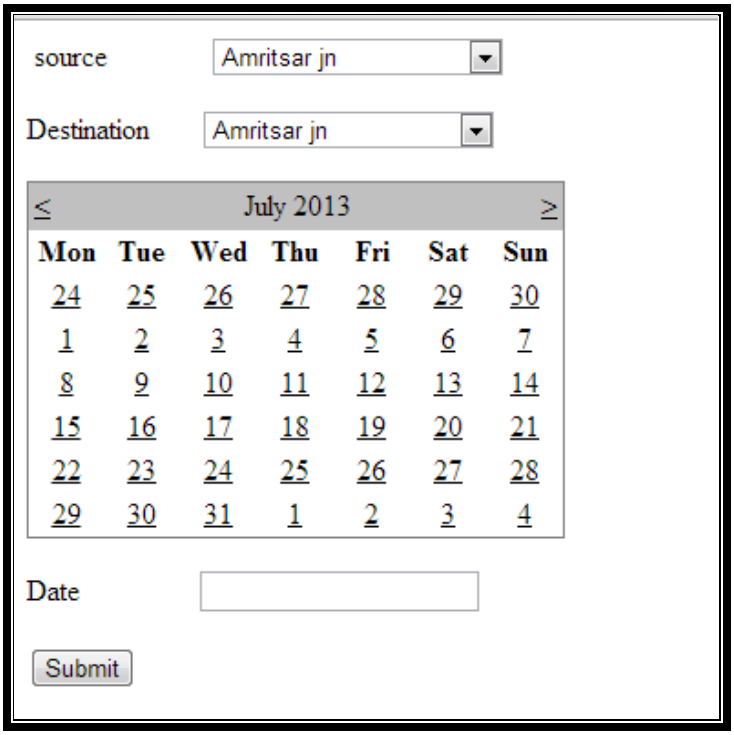

Figure 5.1 First Window of Reserve for Ticket

In which we filled the detail our source, Destination and the Date and then click on Button Submit. And according to our requirement we can find the number of train than we can see the detail of train. How many hours it will take for whole journey? And which route follow by a particular train.

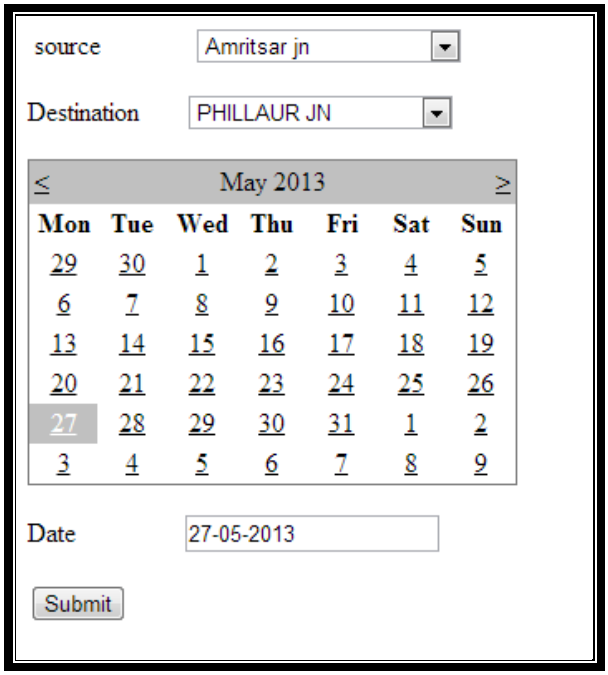

Figure 5.2 select Date, Source and Destination

Next one show the how many trains are available on particular source to destination and Date

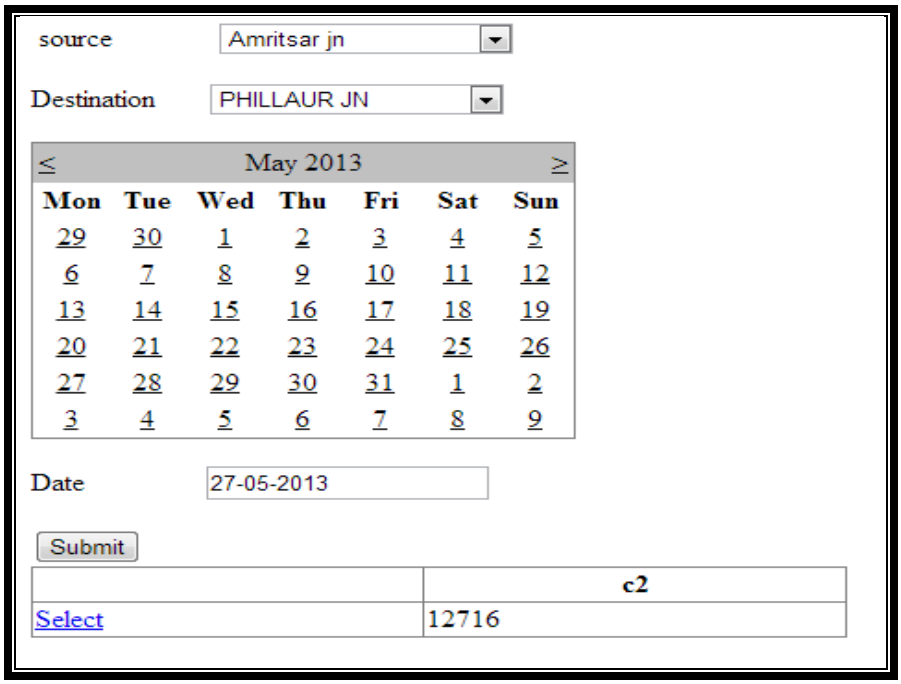

Figure 5.3 Show availability of Train

This snapshot shows how many seats are available on a particular train and we fill the seats according to availability and there is two button show first is booking and second one is assumption which is disable.

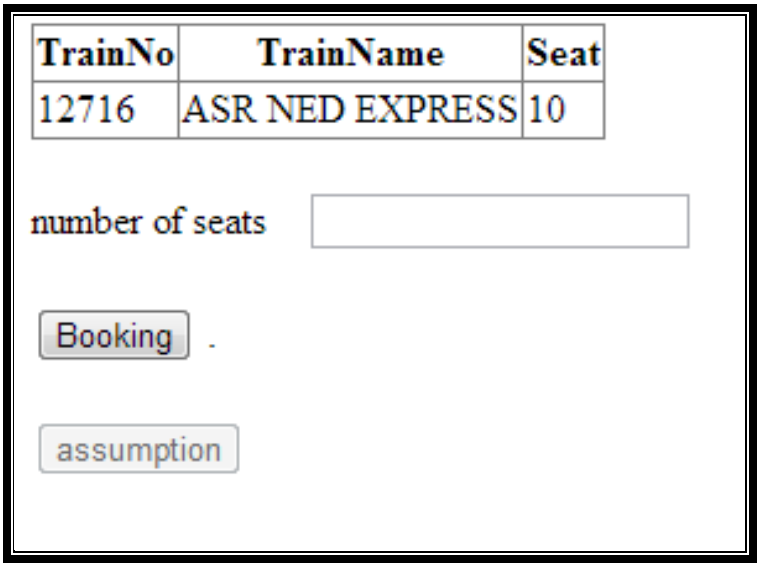

Figure 5.4 shows Availability of Seats and for Booking

Now we fill the seats which are required by us and then click on booking button if seats are available than our reservation will we done successfully. And we see one more thing no. of seats are deducted immediately from the total available seats and still assumption button is disable.

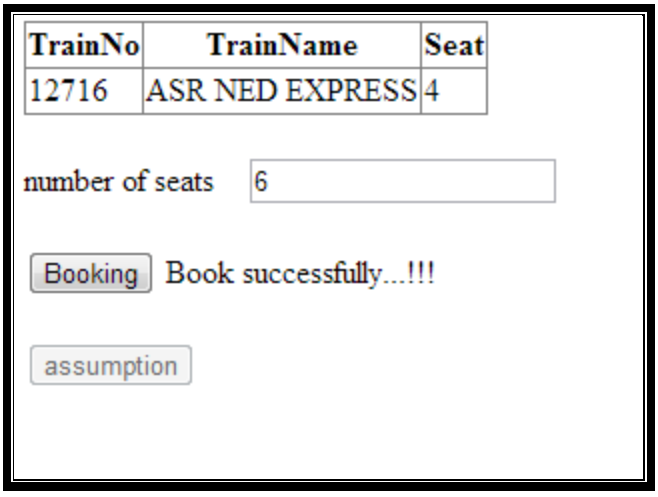

Figure 5.5 Shows Booking is Successful and Remaining Seats

In this screenshot, we reserve seats more than availability than we find a note "Not available…. press Assumption button" and now assumption button is enable .

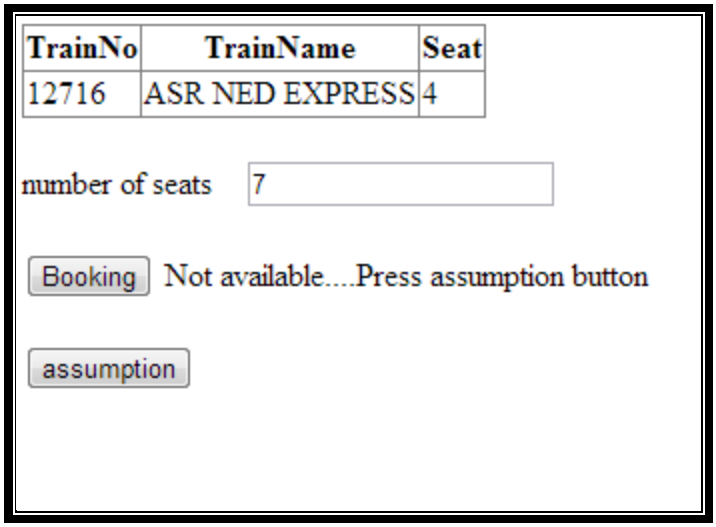

Figure 5.6 shows Assumption Button and not available Signature

In this snapshot we find all assumption which is define in fourth scenario

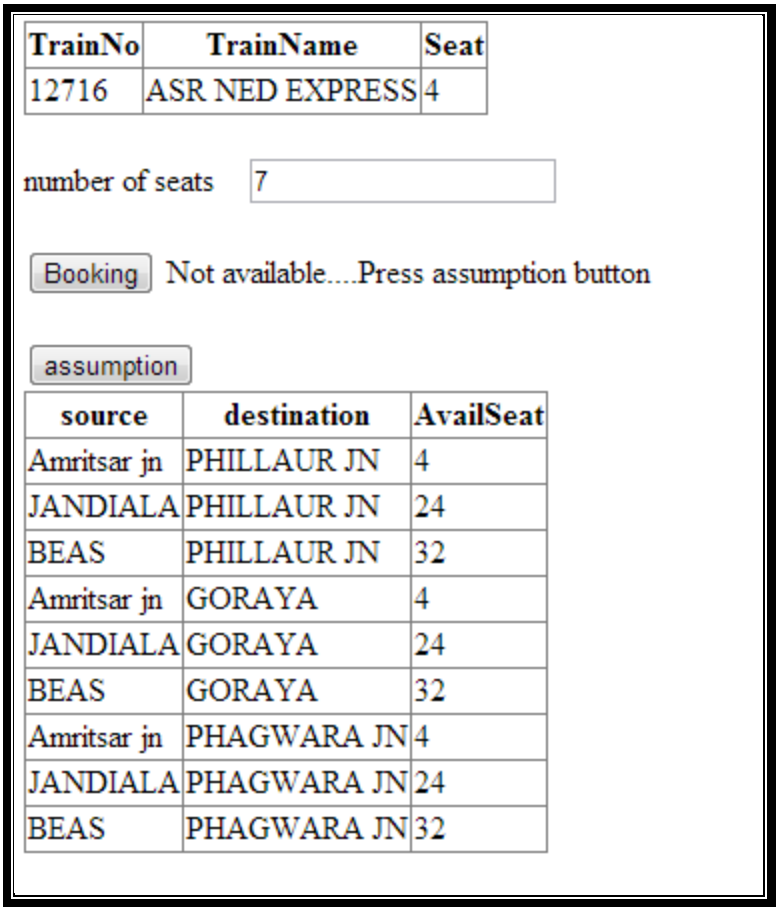

Figure 6.7 showing all Assumptions

Chapter 6

Conclusion and future scope

Railway network structure is searched based on an enhanced Depth First Search computation in a large and dense graph. The proposed acceleration algorithm (DFS_h) is based on hMetis partitioning and it has two important features. First it divides the large graph into fragments where the number of nodes in each fragment is equal. The other important feature of the algorithm is that it reduces the number of the crossing edges. Running time is shortened and the number of I/O operations is reduced when the algorithm is tested on Railway networks structure. Therefore, the proposed algorithm provides great benefits and can be potentially tested on other graph structures that have the appropriate characteristics. Furthermore, a new parallel version of DFS_h can be introduced when deployed on a sparse graph. And with the help of these algorithm we present the more user friendly and efficient system.

Future scope:

It's a new revolution in railway history. If it is implement in railway system than it reduces the loss of railway. Almost all the seats will Occupied by the passengers and if we add extra detail in the system which is actual boarding and actual departure (exit) from the train with source and destination than it is easy to search where and when seats will be free and it's also easy to check tickets from the TC, and it will definitely reduce the corruption in railway. And in which we can also provide one facility if a passengers didn't get the reservation till original station where he /she wants to go. So passenger pay the reservation amount till he get reservation and then after he will pay the general amount which is general bogie amount. With the help of this facility passenger is also feel relax and economic journey.

References

[1] Lixing Yang, Xuesong Zhou, Ziyou Gao, Rescheduling trains with scenario-based fuzzy recovery time representation on two-way double-track railways, Springer-Verlag 2012

[2] Keivan Ghoseiri And Fahimeh Morshedsolouk Acs-Ts: Train Scheduling using Ant Colony System Received 6 July 2005; Revised 15 January 2006; Accepted 18 January 2006

[3] Vikram Chopra, Director Operations, Centre for Railway Information Systems The Unreserved Ticketing System Of Indian Railways, cited as: http://cris.org.in/CRIS/Projects/UTS

[4] J. T¨Ornquist and J. A. Persson, "N-tracked railway traffic re-scheduling during disturbances", Transportation Research Part B: Methodological, Vol. 41, No. 3, pp. 342–362, 2007.

[5] J. T¨ornquist Krasemann, "Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances", Transportation Research Part C: Emerging Technologies, In Press, Corrected Proof, 2010.

[6] X. Zhou and M. Zhong, "Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds", Transportation Research Part B: Methodological, Vol. 41, No. 3, pp. 320 – 341, 2007.

[7] Y. Lee and C.-Y. Chen, "A heuristic for the train pathing and timetabling problem", Transportation Research Part B: Methodological, Vol. 43, No. 8, pp. 837 – 851, 2009.

[8] Karypis , G.; Kumar ,V. , hMETIS Ahypergraph Partitioning Package version 1.5.3. University of Minnesota, 1998.

[9] Vitter , Jeffrey Scott, "Algorithms and data structures for external memory", Foundations and Trends in Theoretical Computer Science, Vol. 2, No. 4, pp. 305-474, 2008.

[10] Sahar Idwan, Adnan Mukattash, Moh'd Sami Ashhab, Ayat Nizar, Wael Etaiwi, **"**Application of Enhanced Depth First Search Algorithm to Jordan Road Networks Structure", ISSN 1450-216X, Vol. 50, No. 3, pp. 327-332, 2011

[11] Mustafa Sakalli, William A. Pearlman, and Masoud Farshchian SPIHT algorithms using Depth First Search Algorithm with minimum memory usage IEEE 2006

[12] Jop F. Sibeyn, James Abello, Ulrich Meyer, "Heuristics for Semi-External Depth First Search on Directed Graphs", *SPAA'02,* August 10-13, 2002, Winnipeg, Manitoba, Canada. Copyright 2002 ACM

[13] Dipak K. Kole1, Hafizur Rahaman, Debesh K Das, and Bhargab B. Bhattacharya, "Optimal Reversible Logic Circuit Synthesis based on a Hybrid DFS-BFS Technique" 2010 International Symposium on Electronic System Design.

[14] Sanguthevar Rajasekaran and John H. Reif. Randomized parallel computation, In *Proc. International Conference on Fundamentals of Computation Theory,* Kazan, June *1987*. Springer-Verlag Lecture Notes in Computer Science *278*, pp. *364-376*.

[15] Jon Freeman, "Parallel Algorithms For Depth-First Search", Department of Computer and Information Science University of Pennsylvania Philadelphia, PA 19 104 October 1991

[16] Jon Kleinberg and ÉvaTardos, *Algorithm Design*. Pearson Education, 2006.

[17] Don Sannella CS2 Algorithm and data structure note 10 CS2BH 31 January 2005

[18] Broder, A., R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata and A. Tomkins, J. Wiener, "Graph Structure in the Web", *Computer Networks*, Vol. 33, pp. 309–320, June 2000.

[19] Aggarwal, A., R.J. Anderson, "A Random NC Algorithm for Depth First Search", *Combinatorica*, Vol. 8, No. 1, pp. 1–12,1988.

[20]. F. Corman, "Real-time Railway Traffic Management, Dispatching in complex, large and busy railway networks", Ph.D. thesis, Technische Universiteit Delft, The Netherlands, December 2010, 90-5584-133-1.

[21]. A. Grama and V. Kumar, "State of the art in parallel search techniques for discrete optimization problems", IEEE Trans. on Knowledge and Data Engineering, Vol. 11, No. 1, pp. 28–35, 2002.

[22] X. Zhou and M. Zhong, "Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds", Transportation Research Part B: Methodological, Vol. 41, No. 3, pp. 320 – 341, 2007.

[23] L. Ingolotti, A. Lova, F. Barber, P. Tormos, M.A. Salido, and M. Abril. New heuristics to solve the csop railway timetabling problem. *Advances in Applied Artificial Intelligence. LNAI, Subseries of Lecture Notes in Computer Science*, 2006.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[25] M. Goodrich and R. Tamassia, *Algorithm Design*. John-Wiley and Sons. 2002.

[26] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs", SIAM Journal on Scientific Computing, 1998.

[27] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning", Journal of Parallel and Distributed Computing, Vol. 48, No. 1, pp. 96-129, 1998.

[28] G. Karypis and V. Kumar, "METIS, A software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0", http://glaros.dtc.umn.edu/gkhome/metis/metis/download

[29] G. Karypis and V. Kumar, "hMETIS, A Hypergraph Partitioning", Package Version 1.5.3," http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download

[30] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, Vol. 49, No. 2, pp. 291–307, 1970.

[31] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *In Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[32] George Karypis, Rajat Aggarwal, Vipin Kumar and Shashi Shekhar, Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the Design and Automation Conference*, 1997

List of Publications

Communicated

1. Gaurav Rathi and Shivani Goel, "Depth first Search and its Applications" Journal of Embedded System & Applications