Cost based Graphical Vertical Fragmentation in Relational Database

A Dissertation Submitted in partial fulfillment of the requirement

For the Award of degree of

MASTER OF TECHNOLOGY

In

Software Engineering

By

Kushal Verma

(Roll No. 05/SWE/2010)

Under the Guidance of

Dr. Rajni Jindal

(Associate Professor and Project Guide)

Delhi Technological University

Department of Computer Engineering

Delhi Technological University

July – 2012

CERTIFICATE

This is to certify that project entitled "Cost based graphical vertical fragmentation in relational database" has been completed by Kushal Verma**, Roll No. 05/SWE/2010** in the partial fulfillment of the requirement for the award of degree of **Master of Technology** in **Software Engineering**.

This work is carried out by him under my supervision and support. This is a beneficial work in the field of large scale distributed Database.

Dr. Rajni Jindal
Associate Professor & Project Guide
(Dept. of Computer Engineering)
Delhi Technological University
Bawana Road, Delhi- 110042.

ACKNOWLEDGEMENT

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisor **Dr. Rajni Jindal** (Associate Professor of Computer Engineering Dept., Delhi Technological University) for her precious guidance, encouragement and patient reviews. Her continuous inspiration only has made me complete this dissertation. All of them kept on boosting me time and again for putting an extra ounce of effort to realize this work.

I am also thankful to **Prof. Dr. Daya Gupta**, Head of the Department, Computer Engineering, Delhi Technological University, Delhi 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 the hour and provided with all the help and facilities, which I required for the completion of my thesis work.

Finally I am also thankful to my classmates and my family members for their unconditional support and motivation during this work.

Kushal Verma
Master of Technology
(Software Engineering)
Dept. of Computer Engineering
Delhi Technological University
Bawana Road, Delhi -110042

Table of contents

Figure Index

Table Index

Abstract

In a distributed database environment, query processing heavily depends upon locality of the requested data at the query site. When relations are vertically fragmented, data locality can be improved significantly since the fragments can be replicated flexibly without replication incurring overwhelming update cost.

Vertical partitioning clusters attributes of a relation to generate fragments suitable for subsequent allocation over a distributed platform. Partitioning is also important for centralized and web databases because data items that are mostly accessed together should reside in the same fragment. The target is to minimize the execution time of user applications.

In this thesis we use graphical and heuristic approach for vertical fragmentation scheme for the DDBMS to allocate the fragments generated by the graphical approach. In our proposal where allocation is handled along with Fragmentation, Fragmentation will be done by using graphical approach and then using the cost model of heuristic approach, these fragments will be allocated.

Input consists of a set of relations, together with information about the important transactions on the proposed database. It is not necessary to collect information about 100% of transactions. According to 80-20 rule [2], 20% of heavily used transactions account for the 80% of database activity. Hence we provide information about 20% of heavily used transaction.

CHAPTER 1

Problem Definition

1.1 Overview

Now a day's size of databases is so huge that it's almost impossible to keep all the data at one place. Even the organizations are spreading their businesses all around the world, which will create bottleneck problem with that single database server. Many large organizations of different types have a history of separate business units developing and maintaining independent customer databases. Typically these legacy systems have been developed autonomously and use a variety of data structures and identifiers to record personal information. In addition, these databases are often 'owned and operated' by separate functional units within the organization. Consequently, the personal information an organization holds about individuals will be fragmented across a number of databases using a variety of different data structures. This makes accessing and collating personal information difficult and time-consuming. Rarely in these cases is there a unified and consolidated view of the information an organization holds about an individual.

Now, the design of distributed database is an optimization problem and the resolution of several sub problems such as data fragmentation (horizontal, vertical, and hybrid), data allocation (with or without redundancy), optimization and allocation of operations (request transformation, selection of the best execution strategy, and allocation of operations to sites). There are some different approaches to solve each problem, so this means that the design of the distributed databases becomes cumbersome. There are many

researches connected to the data fragmentation and they are presented both in the case of relational database and in the case of object-oriented database. Here we present a new technique for graphical partitioning approach for vertical fragmentation in distributed databases, especially to solve the problem of allocation in distributed databases.

1.2 Related Work & Motivation

The concept of using fragmentation and allocation of data as means of improving the performance of database management systems has often emerged in the literature. Most of the past research considers either horizontal fragmentation schemes or vertical fragmentation schemes [1] [2] [4] [5] [6].

Shamkant B. Navathe, K Karlapalemand M. Ra proposed a mixed fragmentation methodology by means of graphical approach to both HF and VF in [5]. This uses the graphical approach using partitioning algorithm given in [1] for both horizontal and vertical partitioning. This is not a cost model hence it does not deal with the problem of allocation. Allocation of fragments to sites would normally involve a cost model. To the best of our knowledge no such hybrid fragmentation approach exists that performs fragmentation and allocation simultaneously. Our emphasis in this paper is a cost based graphical partitioning approach for VF. We have developed a cost model that addresses the problem of allocation in distributed databases. Our methodology uses the modified formulae of heuristic approach [6] for vertical fragmentation by H. Ma, KD Schewe and M. Kirchberg. By combining both of these techniques we can do the allocation of fragments. This allows us to allocate the fragments computed by the graphical partitioning approach. We will discuss this in detail in later chapters.

1.3 Solution Flow

The complete distributed database design is a three step process, namely: initial design, application of design and redesign. Initial design consists of selecting fragmentation and allocation algorithms. Second and third step are iterative processes that depends upon logical and physical changes in the distributed database environment.

Input consists of a set of relations, together with information about the important transactions on the proposed database. It is not necessary to collect information about 100% of transactions. According to 80-20 rule [4], 20% of heavily used transactions account for the 80% of database activity. Hence we provide information about 20% of heavily used transaction. Input to the fragmentation techniques is defines as follows:

- Schema information includes relations, attributes, attribute sizes.
- Information on transactions includes frequency and attributes usage.

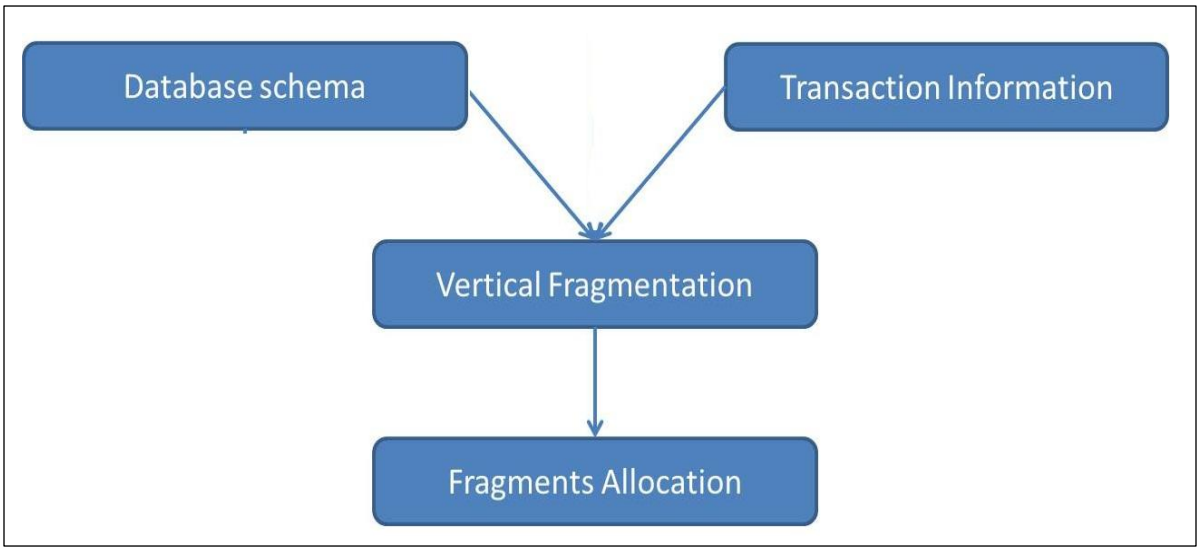


Figure 1: Solution Flow

After gathering input, next is the fragmentation which is done using graphical approach. After generating the fragments from the above approach next is the allocation, for which cost based model is used. The fragments will be allocated to the site where its cost is lesser than the other sites, and cost is nothing but the transaction delay.

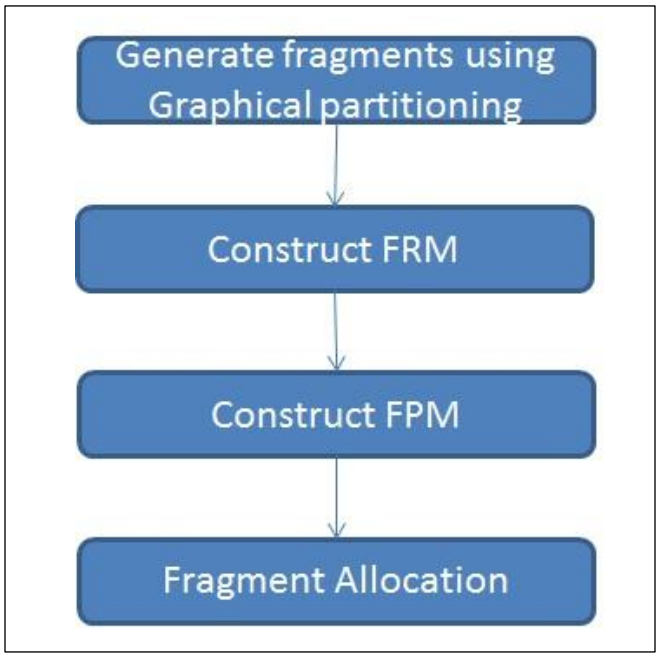


Figure 2: Step by step solution flow

1.4 Organization of Dissertation

The rest of this thesis is organized as follows. In chapter 2, we have presented literature review for distributed databases. It describes the various distributed databases design techniques.

Chapter 3 We will explain Graphical approach using graph partitioning algorithm that partition the relation using attribute affinity. We presented an example to explain the above technique.

Moving to chapter 4 we will explain heuristic approach for VF that uses an attribute usage matrix and cost model. We presented an example to explain the above technique.

In chapter 5 we introduce how to perform allocation of the fragments generated by the graphical partitioning approach. We present the proposed algorithm and an example to explain the above technique.

In chapter 6 we have shown the screen shots of the tool which implements all these approaches.

Finally we conclude with a short summary and the future modifications of the proposed methodology in chapter 7.

CHAPTER 2

Distributed Databases

In the previous chapter we discussed about our approach and motivation behind our work. This chapter explains the various aspects of a distributed database design problem, advantages and disadvantages of distributed databases.

2.1 Introduction

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data in a distributed database can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Primary concern of distributed database system design is to making fragmentation of the relations in case of relational database or classes in case of object oriented databases, allocation and replication of the fragments in different sites of the distributed system, and local optimization in each site.

The fragmentation and allocation of databases improves database performance at end-user worksites. We can influence communication costs, load balancing and availability by fragmenting a relation and allocating it accordingly using an optimized approach.

Fragmentation decomposes a relation into smaller, disjunctive fragments. These fragments are distributed across nodes or may be replicated. Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be very complex and time consuming depending on the size and number of the distributive databases. This process can also require a lot of time and computer resources. On the other hand allocation deals with keeping the fragments at the sites where they are required most. Assigning fragments to sites in the computer network depends upon communication cost and transaction information. Using a cost model we can efficiently minimize the cost of remote access and avoid bottleneck problem.

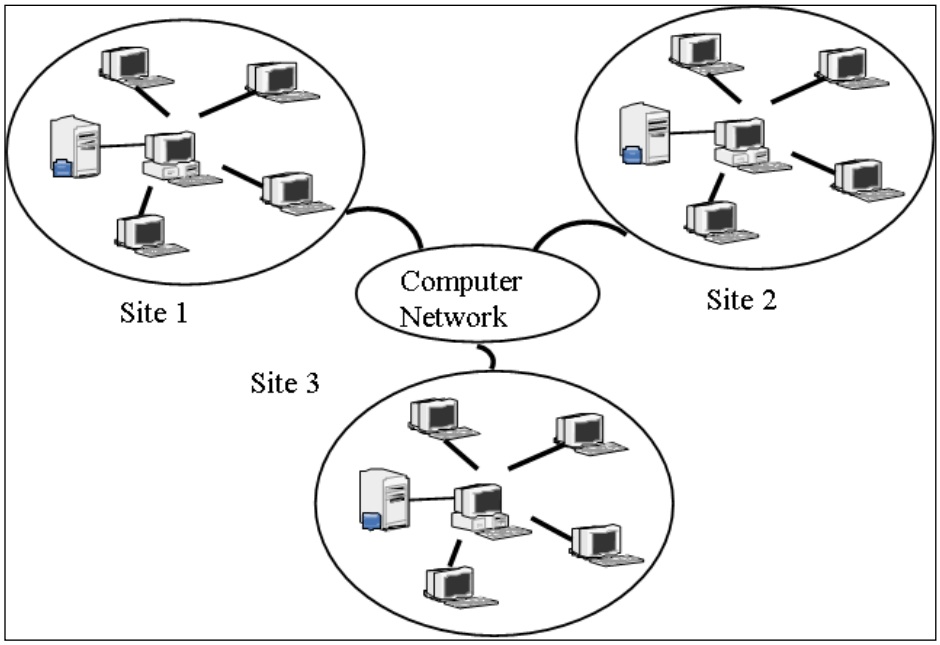


Figure 3 : Distributed Databases

The design of distributed databases is an optimization problem requiring solutions to following two problems:

- Designing the fragmentation of global relations
- Designing the allocation of fragments to the sites of communication network

2.2 Fragmentation

Distributed processing on database management systems (DBMS) is an efficient way of improving performance of applications that manipulate large volumes of data. This may be accomplished by removing irrelevant data accessed during the execution of queries

and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases. The main reasons of fragmentation of the relations are to

- increase locality of reference of the queries submitted to database
- improve reliability and availability of data and performance of the system, balance storage capacities and minimize communication costs among sites

Fragmentation is a design technique to divide a single relation or class of a database into two or more partitions such that the combination of the partitions provides the original database without any loss of information. This reduces the amount of irrelevant data accessed by the applications of the database, thus reducing the number of disk accesses. Fragmentation can be horizontal, vertical or mixed/hybrid.

- **Horizontal fragmentation** (HF) allows a relation or class to be partitioned into disjoint tuples or instances.
- **Vertical fragmentation** (VF) allows a relation or class to be partitioned into disjoint sets of columns or attributes except the primary key.
- Combination of horizontal and vertical fragmentations to **grid** or **mixed** or **hybrid fragmentations** (MF) are also proposed, which have properties of both. To perform hybrid fragmentation we can perform HF first and then VF (HV fragmentation) or vice versa (VH fragmentation) as shown in the figure 4. Final result is independent of the order of HF and VF.

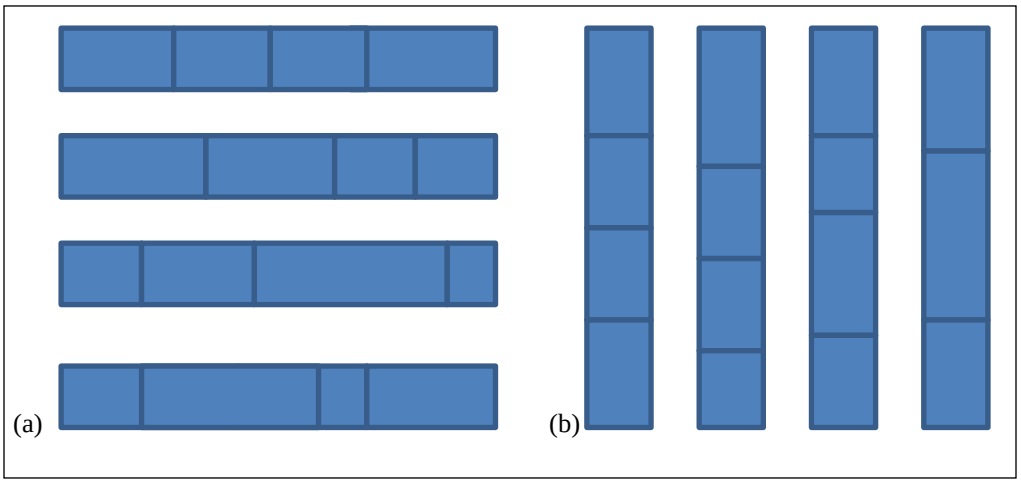


Figure 4 : Hybrid fragmentation (a) HV fragmentation & (b) VH fragmentation

The problem of fragmenting the database is difficult one in itself and variety of approaches exist for fragmenting the database. Many approaches for vertical and horizontal have been researched before. Some of them are graphical approaches that consider predicate affinity [5] or attribute affinity [1] to form clusters. A mixed fragmentation technique [5] uses graphical approach for both HF and VF for creating grid

cells. In this thesis we propose a new type of Graphical Fragmentation method which uses cost model for allocation of fragments generated by the graphical approach. It uses query frequency and cost of a query to calculate attribute locality precedence for HF [9]. For VF it uses cost of allocating a particular attribute to particular site on the basis of transaction information [7].

2.3 Allocation

Allocation is the process of assigning the fragments of a database on the sites of a distributed network. When data is allocated, it may either be replicated or maintained as a single copy. The replication of fragments improves reliability and efficiency of read-only queries but increase update cost.

The problem of allocating data in a distributed database system has an important impact upon the performance and reliability of the system as a whole. The aim is to store the fragments closer to where they are more frequently used in order to achieve best performance.

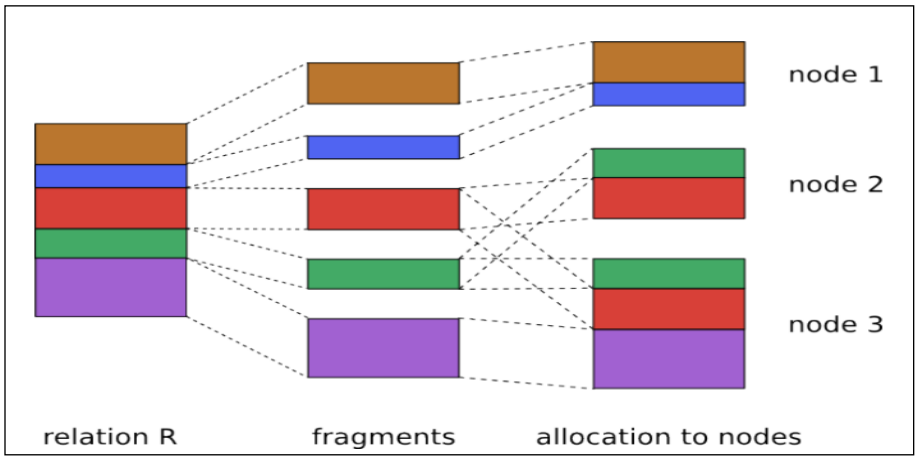


Figure 5 : Fragmentation and Allocation

So, one key principle in distribution design is to achieve maximum locality of data and applications. Since, distributed databases enable more sophisticated communication between sites; the major motivation for developing a distributed database is to reduce communication by allocating data as close as possible to the applications which use them. Thus in a well-designed distributed database 90 percent of the data should be found at the local site, and only 10 percent of the data should be accessed on a remote site. A poorly designed data allocation can lead to inefficient computation, high access cost and high network loads. Various approaches have already evolved for allocation of data in

distributed database. In most of these approaches, data allocation has been proposed prior to the design of a database depending on some static data access patterns and/or static query patterns. In a static environment, where the access probabilities of nodes to fragments never change, a static allocation of fragments provides the best solution. However, in a dynamic environment where these probabilities change over time, the static allocation solution would degrade the database performance.

2.4 Advantages of Distributed Databases

There are multiple advantages of DDB over a CDB. Few of them are listed below from [9].

- Management of distributed data with different levels of transparency.
- Increase reliability and availability.
- Easier expansion.
- Reflects a proper organizational structure — database fragments are located in the departments which they relate to.
- Local autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).
- Reliable transactions - Due to replication of database.
- Hardware, OS, N/w, Fragmentation, DBMS, Replication and Location Independence.
- Continuous operation…
- Distributed Query processing.
- Distributed Transaction management.
- Single site failure does not affect performance of system. All transactions follow A.C.I.D. property: a-atomicity, the transaction takes place as whole or not at all; c-consistency, maps one consistent DB state to another; i-isolation, each transaction sees a consistent DB; d-durability, the results of a transaction must

survive system failures. The Merge Replication Method used to consolidate the data between databases.

2.5 Disadvantages of Distributed Databases

There some disadvantages of a DDB. Few of them are listed below from [9].

- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more expansive infrastructure means extra costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- Difficult to maintain integrity — in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- Inexperience — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- Lack of standards — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- Database design more complex — besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.
- Additional software is required.
- Operating System should support distributed environment.
- Concurrency control: it is a major issue. It is solved by locking and time stamping.

CHAPTER 3

Graphical Approach for VF

Vertical Partitioning is the process of subdividing the attributes of a relation into groups, creating fragments. Vertical partitioning is used to store the most closely accessed attributes in the primary memory. During distributed database design, fragments are allocated and replicated at different sites that improve the performance of transaction processing. For better performance, fragments must be closely matched with the transaction requirements.

In this approach we consider AAM as a complete graph called affinity graph. This method was proposed by S. B. Navathe and M. Ra for VF in [1]. It uses Attribute Affinity Matrix. Then it forms linearly connected tree & form cycles out of that [12]. Each cycle would be considered as a separate partition. For algorithm refer to section 4.2.

.

3.1 Definitions

- **Cycle completing edge** denotes edges that would complete a cycle.
- **Cycle connecting edge** denotes edges that connect two cycles or partitions.
- Every node in a cycle is called **Cycle node.**
- **End Nodes** are the end points of linearly connected tree. Only two end nodes exist in a linearly connected tree.

- When a cycle encloses any smaller cycle, then the smaller cycle is called **Enclosed Cycle** and bigger cycle is called **Enclosing cycle.** For Example, as shown in the figure 11 cycle ABCDA is enclosing cycles and the cycle BCDB is enclosed cycle.

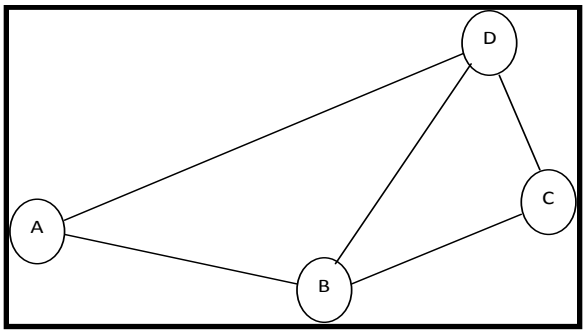


Figure 6 : Enclosed and Enclosing Cycle

3.2 Algorithm

First of all create affinity graph of the attributes. Note: that AAM is itself an adequate data structure to represent this graph. Algorithm for partitioning the graph by forming linearly connected tree and cycles later on is as follows:

1. Start from any node
2. Select an edge which satisfies the following conditions:
   a) Go to step 5 if all nodes are used for tree construction
   b) It should be linearly connected to the tree already constructed.
   c) It should have the largest value among the possible choices of edges at each end of the tree. Note that if there are several largest values, anyone can be selected.
3. When the next selected edge form a cycle and it does not connect to the cycle node of any cycle except the enclosed cycle.
   a) Mark this edge as cycle-completing edge but this edge is not considered as part of the tree and end nodes remain the same.
   b) If this cycle encloses any cycle whose cycle-completing edge have low or equal affinity than this edge then
      (i) remove the cycle-completing tag from the edge of enclosed cycle and
      (ii) remove the cycle separating tag from the edge that have it inside the enclosing cycle so formed.
   c) Go to step 2.
4. When the next edge selected does not form a cycle
   a) Change the end node to the next node to which this edge is connected.

b) If the first node of this edge is cycle node then mark this edge as cycle-connecting edge.
c) Go to step 2.

5. When all the nodes get traversed & become a part of the tree.
   a) Choose the non-selected edge from the any non-cycle node with the highest affinity & go to step 3.
   b) Then mark the selected-edge connecting this cycle with the rest of the graph as cycle-separating edge.
6. Separating out cycles.
   a) Choose the cycle completing-edge with highest affinity and complete the cycle. If the cycle so formed contains a cycle-separating edge then ignore this cycle. Keep doing this until we get all the nodes as part of some cycle.

3.3 Example

We will explain the application of this algorithm with the help of an example. First of all we consider an attribute usage matrix as shown in the table 1, which specifies which attributes are used in which transaction and frequency of transaction.

Table 1 : Attribute Usage Matrix

Trans \ Attr	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
T_1	35	0	0	0	35	0	35	0	0	0	35	35
T_2	0	50	50	0	0	0	0	0	50	0	0	50
T_3	0	0	25	25	0	25	0	0	0	25	0	0
T_4	0	35	0	0	0	0	35	0	35	0	35	0
T_5	25	25	25	0	0	0	25	25	25	0	0	25
T_6	25	0	25	0	25	0	0	25	0	0	25	0
T_7	0	0	25	0	0	0	0	25	0	0	0	25
T_8	0	0	15	15	0	15	0	0	15	15	15	0

Table 2 shows the attribute affinity matrix for above attribute usage matrix. Affinity of two attributes is the sum of frequencies of the transactions which access both of these two attributes. Attribute affinity matrix represents the affinity graph. We will apply modified graph partitioning algorithm on this graph and form subsets of attributes. Resultant subset of attributes defines the fragments.

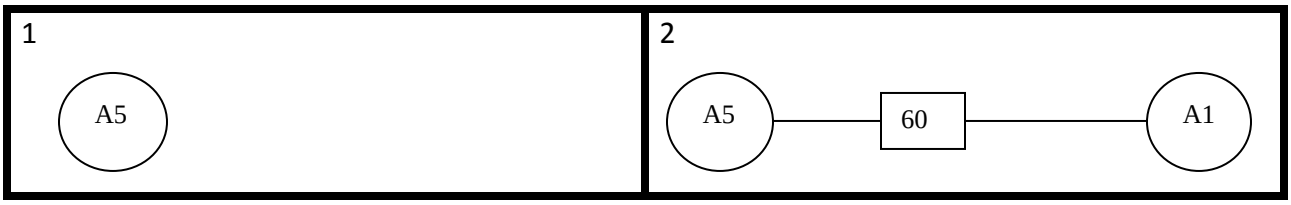
Table 2 : Attribute Affinity Matrix

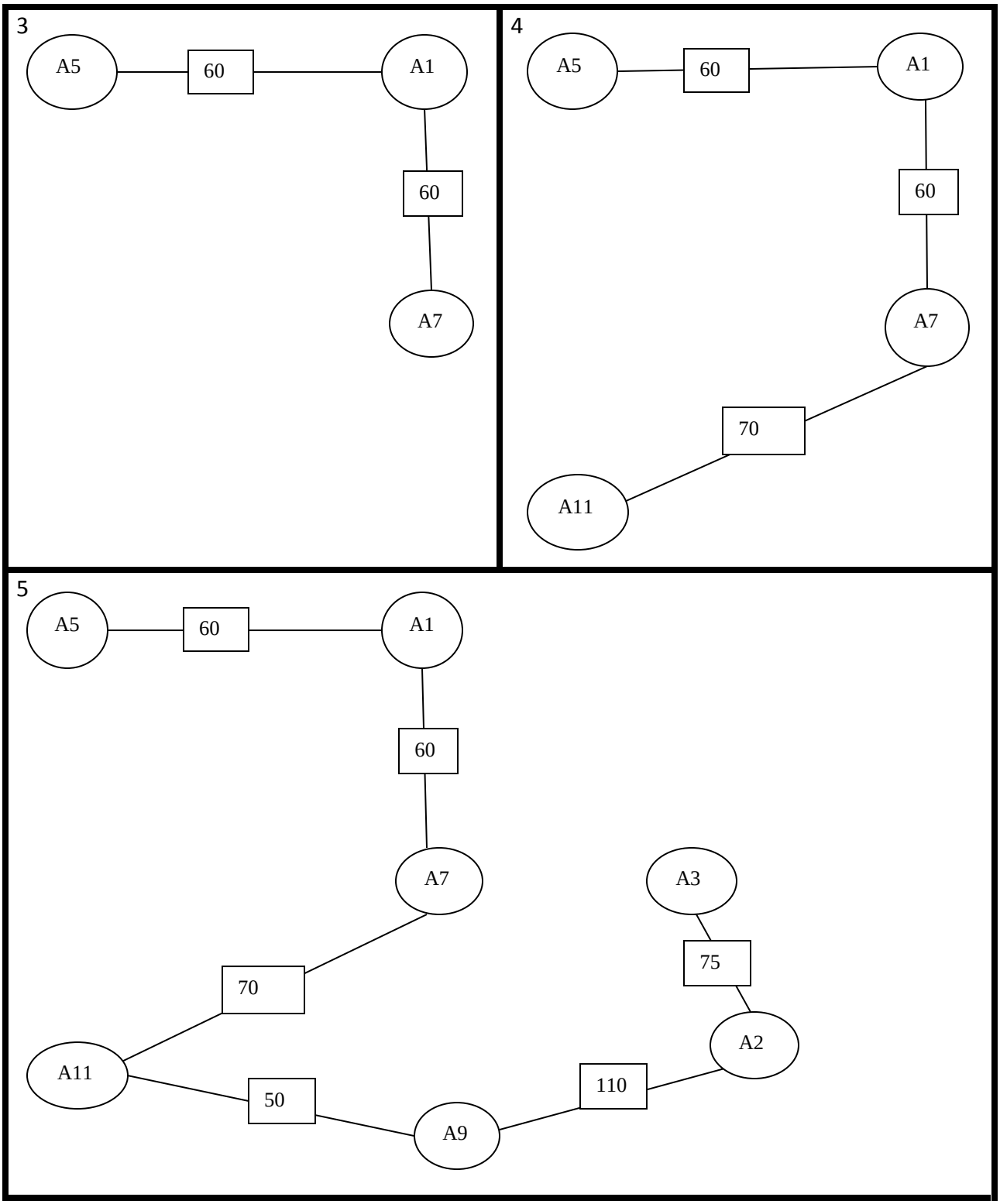
Attr\Attr	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
A_1	85	25	50	0	60	0	60	50	25	0	60	60
A_2	25	110	75	0	0	0	60	25	110	0	35	75
A_3	50	75	165	40	25	40	25	75	90	40	40	100
A_4	0	0	40	40	0	40	0	20	15	40	15	0
A_5	60	0	25	0	60	0	35	25	0	0	60	35
A_6	0	0	40	40	0	40	0	0	15	40	15	0
A_7	60	60	25	0	35	0	95	25	60	0	70	60
A_8	50	25	75	20	25	0	25	75	25	0	25	50
A_9	25	110	90	15	0	15	60	25	125	15	50	75
A_{10}	0	0	40	40	0	40	0	0	15	40	15	0
A_{11}	60	35	40	15	60	15	70	25	50	15	110	35
A_{12}	60	75	100	0	35	0	60	50	75	0	35	135

3.4 Step by Step Solution

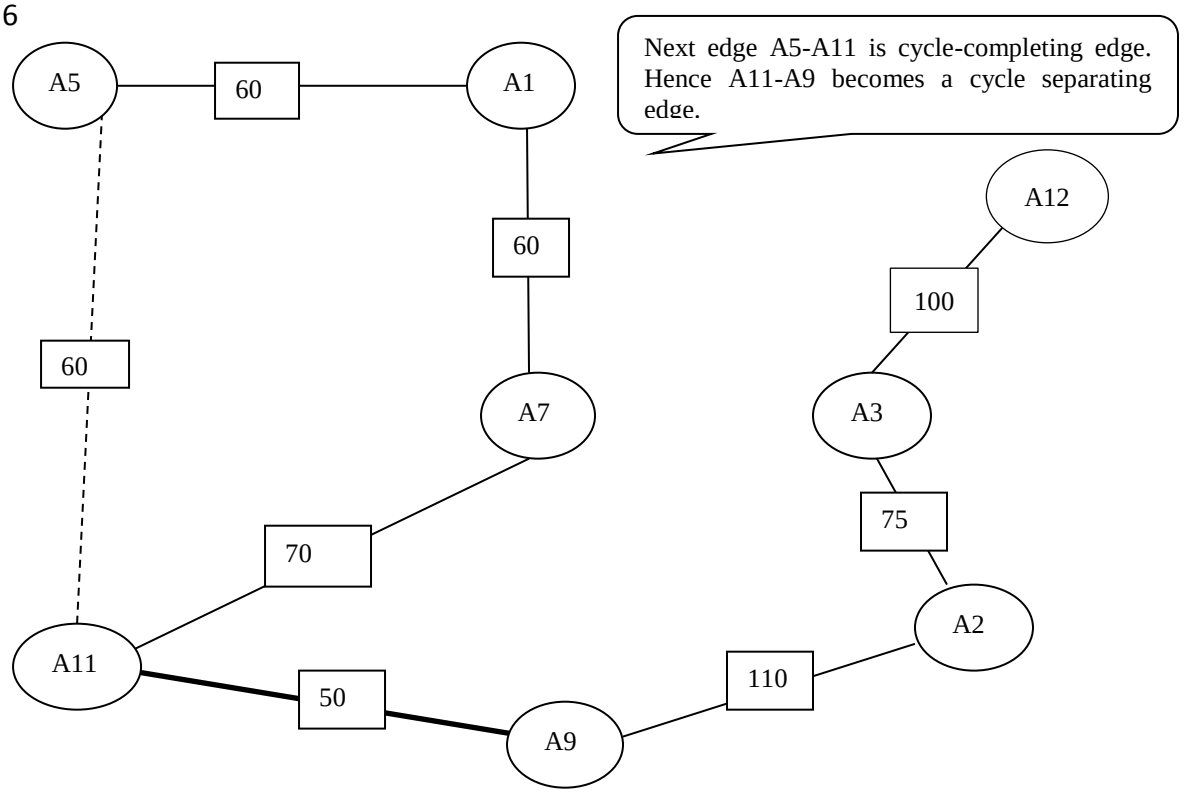
Now we will describe the complete solution of given example graphically using our modified graph-partitioning algorithm. Complete step by step solution is shown in the figure 17.

Figure 7 : Graphical Method for VF

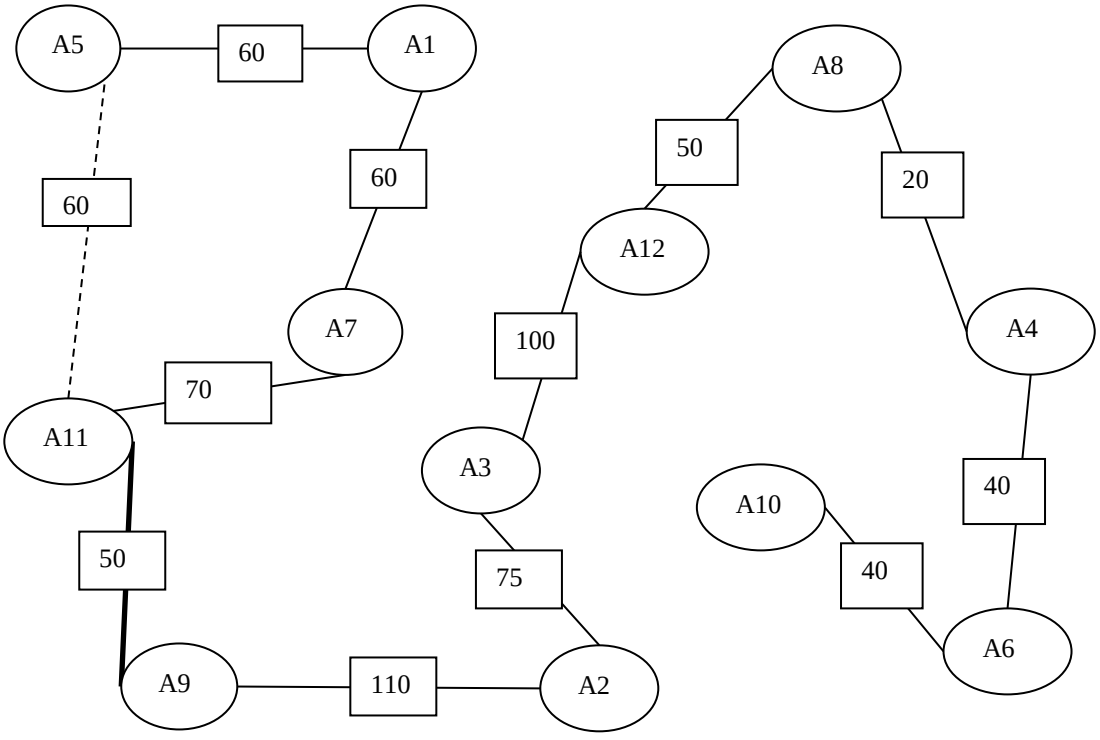



3
A5
60
A1
60
A7
4
A5
60
A1
60
A7
70
A11
5
A5
60
A1
60
A7
A3
75
A2
70
A11
50
A9
110

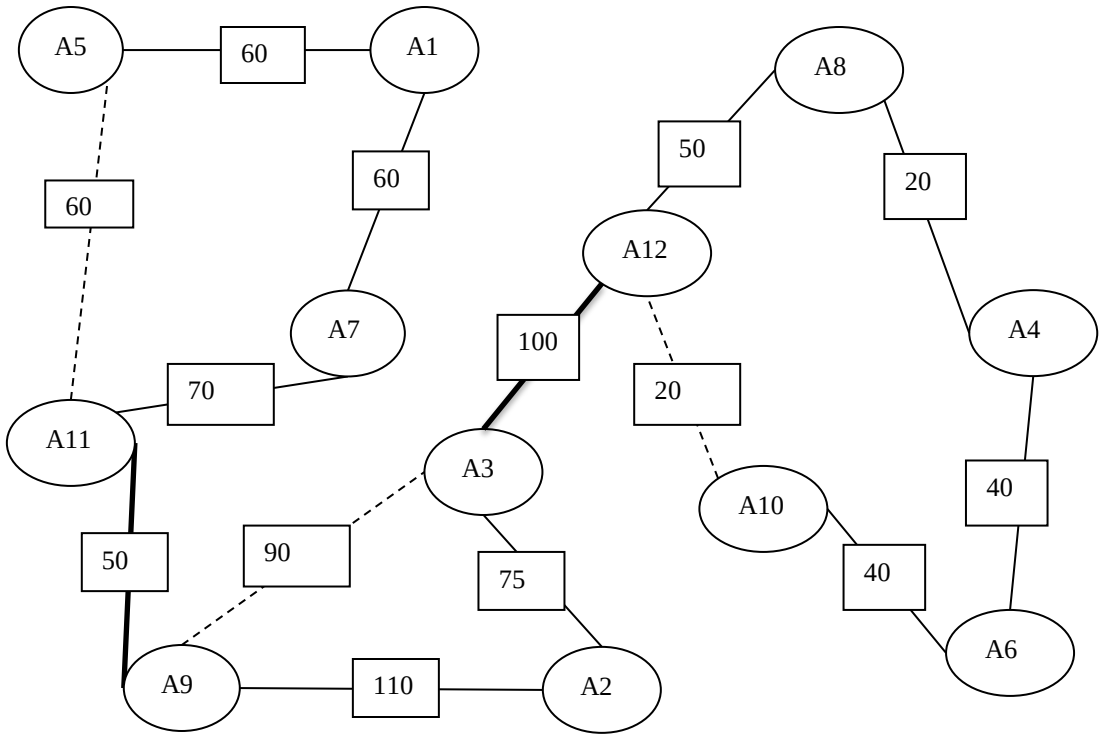
6


Next edge A5-A11 is cycle-completing edge. Hence A11-A9 becomes a cycle separating edge.
A5
60
A1
60
60
A7
A12
100
A3
75
A2
70
A11
50
A9
110

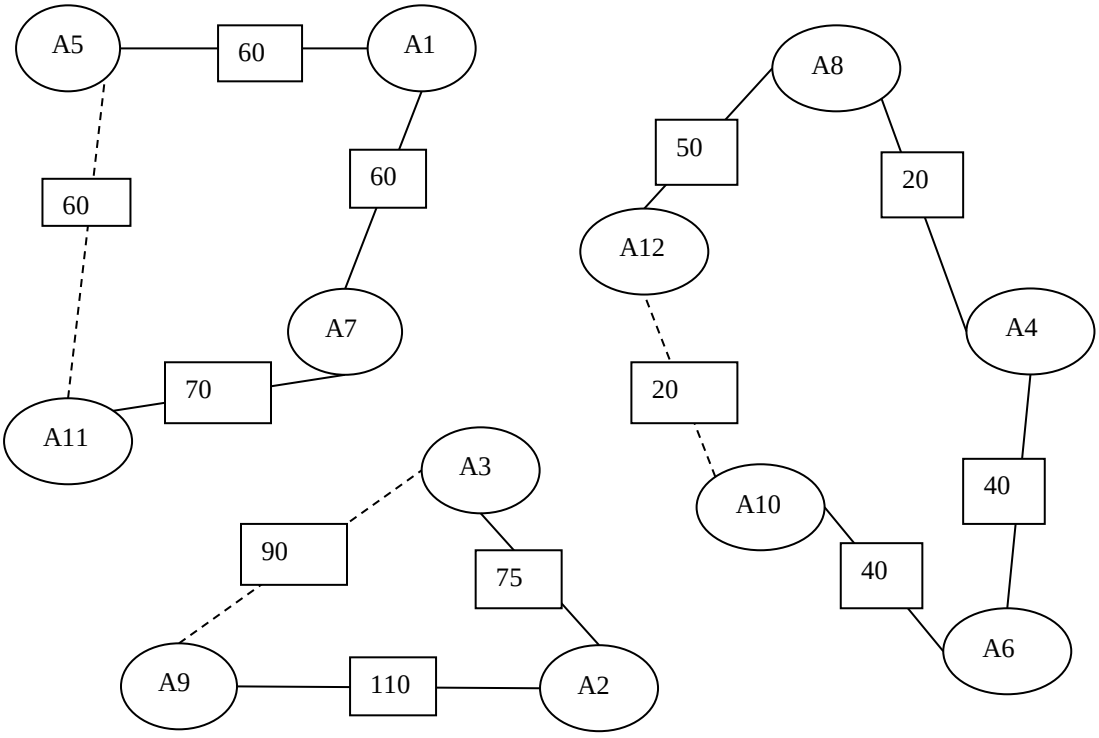
7


A5
60
A1
60
60
A7
A8
50
20
A12
A4
100
70
A11
A3
40
A10
50
75
40
A6
A9
110
A2

8


A5
60
A1
60
60
A7
70
A11
50
90
A9
110
A2
75
A3
100
A12
50
A8
20
20
A4
A10
40
40
A6

9


A5
60
A1
60
60
A7
70
A11
A3
90
75
A9
110
A2
A8
50
20
A12
20
A4
A10
40
40
A6

The resulting vertical fragments are:

1) (A_1, A_5, A_7, A_{11})
2) (A_2, A_3, A_9)
3) (A_4, A_6, A_8, A_{10}, A_{12})

3.5 Advantages

We summarize the major advantages of this method over the previous approaches:

1. There is no need for iterative binary partitioning. The major weakness of iterative binary partitioning discussed in the [2] is that at each step two new problems are generated increasing the complexity.
2. The complexity of the approach is $O(n^2)$ as opposed to $O(n^2\log(n))$ of the [2] approach.

3.6 Disadvantages

Although the overall process is very effective one but it also has some drawbacks. It suits centralized databases, as it does not address the problem of allocation of fragments to different sites. In our thesis we proposed cost model for the allocation of these fragments to overcome this disadvantage.

CHAPTER 4

Heuristic Approach for VF

In last section we discussed graphical partitioning algorithm for VF using attribute affinities. There are many other algorithms proposed in the literature using attribute affinities. Fragmentation, allocation and replication are database distribution design techniques that aim at improving the system performance. Among the two fragmentation techniques, vertical fragmentation is often considered more complicated than horizontal fragmentation, because the huge number of alternatives makes it nearly impossible to obtain an optimal solution to the vertical fragmentation problem. Therefore, we can only expect to find out a heuristic solution. Often fragmentation and allocation are considered separately, disregarding that they are using the same input information to achieve the same objective, i.e. improve the overall system performance. Here we discuss vertical fragmentation and allocation simultaneously in the context of the relational model. The core of our approach is a heuristic approach to vertical fragmentation, which uses a cost model and is targeted at globally minimizing these costs. This algorithm was proposed by Hui Ma, Klaus-Dieter Schewe and Markus Kirchberg in [4].

We will incorporate all query information, including the site information, by using a simplified cost model for VF. Doing this way, we can obtain vertical fragmentation and fragment allocation simultaneously with low computational complexity and resulting high system performance.

4.1 Notations and Definitions

We now define some terms that will be used in our discussion.

- Assume a relation $R = \{a_1, \ldots, a_n\}$ being accessed by a set of queries $Q_m = \{Q_1, \ldots, Q_j, \ldots Q_m\}$ with frequencies $f_1, \ldots, f_m$, respectively.
- To improve the system performance, relation R is vertically fragmented into a set of fragments $\{F_1, \ldots, F_u, \ldots, F_v\}$, each of which is allocated to one of the network nodes $N_1, \ldots, N_h, \ldots, N_k$. Note that the maximum number of fragments is k, i.e., $v \leq k$.
- We use $\lambda(Q_j)$ to indicate the site that issues query Q_j and use $A_j = \{a_i | f_{ji} = f_j\}$ to indicate the set of attributes that are accessed by Q_j, with f_{ji} as the frequency of the query Q_j accessing attributes a_i. Here, $f_{ji} = f_j$ if the attribute a_i is accessed by Q_j. Otherwise $f_{ji} = 0$.

Input to this cost model for optimal vertical fragmentation is:

- Frequency of queries that access the object. When the same query is issued at different sites it is treated as different queries.
- The subset of attributes used by queries.
- The size of each attribute of the object.
- The site that issue the queries.

To record the above input information we introduce Attribute Usage Frequency Matrix (AUFM) which is similar to AUM. Each row represents one query Q_j; the head of column is the set of attributes of a relation E. In addition, there are two columns with one column indicating the site that issues the queries and the other indicating the frequency of the queries. The values on a column indicate the frequency f_{ji} of the query Q_j that use the corresponding attributes a_i grouped by the site that issues queries. Note that we treat the same query at different sites as different queries. Doing this way we only need one matrix to record all the information rather than two matrices, Attribute Usage Matrix and Access Matrix that are used in our previous approach. Subsequently, the following up calculation is easy to be formulated.

From one site each attribute is requested by multiple queries. The request of an attribute at a site *h* is the sum of frequencies of all queries at the site *h* accessing the attribute. It can be calculated with the formula below:

$$request_h(a_i) = \sum_{j=1,\ \lambda(Qj)=h}^{m} f_{ji} \quad \text{.....(4.1)}$$

Let f_{ji} be the frequency of a query accessing an attribute a_i and l_i be the length of this attribute. The *need* of this attribute at a site *h* is calculated with the following formula:

$$need_h(a_i) = l_i * \sum_{j=1,\ \lambda(Qj)=h}^{m} f_{ji} \quad \text{.....(4.2)}$$

$$need_h(a_i) = l_i * request(a_i) \quad \text{.....(4.3)}$$

Finally, a term pay to measure the costs of allocating a single attribute to a network node. The pay of allocating an attribute a_i to a site h measures the costs of accessing attribute a_i from all queries at the other sites h` which is different from h. It can be calculated using the following formula:

$$pay_h(a_i) = \sum_{h`=1, h \neq h`}^{k} request_{h`}(a_i) * c_{hh`} \quad \text{.....(4.4)}$$

Note that the cost factor $c_{hh'} = 0$, if h = h'.

In distributed databases, costs of queries are dominated by the cost of data transportation from a remote site to the site that issues the queries. To compare different vertical fragmentation scheme we would like to compare how it affect the transportation costs.

Taking the simplified cost model we now analyze the relationships between *cost*, the *pay* and the *request*. We compute the following formulae:

$$cost_h(a_i) = \sum_{h`=1, h \neq h`}^{k} need_{h`}(a_i) * c_{hh`} \quad \text{.....(4.5)}$$

$$= l_i * \sum_{h`=1, h \neq h`}^{k} request_{h`}(a_i) * c_{hh`} \quad \text{.....(4.6)}$$

$$cost_h(a_i) = l_i * pay_h(a_i) \quad \text{.....(4.7)}$$

The above formula give rise to two alternative heuristics for the allocation of an attribute a_i (i = 1, . . . , n).

- The first heuristic allocates a_i to a network node N_w such that $pay_w(a_i)$ is minimal, i.e., we choose a network node in such a way that the total transport costs for all queries arising from the allocation are minimized.
- The second heuristic allocates a_i to a network node N_w such that $request_w(a_i)$ is maximal. i.e., we choose the network node with the highest request of the attribute a_i. This guarantees that there is no transportation cost associated with data of attribute a_i for those queries that need the data of a_i most frequently. In addition, the availability of data of attribute a_i will be maximized.

4.2 Algorithm

Taking the first heuristic we perform vertical fragmentation with the following steps. We do not distinguish read and write queries. Take the most frequently used 20% queries Q_n.

1. Optimize all the queries and construct an AUFM for each relation based on the queries.
2. Calculate the request at each site for each attribute to construct an Attribute request matrix using equation (4.1).
3. Calculate the pay at each site for each attribute to construct an attribute pay matrix using equation (4.4).
4. Cluster all attributes to the site which has the lowest value of the pay.
5. Attach the primary key to each fragment.

4.3 Example

We take the example problem used in previous chapter to illustrate how this approach works. Firstly, we take the Attribute Usage Matrix and Attribute Access Matrix to construct an AUFM grouped by site that issues the queries. The AUFM is shown in Table 3.

Table 3 : Attribute Usage Frequency Matrix

Site	Query	Frequency	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
1	T_1	10	10	0	0	0	10	0	10	0	0	0	10	10
1	T_2	10	0	10	10	0	0	0	0	0	10	0	0	10
1	T_4	10	0	10	0	0	0	0	10	0	10	0	10	0
1	T_6	10	10	0	10	0	10	0	0	10	0	0	10	0
1	T_5	5	5	5	5	0	0	0	5	5	5	0	0	5
1	T_7	5	0	0	5	0	0	0	0	5	0	0	0	5
1	T_8	5	0	0	5	5	0	5	0	0	5	5	5	0
2	T_2	20	0	20	20	0	0	0	0	0	20	0	0	20
2	T_1	15	25	0	0	0	25	0	25	0	0	0	25	25
2	T_5	10	10	10	10	0	0	0	10	10	10	0	0	10
2	T_7	10	0	0	10	0	0	0	0	10	0	0	0	10
2	T_6	5	5	0	5	0	5	0	0	5	0	0	5	0
2	T_8	5	0	0	5	5	0	5	0	0	5	5	5	0

3	T_3	15	0	0	15	15	0	15	0	0	0	15	0	0
3	T_4	15	0	15	0	0	0	0	15	0	15	0	15	0
3	T_2	10	0	10	10	0	0	0	0	0	10	0	0	10
3	T_5	5	5	5	5	0	0	0	5	5	5	0	0	5
3	T_6	5	5	0	5	0	5	0	0	5	0	0	5	0
3	T_7	5	0	0	5	0	0	0	0	5	0	0	0	5
3	T_8	3	0	0	3	3	0	3	0	0	3	3	3	0
4	T_2	10	0	10	10	0	0	0	0	0	10	0	0	10
4	T_3	10	0	0	10	10	0	10	0	0	0	10	0	0
4	T_4	10	0	10	0	0	0	0	10	0	0	0	10	0
4	T_5	5	5	5	5	0	0	0	5	5	5	0	0	5
4	T_6	5	5	0	5	0	5	0	0	5	0	0	5	0
4	T_7	5	0	0	5	0	0	0	0	5	0	0	0	5
4	T_8	2	0	0	2	2	0	2	0	0	2	2	2	0

Secondly, we compute the request using equation (4.1) for each attribute at each site and get the Attribute request Matrix shown in Table 4.

Table 4 : Attribute *request* Matrix

Site	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
1	25	25	25	5	25	5	25	30	20	5	35	30
2	30	30	45	5	40	5	25	40	35	5	25	55
3	10	30	23	18	10	18	20	35	18	18	23	20
4	10	25	22	12	10	12	15	30	17	12	17	20

Thirdly, assuming we have been given the values of transportation cost factors in Table 5. We can now calculate the pay of each attribute at each site by putting the values of the request given in Table 4 and values of cost factors given in Table 5 in equation (4.4).

Table 5 : Transportation Cost Factors

Site	1	2	3	4
1	0	10	25	20
2	10	0	20	15
3	25	20	0	15
4	20	15	15	0

The resultant attribute pay matrix is shown in Table 6.

Table 6 : Attribute *pay* Matrix

Attribute	1	2	3	4	5	6	7	8	9	10	11	12
$pay_1(a_i)$	950	1550	1685	610	1150	610	1100	1925	1310	610	1285	1800
$pay_2(a_i)$	850	1475	1290	640	850	640	1125	1750	1015	640	1415	1300
$pay_3(a_i)$	1125	1350	1605	355	1325	355	1100	1700	1255	355	1280	1850
$pay_4(a_i)$	1100	1400	1520	445	1250	445	1175	1725	1195	445	1420	1725

Finally, for each attribute we compare all the pay at all sites to find the minimal one. We subsequently allocate attribute a_i to the site with minimal pay. The allocation of attributes to sites is shown in Table 7.

Table 7 : Attribute Allocation

Attribute a_i	1	2	3	4	5	6	7	8	9	10	11	12
Site N_j	2	3	2	3	2	3	1	3	2	3	3	2

Therefore, relation R has been fragmented into two fragments with $F_1 = \{a_1, a_3, a_5, a_9, a_{12}\}$ $F_2 = \{a_2, a_4, a_6, a_8, a_{10}\}$ and $F_3 = \{a_7\}$, which have been allocated to site 2, 3 and 1 respectively.

4.4 Advantages of this Approach

The advantages of this heuristic approach for vertical fragmentation and allocation are:

- Except primary-key attributes, there is no overlap among all the vertical fragments. Therefore, we do not need extra procedure to remove overlaps.
- The change of queries will be reflected by the fragmentation solution. Query information may reflect the needs to retain attributes from some sites more often than some other sites. Even though on the affinity graph the cutting edges will be same.
- The complexity of this approach is low. Let m be the number of queries, n be the number of attributes, k be the number of network nodes. The complexity of our approach, which deals with vertical fragmentation and allocation, is $O(m * n + k^2 * n)$, while the complexity of graphical approach is $O(n^2 * m + k^n)$ for the whole design procedure, including building the affinity matrix, vertical fragmentation and allocation.
- This approach suits the situation that for each relation the number of attributes is small and the number of queries is big. Usually, the number of queries taken into consideration is bigger than the number of attributes of a relation.

CHAPTER 5

Cost based Graphical Partitioning

We have discussed two fragmentation techniques graphical approach and heuristic approach of vertical fragmentation in the previous chapters. We also discussed the problem of allocation in graphical approach. Here we will explain the proposed work of combining the two approaches of vertical partitioning namely graphical approach and heuristic approach, to solve the problem of allocation in graphical approach so that it can also work on the distributed databases.

As we have seen in the graphical approach that we can create the fragments by finding out the graph cycles. Then we will create the Fragment Request Matrix same as the Attribute Request Matrix by using the equations of heuristic approach. By using the Transportation Cost Factor Matrix we will compute the Fragment Pay Matrix as we computed the Attribute Pay Matrix using the equations of heuristic approach. Finally, for each fragment we compare all the pay at all sites to find the minimal one. We subsequently allocate fragment fi to the site with minimal pay in the Fragment Allocation.

5.1 Notations and Definitions

We will propose three new matrices for fragment based cost estimation, namely:

1. Fragment Request Matrix (FRM)
   It will contain the information about the fragments' requests made by the different sites, the total request made by one site for particular fragment.
2. Fragment Pay Matrix (FPM)
   It will contain the information, about the total pay associated with one particular fragment, if it is allocated to one particular site.
3. Fragment Allocation (FA)
   It will contain the site information on which one particular fragment will be allocated.

From one site each attribute is requested by multiple queries. The request of a fragment at a site *h* is the sum of frequencies of all queries at the site *h* accessing the attributes of that fragment. It can be calculated with the formula below:

$$request_h(fr_i) = \sum_{k=1}^{n} \sum_{j=1,\ \lambda(Qj)=h}^{m} f_{ji} \quad \text{.....(5.1)}$$

Let f_{ji} be the frequency of a query accessing an attribute a_i and l_i be the length of this attribute. The *need* of this fragment at a site *h* is calculated with the following formula:

$$need_h(f_i) = l_i * \sum_{k=1}^{n} \sum_{j=1,\ \lambda(Qj)=h}^{m} f_{ji} \quad \text{.....(5.2)}$$

$$need_h(f_i) = l_i * request(f_i) \quad \text{.....(5.3)}$$

Finally a term pay to measure the costs of allocating a single fragment to a network node. The pay of allocating a fragment fr_i to a site h measures the costs of accessing all of the attributes of that fragment fr_i from all queries at the other sites h` which is different from h. It can be calculated using the following formula:

$$pay_h(fr_i) = \sum_{k=1}^{n} \sum_{h`=1, h \neq h`}^{k} request_{h`}(ai) * c_{hh`} \quad \text{.....(5.4)}$$

Note that the cost factor $c_{hh'} = 0$, if h = h'.

In distributed databases, costs of queries are dominated by the cost of data transportation from a remote site to the site that issues the queries.

Taking the simplified cost model we now analyze the relationships between *cost*, the *pay* and the *request*. We compute the following formulae:

$$cost_h(fr_i) = \sum_{k=1}^{n} \sum_{h`=1, h \neq h`}^{k} need_{h`}(ai) * c_{hh`} \quad \text{.....(5.5)}$$

$$= l_i * \sum_{k=1}^{n} \sum_{h`=1, h \neq h`}^{k} request_{h`}(a_i) * c_{hh`} \quad \text{.....(5.6)}$$

$$cost_h(fr_i) = l_i * pay_h(fr_i) \quad \text{.....(5.7)}$$

All these formulae are the modified formulae of the heuristic approach to apply them on the fragments.

5.2 Proposed Algorithm:

1. Generate fragments using graphical approach.
2. Optimize all the queries and construct an Attribute Usage Frequency Matrix for each relation based on the queries.
3. Calculate the request at each site for each fragment to construct a Fragment Request Matrix using the equation 5.1.
4. Calculate the pay at each site for each fragment to construct a Fragment Pay Matrix using the equation 5.4.
5. Cluster all fragments to the site which has the lowest value of the pay.

5.3 Example

We will explain the application of this algorithm with the help of an example. First of all we consider an attribute usage matrix as shown in the table 8, which specifies which attributes are used in which transaction and frequency of transaction.

Table 8 : Attribute Usage Matrix

Trans \ Attr	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
T_1	35	0	0	0	35	0	35	0	0	0	35	35
T_2	0	50	50	0	0	0	0	0	50	0	0	50
T_3	0	0	25	25	0	25	0	0	0	25	0	0
T_4	0	35	0	0	0	0	35	0	35	0	35	0
T_5	25	25	25	0	0	0	25	25	25	0	0	25
T_6	25	0	25	0	25	0	0	25	0	0	25	0
T_7	0	0	25	0	0	0	0	25	0	0	0	25
T_8	0	0	15	15	0	15	0	0	15	15	15	0

Table 9 shows the attribute affinity matrix for above attribute usage matrix. Affinity of two attributes is the sum of frequencies of the transactions which access both of these two attributes. Attribute affinity matrix represents the affinity graph. We will apply graph partitioning algorithm on this graph and form subsets of attributes. Resultant subset of attributes defines the fragments.

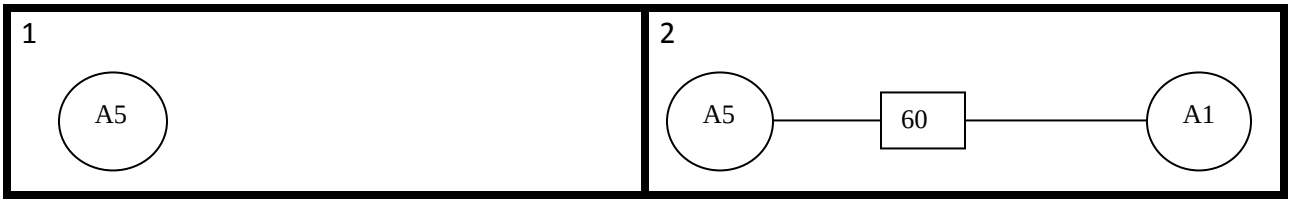
Table 9 : Attribute Affinity Matrix

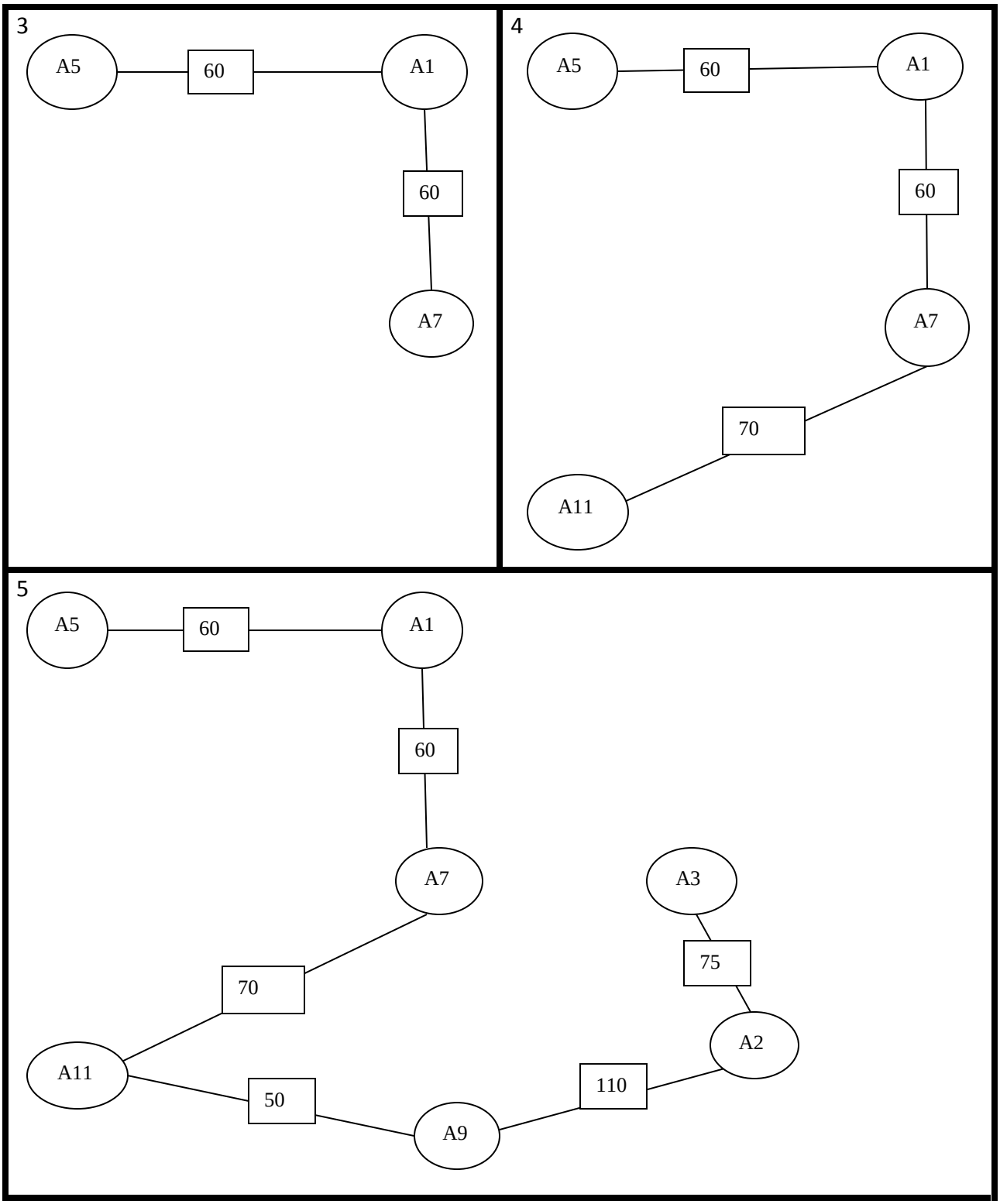
Attr\Attr	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
A_1	85	25	50	0	60	0	60	50	25	0	60	60
A_2	25	110	75	0	0	0	60	25	110	0	35	75
A_3	50	75	165	40	25	40	25	75	90	40	40	100
A_4	0	0	40	40	0	40	0	20	15	40	15	0
A_5	60	0	25	0	60	0	35	25	0	0	60	35
A_6	0	0	40	40	0	40	0	0	15	40	15	0
A_7	60	60	25	0	35	0	95	25	60	0	70	60
A_8	50	25	75	20	25	0	25	75	25	0	25	50
A_9	25	110	90	15	0	15	60	25	125	15	50	75
A_{10}	0	0	40	40	0	40	0	0	15	40	15	0
A_{11}	60	35	40	15	60	15	70	25	50	15	110	35
A_{12}	60	75	100	0	35	0	60	50	75	0	35	135

5.4 Step by Step Solution

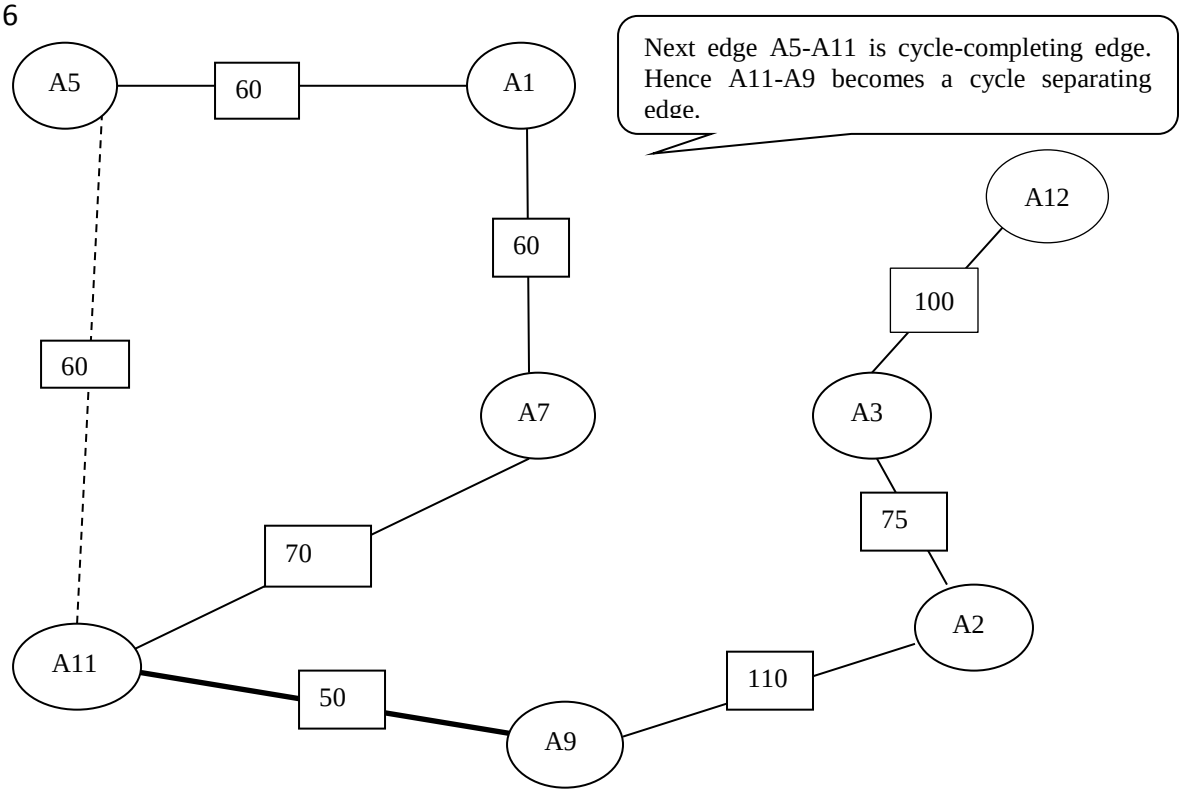
Now we will describe the complete solution of given example graphically using graph-partitioning algorithm. Complete step by step solution is shown in the figure 8.

Figure 8 : Graphical Method for VF

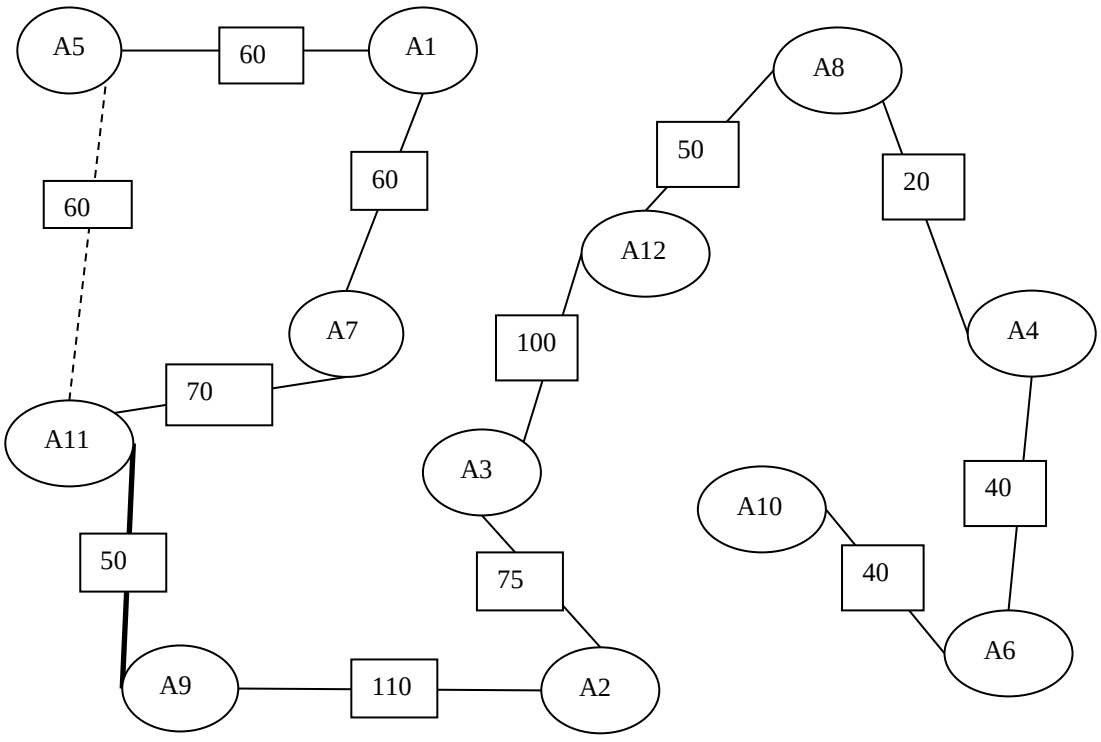



3
A5
60
A1
60
A7
4
A5
60
A1
60
A7
70
A11
5
A5
60
A1
60
A7
A3
75
A2
70
A11
50
A9
110

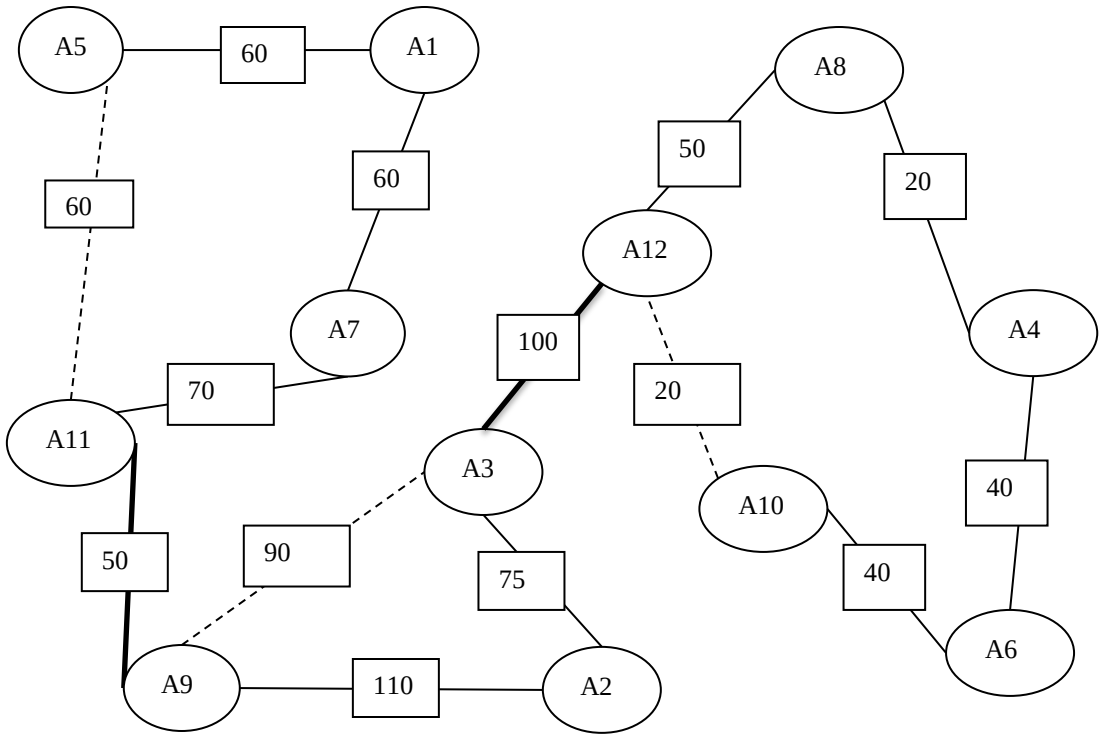
6


Next edge A5-A11 is cycle-completing edge. Hence A11-A9 becomes a cycle separating edge.
A5
60
A1
60
A12
100
60
A7
A3
75
70
A2
A11
50
110
A9

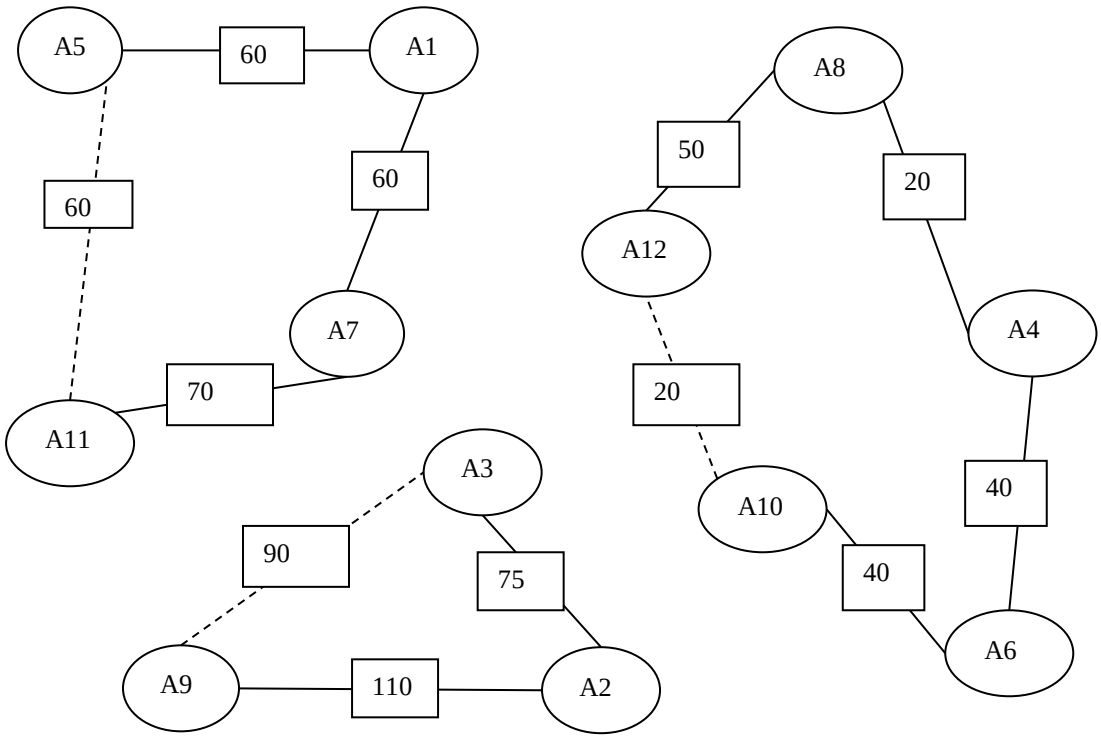
7


A5
60
A1
A8
50
60
20
60
A12
A7
A4
100
70
A11
A3
40
A10
50
75
40
A9
110
A2
A6

8


A5
60
A1
60
60
A7
70
A11
50
90
A9
110
A2
75
A3
100
A12
50
A8
20
20
A4
A10
40
40
A6

9


A5
60
A1
60
60
A7
70
A11
A3
90
A9
110
A2
75
A12
50
A8
20
20
A4
A10
40
40
A6

The resulting vertical fragments are:

1) (A1, A5, A7, A11)
2) (A_2, A_3, A_9)
3) (A_4, A_6, A_8, A_{10}, A_{12})

Table 10 : Attribute Usage Frequency Matrix

Site	Query	Frequency	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
1	T_1	10	10	0	0	0	10	0	10	0	0	0	10	10
1	T_2	10	0	10	10	0	0	0	0	0	10	0	0	10
1	T_4	10	0	10	0	0	0	0	10	0	10	0	10	0
1	T_6	10	10	0	10	0	10	0	0	10	0	0	10	0
1	T_5	5	5	5	5	0	0	0	5	5	5	0	0	5
1	T_7	5	0	0	5	0	0	0	0	5	0	0	0	5
1	T_8	5	0	0	5	5	0	5	0	0	5	5	5	0
2	T_2	20	0	20	20	0	0	0	0	0	20	0	0	20
2	T_1	15	25	0	0	0	25	0	25	0	0	0	25	25
2	T_5	10	10	10	10	0	0	0	10	10	10	0	0	10
2	T_7	10	0	0	10	0	0	0	0	10	0	0	0	10
2	T_6	5	5	0	5	0	5	0	0	5	0	0	5	0
2	T_8	5	0	0	5	5	0	5	0	0	5	5	5	0
3	T_3	15	0	0	15	15	0	15	0	0	0	15	0	0
3	T_4	15	0	15	0	0	0	0	15	0	15	0	15	0
3	T_2	10	0	10	10	0	0	0	0	0	10	0	0	10
3	T_5	5	5	5	5	0	0	0	5	5	5	0	0	5
3	T_6	5	5	0	5	0	5	0	0	5	0	0	5	0
3	T_7	5	0	0	5	0	0	0	0	5	0	0	0	5
3	T_8	3	0	0	3	3	0	3	0	0	3	3	3	0
4	T_2	10	0	10	10	0	0	0	0	0	10	0	0	10
4	T_3	10	0	0	10	10	0	10	0	0	0	10	0	0
4	T_4	10	0	10	0	0	0	0	10	0	0	0	10	0
4	T_5	5	5	5	5	0	0	0	5	5	5	0	0	5
4	T_6	5	5	0	5	0	5	0	0	5	0	0	5	0
4	T_7	5	0	0	5	0	0	0	0	5	0	0	0	5
4	T_8	2	0	0	2	2	0	2	0	0	2	2	2	0

Secondly, we compute the request using equation (5.1) for each fragment at each site and compute the Fragment Request Matrix shown in Table 11.

Table 11: Fragment Request Matrix

Site	F_1	F_2	F_3
1	110	70	75
2	120	110	110
3	63	71	109
4	52	64	86

Thirdly, assuming we have been given the values of transportation cost factors in Table 12. We can now calculate the pay of each fragment at each site by putting the values of the request given in Table 11 and values of cost factors given in Table 12 in equation (5.4).

Table 12: Transportation Cost Factors

Site	1	2	3	4
1	0	10	25	20
2	10	0	20	15
3	25	20	0	15
4	20	15	15	0

The resultant Fragment Pay Matrix is shown in Table 13.

Table 13: Fragment Pay Matrix

Fragment	1	2	3
pay1(fi)	4385	4545	5555
pay2(fi)	4240	3780	4970
pay3(fi)	4830	4210	4615
pay4(fi)	4945	4115	4785

Finally, for each fragment we compare all the pay at all sites to find the minimal one. We subsequently allocate fragment F_i to the site with minimal pay. The allocation of fragments to sites is shown in Table 14.

Table 14: Fragment Allocation

Fragment F_i	1	2	3
Site N_j	2	2	3

Therefore, the fragments $F_1 = \{a_1, a_5, a_7, a_{11}\}$ $F_2 = \{a_2, a_3, a_9\}$ and $F_3 = \{a_4, a_6, a_8, a_{10}, a_{12}\}$ have been allocated to site 2, 2 and 3 respectively.

From the example we have seen that we got the fragments using the graphical partitioning approach, then we applied the cost based model on these fragments according to the heuristic approach and computed the terms using the modified formulae of request, cost and pay, and then we allocated the fragments using the pay matrix.

This is how we allocated the fragments computed by the graphical approach to the site where the cost is lesser than the other sites; cost is nothing but the transaction delay.

CHAPTER 6

Implementation

In this chapter we are showing the screen shots of our tool, which implements the approaches described in previous chapters. We are using the same example, as we used in the previous chapters, to run the tool.

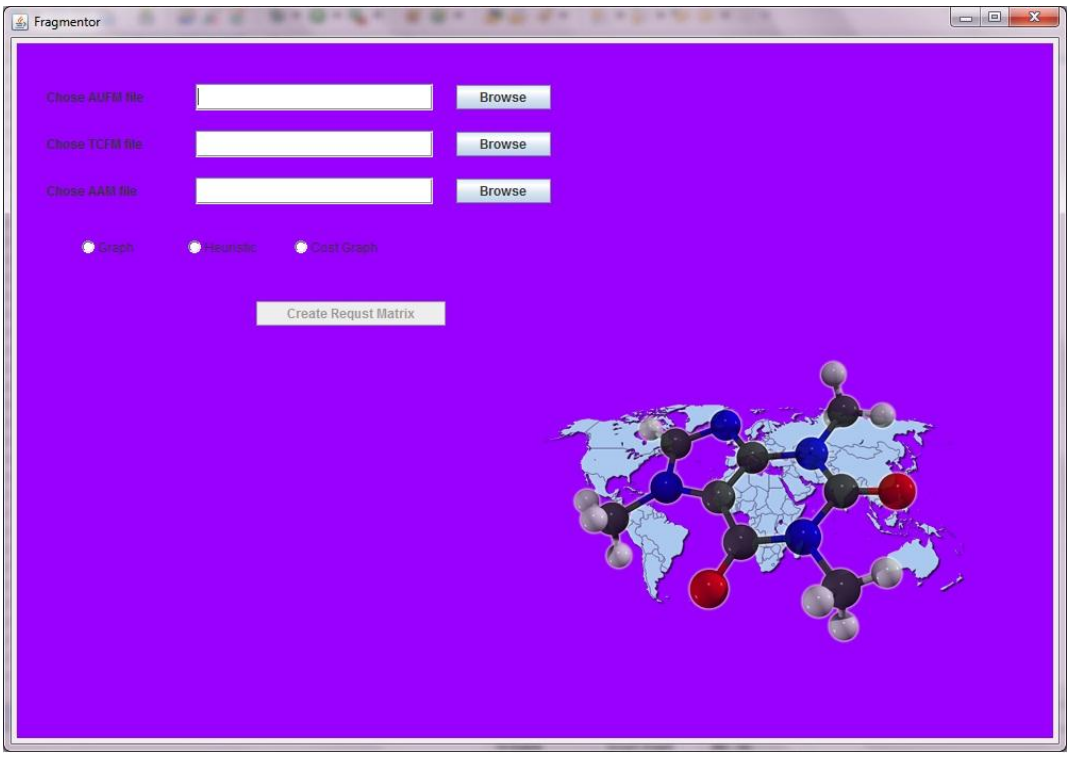


Figure: 9 Home screen

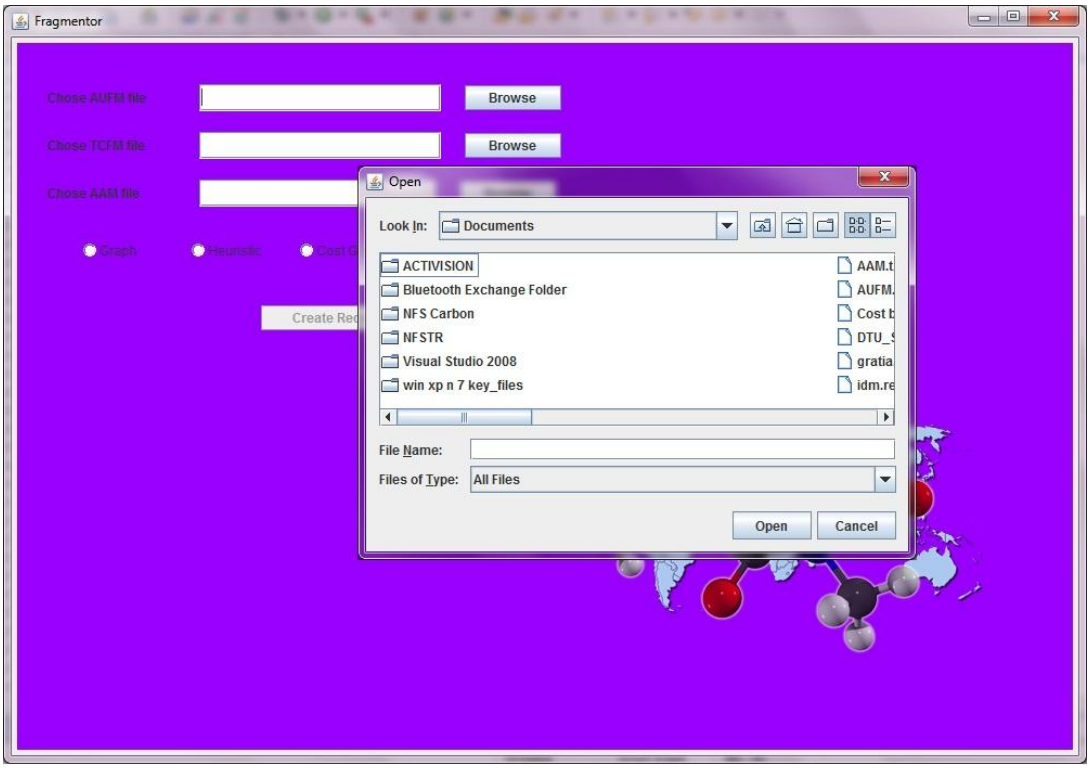


Figure: 10 Select matrices

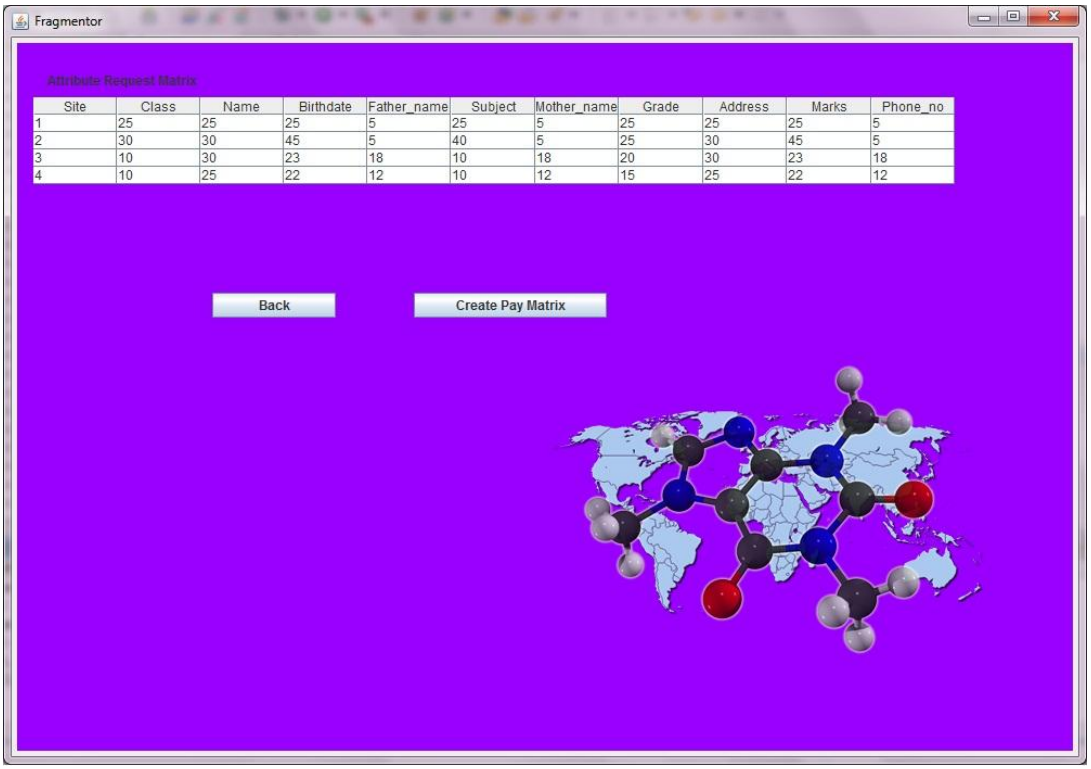


Figure: 11 Computed attribute request matrix

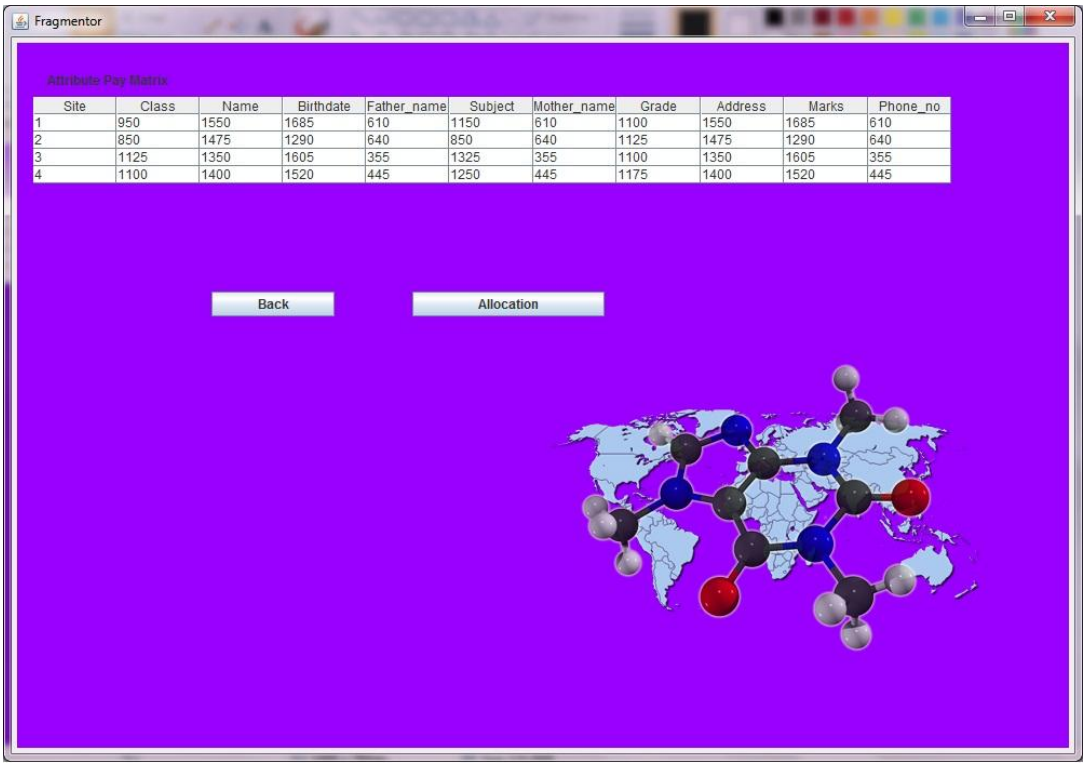


Figure: 12 Computed attribute pay matrix

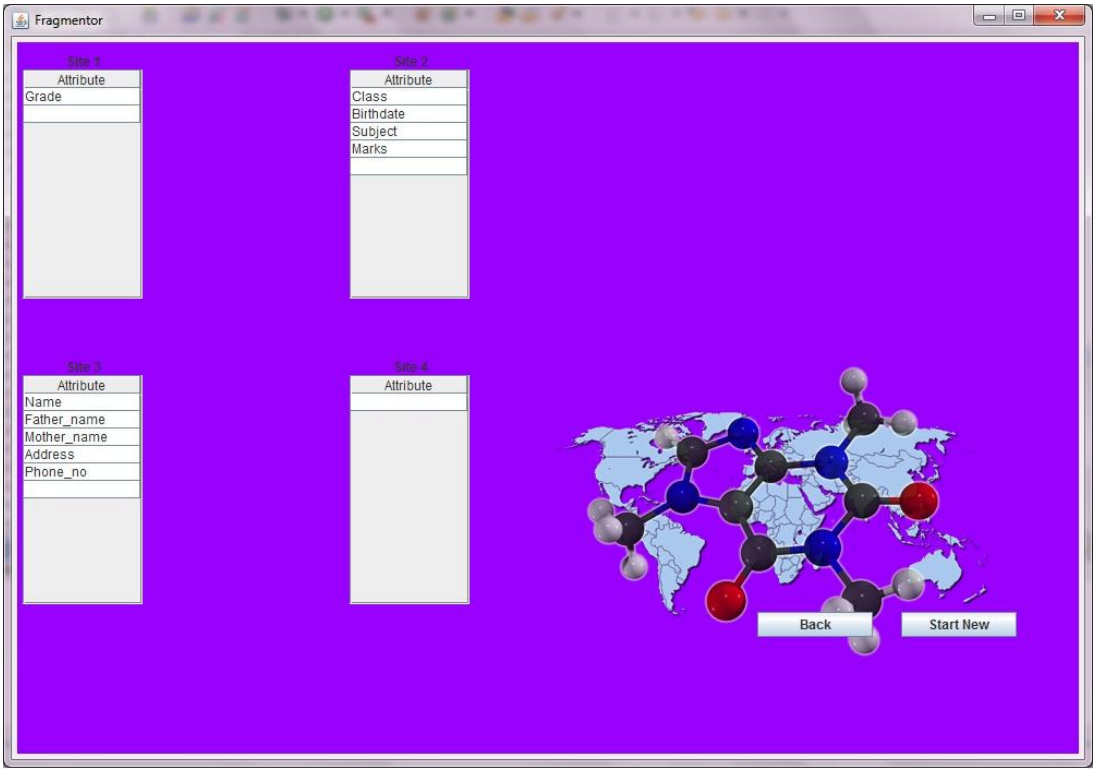


Figure: 13 Allocation of attributes

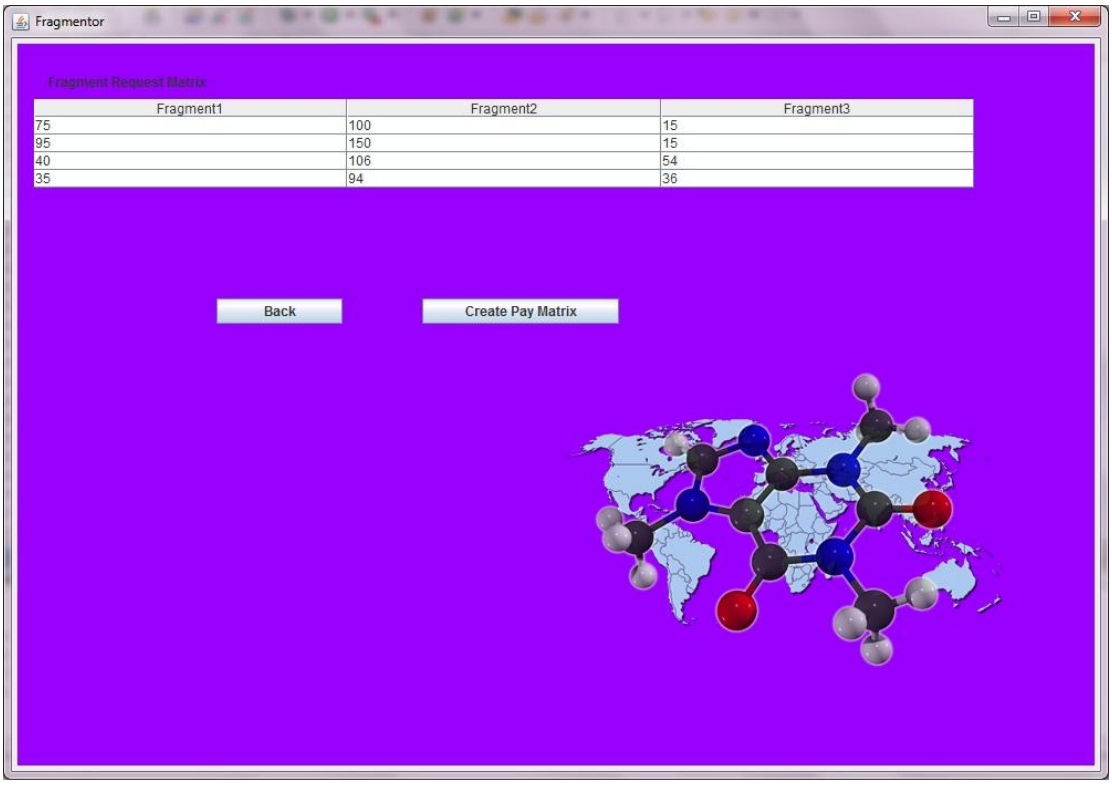


Figure: 14 Computed fragment request matrix

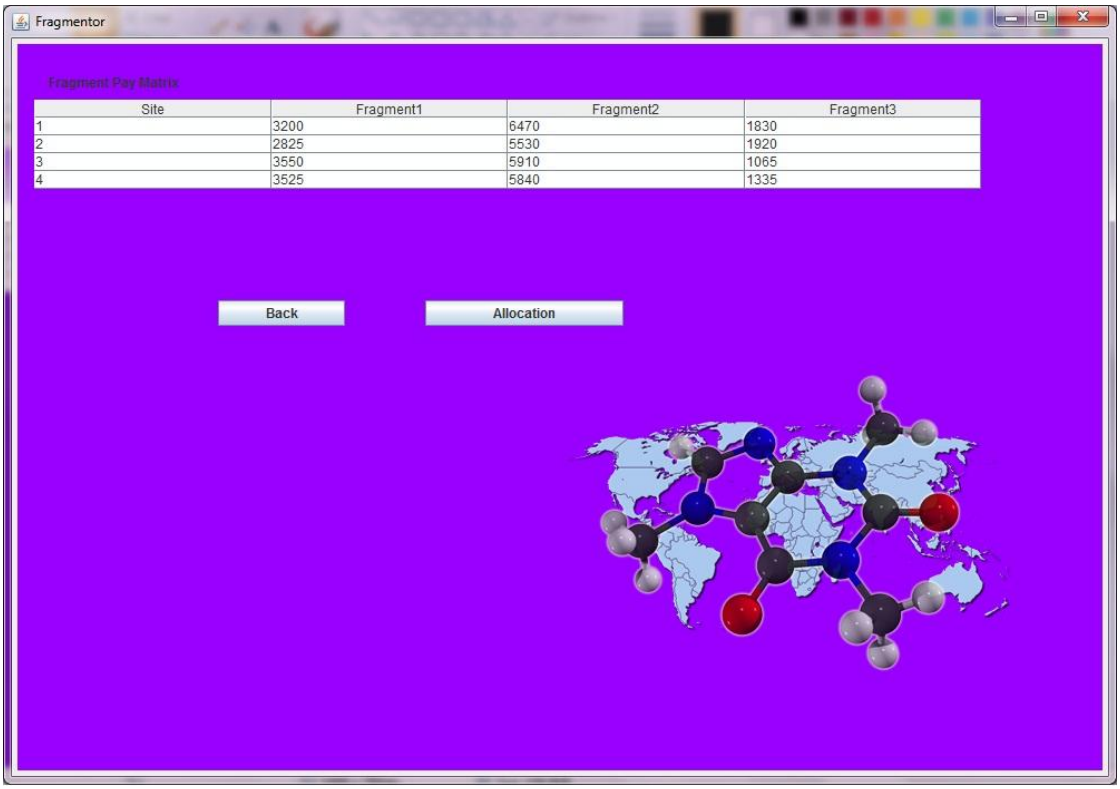


Figure: 15 Computed fragment pay matrix

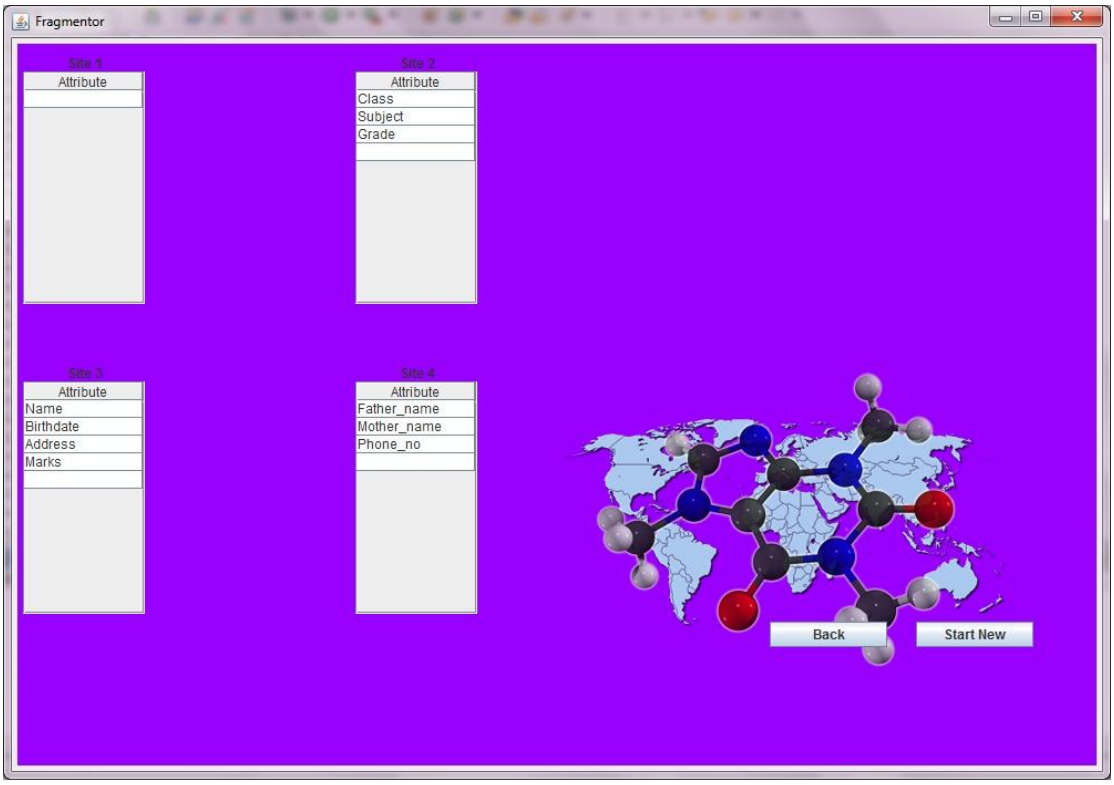


Figure: 16 Allocation of fragments

CHAPTER 7

Conclusion & Future Work

As part of this chapter we conclude our research work with a short summary and also discuss the future scope of this research work.

7.1 Conclusion

In this research work we have proposed an enhancement to the graphical fragmentation approach using the modified cost model of heuristic approach for vertical fragmentation. We also discussed various other fragmentation techniques. The major feature of the proposed approach is that it incorporates fragmentation of relation on the basis of actual cost of fragmentation. It takes frequency of transaction as input. Secondly attribute locality precedence is calculated according to actual frequency of queries.

We also implemented the tool for the proposed methodology, heuristic approach and graphical approach.

By comparing the allocation result of both the approaches heuristic and our approach for the example we considered, we saw that the cost of the queries after the allocation is 91550 and 89575 respectively. And for the example of [2] the cost after allocation is same.

Many techniques have been proposed by the researchers using empirical knowledge of data access and allocation. Most of those techniques are either HF or VF. But they mostly talk about the fragmentation not the allocation. Few approaches talk about the allocation, like optimal binary partitioning algorithm [7], which has the complexity O (2^n) while the attribute based algorithm has the complexity O (2^m), where n and m are the number of transactions and attributes in the system.

Complexity of bond energy algorithm [8] is O ($n^2 + 2^n$). Complexity of graphical approach [2] with the multi_allocate_n and multi_allocate_r approaches is O ($n^2 * m + k^n$) where k is the no of network nodes. While the complexity of our approach is O ($n^2 + m*n + k^2 * n$), which is better than the previous approaches. In our approach we have the advantages of graphical and heuristic approach. To the best of my knowledge no such cost based graphical fragmentation technique exists.

Using our technique the complexity will be low for allocating the fragments to the site of a distributed database as fragments are allocated according to their cost and request at the site. Our technique improves the DDBMS performance significantly by avoiding frequent remote access and high data transfer.

7.2 Future work

Further extension of our research is in the direction of horizontal fragments allocation and grid fragmentation and allocation.

Monitoring the effect of change in transaction and change in data over the fragments and processing of queries. Timing of re-fragmentation is next major decision that depends upon the change in transaction frequency.

Implementing the other approaches into our tool would be a feature enhancement to our tool, and we can also show the comparison result of all the approaches for better judgment.

This research can be extended to support fragmentation in distributed object oriented databases as well.

References

[1] S. B. Navathe and M. Ra, "Vertical partitioning for database design: A graphical algorithm" in Proceedings of ACM SIGMOD International Conference on Management of Data, Vol. 14, No. 4, pp. 440-450, 1989.

[2] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. "Vertical Partitioning Algorithms for Database Design" ACM Transactions on Database Systems, Vol. 9, No. 4, Dec. 1984.

[3] Katja Hose, Ralf Schenkel "Distributed Database Systems, Fragmentation and Allocation," Max-Planck-Institute for Informatik, Cluster of Excellence MMCI. October 28, 2010.

[4] Hui Ma, Klaus-Dieter Schewe and Markus Kirchberg, "A Heuristic Approach to Vertical Fragmentation Incorporating Query Information" in Proc. 7th International Baltic Conference on Databases and Information Systems (DB & IS), pp 69-76, 2006.

[5] J. Muthuraj, S.Chakravarthy, R. Varadarajan, and S.B.Navathe, "A formal approach to the vertical partitioning problem in distributed databases design," in Proc. of Second International Conference on Parallel and Distributed Information Systems, San Diego, California, 1993.

[6] E. S. Abuelyaman, "An optimized scheme for vertical partitioning of a distributed database," Int. Journal of Computer Science & Network Security, Vol. 8, No. 1, 2008.

[7] B. Niamir, "Attribute partitioning in a Self-Adaptive Relational Database System", PhD Dissertation, MIT Lab. for Computer Science, Jan 1978.

[8] W. McCormick, P. Schweitzer and T. White, "Problem Decomposition and Data Reorganization by a Clustering technique *Operations Research*," 20, Sep. 1972.

[9] http://en.wikipedia.org/wiki/Distributed_database

Source code of the tool

```
package tables;

import java.io.BufferedReader;
import java.io.File;

import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.StringTokenizer;

public class TableCreator {

  private ArrayList<ArrayList<Integer>> aufm;
  private ArrayList<ArrayList<Integer>> arm;
  private ArrayList<ArrayList<Integer>> tcfm;
  private ArrayList<ArrayList<Integer>> apm;
  private ArrayList<ArrayList<Integer>> aam;
  private ArrayList<ArrayList<Integer>> frags;
  private ArrayList<Integer> aa;
  private ArrayList<ArrayList<Integer>> fpm;
  private ArrayList<ArrayList<Integer>> frm;
  private ArrayList<Integer> fa;

  public TableCreator() {
    // TODO Auto-generated constructor stub
    aufm = new ArrayList<ArrayList<Integer>>();
    arm = new ArrayList<ArrayList<Integer>>();
    tcfm = new ArrayList<ArrayList<Integer>>();
    apm = new ArrayList<ArrayList<Integer>>();
    aam = new ArrayList<ArrayList<Integer>>();
    aa = new ArrayList<Integer>();
    frags = new ArrayList<ArrayList<Integer>>();
    fpm = new ArrayList<ArrayList<Integer>>();
    frm = new ArrayList<ArrayList<Integer>>();
    fa = new ArrayList<Integer>();
```

```
}

public ArrayList<String> createAUFM(File input) throws IOException {
  BufferedReader br = new BufferedReader(new FileReader(input));
  String row = null;
  ArrayList<String> column=new ArrayList<String>();

  if(!aufm.isEmpty())
  {
    aufm.removeAll(aufm);
  }

  row=br.readLine();
  StringTokenizer st1 = new StringTokenizer(row, " ");
  while (st1.hasMoreTokens()) {
    column.add(st1.nextToken());

  }

  while ((row = br.readLine()) != null) {
    StringTokenizer st = new StringTokenizer(row, " ");
    ArrayList<Integer> temp = new ArrayList<Integer>();
    while (st.hasMoreTokens()) {
      temp.add(Integer.parseInt(st.nextToken()));

    }
    aufm.add(temp);
  }

  br.close();

  return column;
}

public void showAUFM() {
  for (int j = 0; j < aufm.size(); j++) {
    System.out.println(aufm.get(j).toString());
  }
}
```

```
public Object[][] createARM() {


  arm.removeAll(arm);

  int rowno = 0;
  for (int j = 0; j < aufm.size(); j++) {
    if (aufm.get(j).get(0) != rowno) {
      arm.add(aufm.get(j));
      rowno++;
    } else {

      for (int i = 1; i < aufm.get(j).size(); i++) {
        arm.get(rowno - 1).set(i,
            arm.get(rowno - 1).get(i) + aufm.get(j).get(i));
      }
    }

  }

  Object[][] objARMTemp = new Object[arm.size()][arm.get(0).size()];

  for (int i = 0; i < arm.size(); i++) {
    for (int j = 0; j < arm.get(0).size(); j++) {
      objARMTemp[i][j]=arm.get(i).get(j);
    }
  }
  return objARMTemp;
}

public void showARM() {
  for (int j = 0; j < arm.size(); j++) {
    System.out.println(arm.get(j).toString());
  }
}

public void createTCFM(File input) throws IOException {
  BufferedReader br = new BufferedReader(new FileReader(input));
  String row = null;
```

```
  if(!tcfm.isEmpty())
  {
    tcfm.removeAll(tcfm);
  }
  while ((row = br.readLine()) != null) {
    StringTokenizer st = new StringTokenizer(row, " ");
    ArrayList<Integer> temp = new ArrayList<Integer>();
    while (st.hasMoreTokens()) {
      temp.add(Integer.parseInt(st.nextToken()));

    }
    tcfm.add(temp);
  }
  br.close();
}

public void showTCFM() {
  for (int j = 0; j < tcfm.size(); j++) {
    System.out.println(tcfm.get(j).toString());
  }
}

public Object[][] createAPM() {

  if(!apm.isEmpty())
  {
    apm.removeAll(apm);
  }

  for (int j = 0; j < tcfm.size(); j++) {
    ArrayList<Integer> temp = new ArrayList<Integer>();

    temp.add(j+1);
    for (int i = 1; i < arm.get(0).size(); i++) {
      int sum = 0;
      for (int k = 0; k < tcfm.size(); k++) {
        sum += tcfm.get(j).get(k) * arm.get(k).get(i);
      }
      temp.add(sum);
    }
```

```
    apm.add(temp);
  }

  Object[][] objARMTemp = new Object[arm.size()][arm.get(0).size()];

  for (int i = 0; i < apm.size(); i++) {
    for (int j = 0; j < apm.get(0).size(); j++) {
      objARMTemp[i][j]=apm.get(i).get(j);
    }
  }
  return objARMTemp;

}

public void showAPM() {
  for (int j = 0; j < apm.size(); j++) {
    System.out.println(apm.get(j).toString());
  }
}

public ArrayList<Integer> attributeAllocation() {

  if(!aa.isEmpty())
  {
    aa.removeAll(aa);
  }

  for (int i = 1; i < apm.get(0).size(); i++) {
    int min = 0, val = 99999;
    for (int j = 0; j < apm.size(); j++) {
      int temp = apm.get(j).get(i);
      if (temp < val) {
        min = j + 1;
        val = temp;
      }

    }
    aa.add(min);
  }
```

```
  return aa;
}

public void showAA() {

  System.out.println(aa.toString());

}

public void createAAM(File input) throws IOException {
  BufferedReader br = new BufferedReader(new FileReader(input));
  String row = null;
  if(!aam.isEmpty())
  {
    aam.removeAll(aam);
  }
  while ((row = br.readLine()) != null) {
    StringTokenizer st = new StringTokenizer(row, " ");
    ArrayList<Integer> temp = new ArrayList<Integer>();
    while (st.hasMoreTokens()) {
      temp.add(Integer.parseInt(st.nextToken()));

    }
    aam.add(temp);
  }
  br.close();
}

public void showAAM() {
  for (int j = 0; j < aam.size(); j++) {
    System.out.println(aam.get(j).toString());
  }
}

public void createGraph() {
  int a=4;
  int arr[]= new int[aam.size()];
  arr[0]=4;
  HashMap<Integer, Integer> hm=new HashMap<Integer, Integer>();
  hm.put(a, a);
```

```
for(int i=1;i<aam.size();i++)
{
  System.out.println("i="+i);
  int max=0;
  for (int j = 0; j < aam.get(0).size(); j++) {
    if(hm.containsKey(j))
    {
      if(max==j)
        max++;
      continue;
    }
    System.out.println("j="+j);
    System.out.println("max="+aam.get(a).get(max));
    System.out.println("getj="+aam.get(a).get(j));

    if(aam.get(a).get(max)<aam.get(a).get(j))
      max=j;
  }
  hm.put(max, max);
  arr[i]=max;
  a=max;
  System.out.println(max);
}

for (int i = 0; i < arr.length; i++) {
  System.out.println(arr[i]+1);
}

for (int i = 0,k=0; i < arr.length;k++) {
  int max=i+2;
  ArrayList<Integer> temp = new ArrayList<Integer>();
  for (int j = i+3; j < arr.length; j++) {
    if(aam.get(arr[i]).get(arr[max])<=aam.get(arr[i]).get(arr[j]))
      max=j;
  }
  System.out.println("max="+max+" i="+i);
  for (int j = i; j <= max; j++) {
    System.out.print(" "+(arr[j]+1));
    temp.add(arr[j]);
```

```
    }
    System.out.println("temp="+temp.get(0).toString());
    frags.add(temp);
    i=max+1;
  }
  System.out.println(frags.get(0).toString());
}

public Object[][] createFRM() {

  if(!frm.isEmpty())
  {
    frm.removeAll(frm);
  }

  Object[][] objARMTemp = new Object[arm.size()][frags.size()];

  for (int i = 0; i < frags.size(); i++) {
    for (int j = 0; j < arm.size(); j++) {
      int total=0;
      for (int j2 = 0; j2 < frags.get(i).size(); j2++) {
        total+=arm.get(j).get(frags.get(i).get(j2)+1);
      }
      objARMTemp[j][i]=total;
      System.out.println("Total="+total);
    }
  }
  System.out.println();
  for (int i = 0; i < arm.size(); i++) {
    System.out.println();
    ArrayList<Integer> temp=new ArrayList<Integer>();
    temp.add(i+1);
    for (int j = 0; j < frags.size(); j++) {
      System.out.print(objARMTemp[i][j]+"  ");
      temp.add(Integer.parseInt(objARMTemp[i][j].toString()));
    }
    frm.add(temp);
  }
  return objARMTemp;
```

```
}

public void showFRM() {
  for (int j = 0; j < frm.size(); j++) {
    System.out.println(frm.get(j).toString());
  }
}

public Object[][] createFPM() {


  if(!fpm.isEmpty())
  {
    fpm.removeAll(fpm);
  }

  for (int j = 0; j < tcfm.size(); j++) {
    ArrayList<Integer> temp = new ArrayList<Integer>();

    temp.add(j+1);
    for (int i = 1; i < frm.get(0).size(); i++) {
      int sum = 0;
      for (int k = 0; k < tcfm.size(); k++) {
        sum += tcfm.get(j).get(k) * frm.get(k).get(i);
      }
      temp.add(sum);
    }

    fpm.add(temp);
  }

  Object[][] objARMTemp = new Object[frm.size()][frm.get(0).size()];

  for (int i = 0; i < fpm.size(); i++) {
    for (int j = 0; j < fpm.get(0).size(); j++) {
      objARMTemp[i][j]=fpm.get(i).get(j);
    }
  }
  return objARMTemp;
```

```
  }

  public void showFPM() {
    for (int j = 0; j < fpm.size(); j++) {
      System.out.println(fpm.get(j).toString());
    }
  }

  public ArrayList<Integer> fragmentAllocation() {

    if(!fa.isEmpty())
    {
      fa.removeAll(fa);
    }

    for (int i = 1; i < fpm.get(0).size(); i++) {
      int min = 0, val = 99999;
      for (int j = 0; j < fpm.size(); j++) {
        int temp = fpm.get(j).get(i);
        if (temp < val) {
          min = j + 1;
          val = temp;
        }

      }
      fa.add(min);
    }
    return fa;
  }

  public void showFA() {

    System.out.println(fa.toString());

  }
}
```

```
package start;

import java.awt.BorderLayout;
import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.MediaTracker;
import java.awt.Toolkit;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;
import java.awt.CardLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.JLabel;

import tables.TableCreator;
import java.awt.Color;

public class CardGrid extends JFrame {

  private JPanel contentPane;
  private JTextField txt_aufm;
  private JTextField txt_tcfm;
  private JTextField txt_aam;

  private Checkbox graphP;
  private Checkbox heuristicP;
  private Checkbox costGraphP;

  private CheckboxGroup cbg;
```

```
JButton btnCreateAPM;
JButton btnBack_2;
JButton btnAA;
JButton btnBack_1;
JButton btnStartNew;
JButton btnBack;

private int action = -1;

String filename = null;
ArrayList<String> column = null;
JFileChooser fc;
File aufm;
File tcfm;
File aam;

private JButton btnCreateARM;
// private JTable table;
TableCreator tc;
// Object[][] objARM;
private JTable table_1;
Image bgimage = null;

CardLayout card;

class ContentPanel extends JPanel {
  Image bgimage = null;

  ContentPanel() {
    MediaTracker mt = new MediaTracker(this);
    bgimage = Toolkit.getDefaultToolkit()
        .getImage("molecule-world.png");
    mt.addImage(bgimage, 0);
    try {
      mt.waitForAll();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    int imwidth = bgimage.getWidth(null);
    int imheight = bgimage.getHeight(null);
    g.drawImage(bgimage, 500, 300, null);
```

```
    }

    @Override
    public void paint(Graphics g) {
      // TODO Auto-generated method stub
      super.paint(g);
      int imwidth = bgimage.getWidth(null);
      int imheight = bgimage.getHeight(null);
      g.drawImage(bgimage, 500, 300, null);
    }
  }

  /**
   * Launch the application.
   */
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
          CardGrid frame = new CardGrid();
          frame.setVisible(true);
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
  }

  /**
   * Create the frame.
   */
  public CardGrid() {
    setTitle("Fragmentor");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(50, 50, 1000, 700);

    final ContentPanel cp = new ContentPanel();

    contentPane = new JPanel();
    cp.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(cp);
    card = new CardLayout();

    boolean b_aufm = false;
    boolean b_tcfm = false;
```

```
cp.setLayout(card);

final JPanel panel = new JPanel();
panel.setBackground(new Color(153, 0, 255));
cp.add(panel, "panel");

final JPanel panel1 = new JPanel();
panel1.setBackground(new Color(153, 0, 255));
cp.add(panel1, "panel1");

final JPanel panel2 = new JPanel();
panel2.setBackground(new Color(153, 0, 255));
cp.add(panel2, "panel2");

final JPanel panel3 = new JPanel();
panel3.setBackground(new Color(153, 0, 255));
cp.add(panel3, "panel3");

panel.setLayout(null);
panel1.setLayout(null);
panel2.setLayout(null);
panel3.setLayout(null);

btnBack = new JButton("Back");
btnBack.setBounds(679, 522, 106, 23);
panel3.add(btnBack);

btnBack.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    card.previous(cp);

  }
});

btnStartNew = new JButton("Start New");
btnStartNew.setBounds(811, 522, 106, 23);
panel3.add(btnStartNew);
btnStartNew.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    txt_aam.setText("");
    txt_aufm.setText("");
    txt_tcfm.setText("");
    card.first(cp);

  }
});
```

```
tc = new TableCreator();
filename = File.separator + "tmp";
fc = new JFileChooser(new File(filename));

txt_aufm = new JTextField();
txt_aufm.setBounds(168, 38, 223, 25);
panel.add(txt_aufm);
txt_aufm.setColumns(10);

JLabel lblChosseAufmFile = new JLabel("Chose AUFM file");
lblChosseAufmFile.setBounds(28, 43, 184, 14);
panel.add(lblChosseAufmFile);

JButton btnBrowseAUFM = new JButton("Browse");
btnBrowseAUFM.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    fc.showOpenDialog(new JFrame());
    aufm = fc.getSelectedFile();
    txt_aufm.setText(aufm.toString());
    try {
      column = tc.createAUFM(aufm);
      tc.showAUFM();

      if (!(txt_tcfm.getText().equals("")))
        btnCreateARM.setEnabled(true);
    } catch (IOException e1) {
      // TODO Auto-generated catch block
      e1.printStackTrace();
    }
  }
});
btnBrowseAUFM.setBounds(413, 39, 89, 23);
panel.add(btnBrowseAUFM);

JLabel lblChoseTcfmFile = new JLabel("Chose TCFM file");
lblChoseTcfmFile.setBounds(28, 87, 184, 14);
panel.add(lblChoseTcfmFile);

txt_tcfm = new JTextField();
txt_tcfm.setBounds(168, 82, 223, 25);
panel.add(txt_tcfm);
txt_tcfm.setColumns(10);

JButton btnBrowseTCFM = new JButton("Browse");
btnBrowseTCFM.addActionListener(new ActionListener() {
```

```
  public void actionPerformed(ActionEvent e) {
    fc.showOpenDialog(new JFrame());
    tcfm = fc.getSelectedFile();
    txt_tcfm.setText(tcfm.toString());
    try {
      tc.createTCFM(tcfm);
      tc.showTCFM();

      if (!(txt_aufm.getText().equals("")))
        btnCreateARM.setEnabled(true);
    } catch (IOException e1) {
      // TODO Auto-generated catch block
      e1.printStackTrace();
    }
  }
});
btnBrowseTCFM.setBounds(413, 83, 89, 23);
panel.add(btnBrowseTCFM);

JLabel lblChoseAamFile = new JLabel("Chose AAM file");
lblChoseAamFile.setBounds(28, 131, 184, 14);
panel.add(lblChoseAamFile);

txt_aam = new JTextField();
txt_aam.setBounds(168, 126, 223, 25);
panel.add(txt_aam);
txt_aam.setColumns(10);

JButton btnBrowseAAM = new JButton("Browse");
btnBrowseAAM.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    fc.showOpenDialog(new JFrame());
    aam = fc.getSelectedFile();
    txt_aam.setText(aam.toString());
    try {
      tc.createAAM(aam);
      tc.showAAM();
      tc.createGraph();
      // btnCreateARM.setEnabled(true);
    } catch (IOException e1) {
      // TODO Auto-generated catch block
      e1.printStackTrace();
    }
  }
});
btnBrowseAAM.setBounds(413, 127, 89, 23);
```

```
panel.add(btnBrowseAAM);

cbg = new CheckboxGroup();
graphP = new Checkbox("Graph", false, cbg);
heuristicP = new Checkbox("Heuristic", false, cbg);
costGraphP = new Checkbox("Cost Graph", false, cbg);

graphP.setBounds(60, 180, 89, 23);
panel.add(graphP);
heuristicP.setBounds(160, 180, 89, 23);
panel.add(heuristicP);
costGraphP.setBounds(260, 180, 89, 23);
panel.add(costGraphP);

graphP.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent itemEvent) {

    action = 0;
  }
});

heuristicP.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent itemEvent) {

    action = 1;
  }
});

costGraphP.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent itemEvent) {

    action = 2;
  }
});

btnCreateARM = new JButton("Create Requst Matrix");
btnCreateARM.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {

    panel1.removeAll();
    panel1.add(btnBack_2);
    panel1.add(btnCreateAPM);
    panel1.validate();

    if (action == 2) {
      System.out.println("action 2");
```

```
        tc.createARM();
        Object[][] objFRM = tc.createFRM();
        String[] clm = { "Fragment1", "Fragment2", "Fragment3" };

        System.out.println(objFRM.toString());

        JTable table = new JTable(objFRM, clm);

        table.setBounds(10, 50, 800, 300);
        JScrollPane scrollPane = new JScrollPane(table);
        table.setVisible(true);

        scrollPane.setVisible(true);// scrollPane.setBounds(640,
                    // 437,
                    // -631, -259);
        table.setToolTipText("Fragment Request Matrix");
        JLabel lbl = new JLabel("Fragment Request Matrix");
        lbl.setBounds(28, 27, 184, 14);

        panel1.add(lbl);

        scrollPane.setBorder(new EmptyBorder(0, 0, 0, 0));
        scrollPane.setBounds(15, 50, 850, 600);
        scrollPane.getViewport().setBackground(
            new Color(153, 0, 255));
        panel1.add(scrollPane);
        card.show(cp, "panel1");

    } else {
        Object[][] objARM = tc.createARM();

        System.out.println(objARM.toString());
        JTable table = new JTable(objARM, column.toArray());

        table.setBounds(10, 50, 800, 300);
        JScrollPane scrollPane = new JScrollPane(table);
        table.setVisible(true);

        scrollPane.setVisible(true);// scrollPane.setBounds(640,
                    // 437,
                    // -631, -259);
        table.setToolTipText("Attribute Request Matrix");
        JLabel lbl = new JLabel("Attribute Request Matrix");
        lbl.setBounds(28, 27, 184, 14);

        panel1.add(lbl);
```

```
        scrollPane.setBorder(new EmptyBorder(0, 0, 0, 0));
        scrollPane.setBounds(15, 50, 850, 600);
        scrollPane.getViewport().setBackground(
            new Color(153, 0, 255));
        panel1.add(scrollPane);
        card.show(cp, "panel1");
      }
    }
});

btnCreateARM.setEnabled(false);
btnCreateARM.setBounds(225, 242, 178, 23);
panel.add(btnCreateARM);
btnCreateARM.setVisible(true);

btnCreateAPM = new JButton("Create Pay Matrix");
btnCreateAPM.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {

    panel2.removeAll();
    panel2.add(btnBack_1);
    panel2.add(btnAA);
    panel2.validate();

    if (action == 2) {
      tc.createAPM();
      Object[][] objFPM = tc.createFPM();
      String[] clm = { "Site","Fragment1", "Fragment2", "Fragment3" };

      System.out.println(objFPM.toString());
      JTable table = new JTable(objFPM, clm);
      table.setBounds(10, 10, 800, 300);
      JScrollPane scrollPane = new JScrollPane(table);
      table.setVisible(true);
      scrollPane.setVisible(true);// scrollPane.setBounds(640,
                  // 437,
                  // -631, -259);
      table.setToolTipText("Fragment Pay Matrix");
      JLabel lbl = new JLabel("Fragment Pay Matrix");
      lbl.setBounds(28, 27, 184, 14);
      panel2.add(lbl);

      scrollPane.getViewport().setBackground(
          new Color(153, 0, 255));
      scrollPane.setBorder(new EmptyBorder(0, 0, 0, 0));
```

```
        scrollPane.setBounds(15, 50, 850, 600);
        panel2.add(scrollPane);
        card.next(cp);

      } else {
        Object[][] objAPM = tc.createAPM();

        System.out.println(objAPM.toString());
        JTable table = new JTable(objAPM, column.toArray());
        table.setBounds(10, 10, 800, 300);
        JScrollPane scrollPane = new JScrollPane(table);
        table.setVisible(true);
        scrollPane.setVisible(true);// scrollPane.setBounds(640,
                    // 437,
                    // -631, -259);
        table.setToolTipText("Attribute Pay Matrix");
        JLabel lbl = new JLabel("Attribute Pay Matrix");
        lbl.setBounds(28, 27, 184, 14);
        panel2.add(lbl);

        scrollPane.getViewport().setBackground(
            new Color(153, 0, 255));
        scrollPane.setBorder(new EmptyBorder(0, 0, 0, 0));
        scrollPane.setBounds(15, 50, 850, 600);
        panel2.add(scrollPane);
        card.next(cp);
      }
    }
  });

  btnCreateAPM.setBounds(366, 230, 178, 23);
  panel1.add(btnCreateAPM);

  btnBack_2 = new JButton("Back");
  btnBack_2.setBounds(180, 230, 114, 23);
  panel1.add(btnBack_2);
  btnBack_2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      card.previous(cp);

    }
  });

  btnAA = new JButton("Allocation");
  btnAA.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```
panel3.removeAll();
panel3.add(btnStartNew);
panel3.add(btnBack);
panel3.validate();

if (action == 2) {

  ArrayList<Integer> al = tc.attributeAllocation();
  Object[][][] sites = new Object[4][][];

  for (int i = 1; i < 5; i++) {
    int j = 1;

    for (int k = 0; k < al.size(); k++) {
      if (al.get(k) == i)
        j++;
    }
    sites[i - 1] = new Object[j][1];
    System.out.println(j);
  }

  int row[] = { 0, 0, 0, 0 };

  for (int k = 0; k < al.size(); k++) {
    int temp = al.get(k) - 1;

    sites[temp][row[temp]][0] = column.get(k + 1);
    row[temp]++;
    System.out.println(temp);
    System.out.println(row[temp]);
  }

  String[] clm = { "Attribute" };

  JLabel site1 = new JLabel("Site 1");
  site1.setBounds(45, 5, 50, 25);
  panel3.add(site1);
  JTable table = new JTable(sites[0], clm);
  table.setBounds(20, 10, 100, 200);

  JLabel site2 = new JLabel("Site 2");
  site2.setBounds(345, 5, 50, 25);
  panel3.add(site2);

  JTable table2 = new JTable(sites[1], clm);
```

```
table2.setBounds(310, 10, 100, 200);

JLabel site3 = new JLabel("Site 3");
site3.setBounds(45, 285, 50, 25);
panel3.add(site3);

JTable table3 = new JTable(sites[2], clm);
table3.setBounds(10, 310, 100, 200);

JLabel site4 = new JLabel("Site 4");
site4.setBounds(345, 285, 50, 25);
panel3.add(site4);
JTable table4 = new JTable(sites[3], clm);
table4.setBounds(310, 310, 100, 200);

JScrollPane scrollPane = new JScrollPane(table);
JScrollPane scrollPane1 = new JScrollPane(table2);
JScrollPane scrollPane2 = new JScrollPane(table3);
JScrollPane scrollPane3 = new JScrollPane(table4);

// table.setVisible(true);
// scrollPane.setVisible(true);// scrollPane.setBounds(640,
// 437,
// -631, -259);
/*
 * scrollPane.setBounds(5, 5, 550, 600);
 * scrollPane.add(table); scrollPane.add(table2);
 * scrollPane.add(table3); scrollPane.add(table4);
 *
 * scrollPane.repaint();
 *
 * panel3.add(scrollPane);
 */

scrollPane.setBounds(5, 25, 110, 210);
scrollPane1.setBounds(305, 25, 110, 210);
scrollPane2.setBounds(5, 305, 110, 210);
scrollPane3.setBounds(305, 305, 110, 210);

panel3.add(scrollPane);
panel3.add(scrollPane1);
panel3.add(scrollPane2);
panel3.add(scrollPane3);

card.next(cp);
```

```
} else {
  ArrayList<Integer> al = tc.attributeAllocation();
  Object[][][] sites = new Object[4][][];

  for (int i = 1; i < 5; i++) {
    int j = 1;

    for (int k = 0; k < al.size(); k++) {
      if (al.get(k) == i)
        j++;
    }
    sites[i - 1] = new Object[j][1];
    System.out.println(j);
  }

  int row[] = { 0, 0, 0, 0 };

  for (int k = 0; k < al.size(); k++) {
    int temp = al.get(k) - 1;

    sites[temp][row[temp]][0] = column.get(k + 1);
    row[temp]++;
    System.out.println(temp);
    System.out.println(row[temp]);
  }

  String[] clm = { "Attribute" };

  JLabel site1 = new JLabel("Site 1");
  site1.setBounds(45, 5, 50, 25);
  panel3.add(site1);
  JTable table = new JTable(sites[0], clm);
  table.setBounds(20, 10, 100, 200);

  JLabel site2 = new JLabel("Site 2");
  site2.setBounds(345, 5, 50, 25);
  panel3.add(site2);

  JTable table2 = new JTable(sites[1], clm);
  table2.setBounds(310, 10, 100, 200);

  JLabel site3 = new JLabel("Site 3");
  site3.setBounds(45, 285, 50, 25);
  panel3.add(site3);

  JTable table3 = new JTable(sites[2], clm);
```

```
        table3.setBounds(10, 310, 100, 200);

        JLabel site4 = new JLabel("Site 4");
        site4.setBounds(345, 285, 50, 25);
        panel3.add(site4);
        JTable table4 = new JTable(sites[3], clm);
        table4.setBounds(310, 310, 100, 200);

        JScrollPane scrollPane = new JScrollPane(table);
        JScrollPane scrollPane1 = new JScrollPane(table2);
        JScrollPane scrollPane2 = new JScrollPane(table3);
        JScrollPane scrollPane3 = new JScrollPane(table4);

        // table.setVisible(true);
        // scrollPane.setVisible(true);// scrollPane.setBounds(640,
        // 437,
        // -631, -259);
        /*
         * scrollPane.setBounds(5, 5, 550, 600);
         * scrollPane.add(table); scrollPane.add(table2);
         * scrollPane.add(table3); scrollPane.add(table4);
         *
         * scrollPane.repaint();
         *
         * panel3.add(scrollPane);
         */

        scrollPane.setBounds(5, 25, 110, 210);
        scrollPane1.setBounds(305, 25, 110, 210);
        scrollPane2.setBounds(5, 305, 110, 210);
        scrollPane3.setBounds(305, 305, 110, 210);

        panel3.add(scrollPane);
        panel3.add(scrollPane1);
        panel3.add(scrollPane2);
        panel3.add(scrollPane3);

        card.next(cp);
      }
    }
});

btnAA.setBounds(366, 230, 178, 23);
panel2.add(btnAA);

btnBack_1 = new JButton("Back");
```

```
    btnBack_1.setBounds(180, 230, 114, 23);
    panel2.add(btnBack_1);
    btnBack_1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        card.previous(cp);

      }
    });
  }
}
```