DISTRIBUTED DATABASE DESIGN WITH INTEGER LINEAR PROGRAMMING AND EVOLUTIONARY HYBRID ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

UMUT TOSUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

MAY 2013

Approval of the thesis:

DISTRIBUTED DATABASE DESIGN WITH INTEGER LINEAR PROGRAMMING AND EVOLUTIONARY HYBRID ALGORITHMS

submitted by **UMUT TOSUN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Ahmet Coşar
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Ali Doğru
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Murat Manguoğlu
Computer Engineering Department, METU _____

Assist. Prof. Dr. İsmail Sengör Altıngövde
Computer Engineering Department, METU _____

Assist. Prof. Dr. Refik Çağlar Kızılırmak
Electrical and Electronics Engineering Department, KTO Karatay University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: UMUT TOSUN

Signature :

ABSTRACT

DISTRIBUTED DATABASE DESIGN WITH INTEGER LINEAR PROGRAMMING AND EVOLUTIONARY HYBRID ALGORITHMS

Tosun, Umut

Ph.D., Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Ahmet Coşar

May 2013, 118 pages

The communication costs of remote access and retrieval of table fragments required in the execution of distributed database queries, are the major factors determining the quality of a distributed database design. Data allocation algorithms try to minimize these costs by dividing database tables into horizontal fragments, then assigning each fragment at or near the database sites they are needed more frequently. In this thesis, we propose efficient optimization algorithms for centralized and distributed databases based on integer linear programming and the well known Quadratic Assignment Problem (*QAP*). In the context of distributed database design, we have formulated an optimization problem where site capacities, table replicas, communication link capacities and table fragmentation are handled. Integer linear programming is a powerful tool to represent all of these optimization parameters easily while existing models in the literature ignore one or more of these important and practical aspects. The proposed model is based on integer linear programming and for the first time it handles all of these criteria in a consistent formulation. The *QAP* is a difficult and important problem studied in the domain of combinatorial optimization. It is possible to solve *QAP* instances with 10-20 facilities using exhaustive parallel algorithms within a few days on a cluster machine. We solve large QAP instances using a variety of genetic algorithm crossover operators for these problems and experimentally verified their performance using benchmark *QAP* instances from *QAPLIB* library. We propose and experimentally evaluate the island parallel QAP-IPGA algorithm. Then, we propose a new Parallel Hybrid Genetic Tabu Search based algorithm (*PHGETS*) which has an intelligent diversification phase executed on a seed solution obtained using the genetic algorithm. *PHGETS* consistently achieves results on average within 0.05 % of the best solutions given in *QAPLIB* for all problem instances that were considered.

Keywords: Parallel Programming, Multiple Query Optimization (*MQO*), Quadratic Assignment Problem (*QAP*), Tabu Search, Integer Linear Programming

ÖZ

SAYISAL LİNEER PROGRAMLAMA VE BULUŞSAL HİBRİT ALGORİTMALAR İLE DAĞITIK VERİTABANI TASARIMI

Tosun, Umut

Doktora, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Doç. Dr. Ahmet Coşar

Mayıs 2013 , 118 sayfa

Sorgular tarafından tablo parçalarına uzaktan erişim ve bulunan lokasyona getirme sonucu ortaya çıkan haberleşme maliyeti dağıtık veritabanı sorgularındaki temel işletim maliyetini oluşturmaktadır. Veri yerleştirme algoritmaları bu maliyeti parçaları yakın yerleşim birimlerine atayarak minimuma indir meye çalışır. Paralel işleme ve sayısal lineer programlama uygulamalarının artması ile veri yerleştirme ve çoklu sorgu optimizasyonu problemlerini modellemeye olan ihtiyaç daha önemli hale gelmiştir. Bu tezde merkezi ve dağıtık veritabanları için sayısal lineer programlama ve iyi bilinen Karesel Atama Problemi (*KAP*) tabanlı algoritmalar sunulmaktadır. Dağıtık veritabanı tasarımı kapsamında da tesis kapasitesi, tablo replikaları, haberleşme hatları ve tablo parçaları olan ağ topolojilerine konsantre olunmuştur. Bir çok sistem bu gerçek hayatta bulunan faktörlerin bir yada birden fazlasını ihmal ederken, sayısal lineer programlama bu kriterlerin hepsini kolaylıkla güçlü bir şekilde ifade edebilmektedir. İlk defa sayısal lineer programlama ile bütün bu kriterleri uygun bir şekilde ele alan bir model geliştirilmiştir. *KAP*, kombinatoryal optimizasyon alanında çalışılan önemli ve güç bir problem çeşididir. 10-20 tesisli *KAP* örneklerini paralel algoritmalarla birkaç gün içerisinde küme bilgisayarlarda tüm olasılıkları deneyerek çözmek mümkündür. 100 tesis gibi büyük *KAP* örneklerini tüm olasılıkları deneyerek çözmek pratik olarak mümkün değildir. Bu çalışma kapsamında bir çok Genetik Algoritma çaprazlama operatörü araştırılmış ve deneysel olarak performansları *QAPLIB* kütüphanesinin bilinen örnekleri üzerinde denenmiştir. Genetik algoritmayı paralelize etmek için her küme işlemcisinde ada modeli kullanılarak ayrı havuzlar oluşturulmuş ve geliştirilmiştir. Bu tezde genetik algoritma bazlı üretilen tohum üzerinde akıllı çeşitlendirme fazı çalıştırarak oluşan yeni bir paralel hibrit genetik tabu araması algoritması (*PHGTA*) sunulmaktadır. Paralel hesaplama hem çeşitlendirme hem de son arama fazlarını hızlandırır. Deneylerimizi *QAPLIB* kütüphanesi üzerinde çok geniş bir set üzerinde gerçekleştirdik. *PHGTA* algoritmasının çözüm kalitesi ve optimizasyon zamanı olarak literatürdeki sıralı ve paralel yöntemlerle karşılaştırıldığında deneysel olarak daha iyi sonuçlar verdiği gözlemlenmiştir.

PHGTA tutarlı olarak yaklaşık 0.05 % ortalamayla *QAPLIB* kütüphanesindeki kullanılan problem örnekleri için verilen en iyi sonuçları elde eder.

Anahtar Kelimeler: Paralel Programlama, Karesel Atama Problemi, oklu Sorgu Optimizasyonu, Tabu Araması, Sayısal Lineer Programlama

To my parents and my lovely niece Zeynep Ela

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Ahmet Coşar for all his trust, encouragement and support throughout my graduate studies.

I would like to thank my committee members Prof. Dr. Ali Doğru, Assoc. Prof. Dr. Murat Manguoğlu, Assist. Prof. Dr. Refik Çağlar Kızılırmak, Assist. Prof. Dr. İsmail Sengör Altıngö vde for reading this thesis and their valuable comments.

Finally, I am grateful to my family and all my friends who have made my life easier during my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

QAP	Quadratic Assignment Problem
MQO	Multiple Query Optimization
IPGA	Island Parallel Genetic Algorithm
PHGETS	Parallel Hybrid Genetic Tabu Search Algorithm
LP	Linear Programming
TS	Tabu Search
RTS	Robust Tabu Search
GA	Genetic Algorithm
HPC	High Performance Cluster
TSP	Travelling Salesman Problem
PGA	Parallel Genetic Algorithm
CPTS	Cooperative Parallel Tabu Search
B-IPGA	Bullwhip Island Parallel Genetic Algorithm

CHAPTER 1

INTRODUCTION

1.1 Motivation

Advances in database technologies have created an important area for application development in information networks. A dramatic increase has occurred in Distributed Database Management Systems (*DDBMS*) usage in recent years with the development of network based enterprise applications. Quality of Service (*QoS*) is very important in *DDBMS* applications. The response time of the queries, data integrity and consistency are crucial, and efficient algorithms are used to model data fragmentation, data allocation, replication, query frequencies and table update costs. *DDBMS* queries access several tables and fragments over the network. A query is initialized from a site and the major part of the total cost arises from the allocation of fragments which the query must retrieve using network from different sites. Data allocation algorithms try to assign the fragments to *DDB* sites in a way to minimize the total cost of data transfer while executing user and/or application queries. The data allocation problem is NP-complete and it aims to find an optimal fragment assignment solution while also taking into account replication, update costs and average query response times. Different queries may share the same subtasks and the same queries may be issued from different originating sites. We have designed a model based on integer linear programming which considers all of these concerns. We further investigated the problem with different heuristic algorithms for large *DDB* plans and topologies. *DDBMS* design is a problem with a set of multiple objectives including efficient use of computer storage & processing resources, minimal query response times, while taking care not to violate site capacity constraints. It is necessary to model the problem in a way satisfying all of these criteria. Several algorithms have been proposed in the literature for the data allocation and data fragmentation problems in distributed databases. Techniques based on Genetic Algorithms have been used by Corcoran [41] and Frieder [66]. However, their formulations don't consider site capacities and replication of fragments/tables to improve query response times. Ahmad [2] proposed genetic algorithm, simulated annealing and mean field annealing solutions but non redundant data (i.e. replication) is not considered. Adl [1] proposed an ant colony heuristic and modelled the Data Allocation Problem (*DAP*) as a Quadratic Assignment Problem (*QAP*), however update costs and replication costs and/or benefits were not handled in this work. The data allocation problem has similarities with the *QAP* and Fragment Allocation Problem (*FAP*).

The *QAP* is a well known NP-hard combinatorial optimization problem [69] first introduced by [88] as a mathematical model for the location of indivisible economic activities. It has been one of the most interesting challenges of scientists since then, using it to model a variety of problems. Typewriter keyboard design, backboard wiring [136], layout design [127], turbine balancing [124], scheduling [98], facility layout problem [134], graphics hardware acceleration [149], material handling devices [13], dynamic job shop scheduling [6], multiple objective sequencing [7], and data allocation [1] are

some of the problem areas that have been successfully modelled using *QAP*. Assignment of airport gates to arriving and departing airplanes while trying to minimize the distances that have to be walked by passengers [77], and assigning rooms to people with neighborhood constraints [38] have both been modelled as a *QAP*. The allocation of container handling services to storage and maneuvering locations for containers in a shipyard [42], the maximum clique problem, the linear ordering problem, and the graph-partitioning problem are also among the interesting application areas of the *QAP*. Task managing considering cross-training goals [116], optimal process plan selection in manufacturing [105], manufacturing control [138] can also be modelled as *QAP*.

There is no efficient algorithm that is guaranteed to find an optimal solution for *QAP* instances with sizes greater than 30. Therefore, researchers have proposed sequential and parallel solutions for larger instances of the *QAP*. [8] proposed an exact solution using a parallel branch-and-bound algorithm implemented as a parallel algorithm running on a powerful grid-computer and it has been successfully used for optimally solving problem instances with upto 30 facilities. For a detailed survey of the *QAP* solutions, the work by [102] can be referred. We propose a fully informed *IPGA* (*QAP-IPGA*) for solving *QAP*. Three different crossover operators and three different selection mechanisms are evaluated for this new algorithm to decide the best performing one, which is applied on the selected instances of the *QAPLIB*. We have performed extensive experiments on these instances and report their results in this thesis.

Metaheuristics applied for solving the *QAP* can be either sequential or parallel from the perspective of our study. Sequential metaheuristics have many different approaches, here we will give brief information about the most successful ones. Robust Tabu Search (*RTS*) is one of the best known algorithms that can produce high quality solutions quickly [141, 62]. *RTS* is a variant of the simple Tabu Search *(TS)* algorithm. [81], introduced a multi-start *TS* algorithm (*JRG-DivTS*). When the search stagnates, *JRG-DivTS* first finds a local best solution, then diversifies this solution and uses it as a restart point for the search. [109] uses *RTS* as a diversification operator and applies perturbation on *RTS* solutions in order to escape from local minima. Symmetric and asymmetric instances of Taillard in the *QAPLIB* are solved successfully with this algorithm. Sequential metaheuristics like *RTS* can produce very high quality solutions when they are used in combination with variants of Genetic Algorithms *(GAs)*. Misevicius introduced two *GA*s [110] that combine *RTS* with diversification operators. The first one is *M-GA/TS* that performs a ruin-and-recreate strategy. A ruin or crossover method is used for perturbing a local best solution and a variant of *RTS* is used for recreation. The second algorithm *M-GA/TS-I* applies a random ruin procedure to the output of the Genetic Algorithm *GA* operators. [3], [57] and [58] studied on *GA* variants that are incorporated with *TS*.

Ant colony optimization [139, 67], simulated annealing [40], and neural networks [30] are some of the other sequential methods that have been successfully applied to this problem. Sequential metaheuristics have made significant amounts of progress towards the solution of the *QAP* and they can provide very good quality solutions. However, the complexity and the computational requirements of intractable problems are very high. The granularity of the solution space, the repetitive behavior of the metaheuristic methods, and the solution complexity make the parallelization an attractive alternative for these problems. The parallelization can provide better solutions and reduce the search time required.

1.2 Distributed Database Design and Quadratic Assignment Problem

Although distributed databases (*DDBs*) are attractive for very large datasets, their utilization brings new problems. Designing a *DDB* is one of the most complicated problems in this domain. In addition to the classical centralized database design, fragmentation and data allocation are the two new problems to tackle with [119, 55]. *DAP* is an optimization problem with constraints [95]. Disk drive speed, parallelism of the queries, network traffic, load balancing of servers should be considered during the design. Even without most of these decision parameters it is clear that *DAP* is an NP-Hard problem. File allocation problem (*FAP*), *DAP*, and *QAP* have some similarities. From the perspective of data transmission, *DAP* is the same with *FAP*. However, the logical and semantic relations of fragments differ from these two problems. *QAP* has more similarities with *DAP* that it also keeps track of the resource locality. The *QAP* has been first presented in Koopmans and Beckman [88].

1.3 Distributed Database Design and Integer Linear Programming

There have been various studies on distributed database design with integer programming [43, 70, 126]. However, these studies lack of a comprehensive formulation for the database design problem with a general treatment of update costs, replication, query frequencies and communication topology. Their integer programming solutions are rather simple and lack a realistic network topology or a realistic query execution plan. All of these earlier formulations have a high-level view of the problem as allocation of horizontal/vertical fragments to sites/nodes, trying to find the allocation with the minimum cost. A well-known strategy has been developed in[43], deciding how to assign the database relations and join tasks to sites and proposes a query optimization strategy. The methods proposed in this work try to determine where to allocate data and where to execute the join operations in transactions concurrently. They also aim at minimizing the communication costs and achieve good resource utilization by these assignments to sites. Even though they use an integer programming approach, the algorithms proposed are complicated and the query optimization procedure, network utilization and data allocation are considered as separate issues. Ailamaki and Papadomanolakis also used integer linear programming to determine realistic bounds for index selection [121]. They showed that integer linear programming is very efficient and tightly bounded to the optimal solutions when compared to heuristic algorithms. Table 1.1 compares the existing studies in the literature. (+) sign represents a feature existing and (-) sign represents a feature non-existing in the proposed work.

Table1.1: Integer Linear Programming and Distributed Database Design

	Fragmentation	Site Cap.	Replication	Update Cost	Comm. Cost	Query Freq.
CERI, NAVATHE, AND WIEDERHOLD [28]	-	-	-	-	+	+
DOUGLES AND PHILIP (1988) [43]	-	+	-	-	+	-
DOUGLES AND PHILIP (1990) [44]	+	-	+	-	+	-
CHU AND LEONG [150]	+	-	-	-	-	-
LIN AND ORLOWSKA [100]	-	+	-	-	+	-
PAPADOMANOLAKIS AND AILAMAKI [121]	-	-	-	-	-	-

1.4 Contributions

The contributions in this thesis consist of modelling the distributed database design problem and the multiple query optimization problem as integer linear programming problem, proposing efficient parallel algorithms for the well-known *QAP*, implementing the parallel approaches to the production research and finally showing that the *DAP* can be modelled as *QAP* with simulated annealing, tabu search and hybrid genetic algorithms as well as genetic algorithms and ant colony algorithms. We show that hybrid genetic algorithms outperform all these heuristics in terms of solution quality and have much shorter execution times. In the scope of modelling distributed database design problem and multiple query optimization problem, we showed that both of these problems can be solved with integer linear programming in practical times. Our *MQO* model with integer linear programming outperforms the well-known A-star algorithm in terms of execution time. It is possible to solve instances up to 100 queries having more than two plans with our model whereas A-star can solve up to 100 queries with two plans per query because of its exponentially increasing memory requirements as the number of queries and the number of alternative plans per query increase. Distributed database design problem is modelled with integer programming where network topologies, replication, update costs, originating sites, site capacities, query frequencies are all considered and related constraints are included in the problem model. To the best of our knowledge, it is the first time that a model in this detail is being proposed.

Another research direction we have followed in this thesis is to how the distributed database design problem can be transformed into *QAP*. As a consequence, we also proposed *QAP-IPGA* and *PHGETS* algorithms which can be executed on powerful cluster computers and outperform many of the *QAP* heuristics reported in the literature in terms of both run-time and also the solution quality. In addition to these contributions, we show that our parallel algorithms can be also used in production research domain especially to reduce the *bullwhip effect* in supply chains. Finally, we show that the distributed database design problem can be modelled and efficiently solved with simulated annealing, tabu search and hybrid genetic algorithms as well as genetic algorithms and ant colony algorithms. We show that the tabu search and hybrid genetic algorithms outperform the existing genetic algorithms and ant colony algorithms. The organization of the thesis is provided in the following paragraphs. This dissertation is divided into six chapters. Figure 1.1 shows the organization of the thesis.

Chapter 1 discusses the *MQO* problem, distributed database design problem and the application areas of the *QAP*. It presents the general motivation behind this thesis and gives a formal description of the *QAP*.

Chapter 2 gives preliminary information about integer programming, heuristic algorithms and *QAP*. It describes the usage of shortest path algorithm and integer linear programming to model a distributed database. It presents the application areas of the *QAP* and implementations to transform the problem into *DAP*.

Chapter 3 explains how our framework uses the integer linear programming to solve the *MQO* problem. A-star algorithm and integer linear programming formulation are explained in detail.

Chapter 4 presents the distributed database design problem and the corresponding integer linear programming formulation. Several examples are presented to explain how site capacities, replication, and query frequencies are reflected in the integer linear programming formulation and affect allocation of the tables. Later, the *QAP-IPGA* is proposed as our first *QAP* algorithm. The crossover techniques used in the genetic algorithm and how the permutation space can be assigned to the cluster nodes are explained. Furthermore, a brief overview of the general strategies of Parallel Genetic Algorithms *(PGAs)*

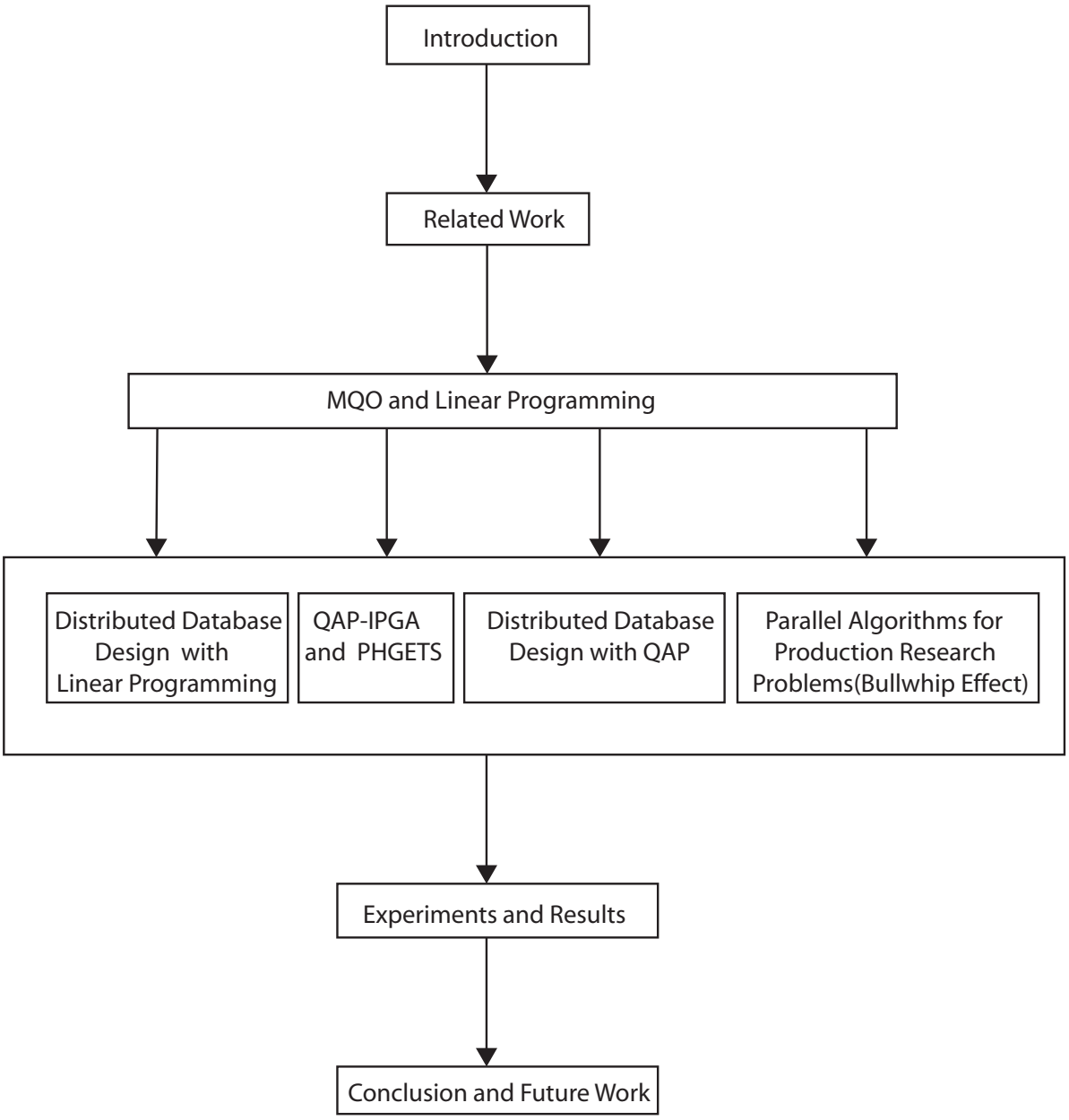

Figure 1.1: Organization of the Dissertation According to the Sections and Subsections.

and the three different crossover operators with the selection mechanisms we applied in the proposed *QAP-IPGA* are explained. The details of the exhaustive parallel algorithm are discussed which we developed to compare with the results of the *QAP-IPGA* to see how close the solutions are to optimal. We present our *PHGETS* algorithm and a brief overview of general strategies of the (*PGA*s) which is used for seed generation showing the cooperation of the *QAP-IPGA* and *RTS*. Finally, we formulate the Data Allocation Problem (*DAP*) as a *QAP*. We present a set of algorithms to solve *DAP*. We give a brief information about the related works for Fragment Allocation Problem *FAP*, *DAP*, and QAP. A new algorithm is introduced to minimize the well-known bullwhip effect in supply chains.

Chapter 5 discusses the experimental setup and results of the proposed methods on real world and synthetic problem instances generated by our software for performance evaluation and comparison.

Chapter 6 includes our concluding remarks and further discussions on the framework.

CHAPTER 2

RELATED WORK

A fundamental class of optimization problems involves assigning assets to tasks to minimize a desired cost function. These problems are categorized as Assignment Problems [123]. Variations of these problems have been studied over the years with a wide range of applications in the domains of telecommunications, transportation systems and signal processing [19]. The classical approach to the Quadratic Assignment Problem (*QAP*) was first introduced by Koopmans and Beckmann [88] as a mathematical model for the location of indivisible economic activities. Since then it has been one of the most interesting challenges for scientists having been used for modelling a great variety of problems. Typewriter keyboard design, backboard wiring [136], layout design [127], turbine balancing [124], scheduling [98], and data allocation [1] are some of the problems that have been successfully modeled as *QAP*. The service allocation problem with the purpose of minimizing the container re-handling operations at a shipyard [42], travelling salesman, bin-packing, maximum clique, linear ordering, and the graph-partitioning problem are among the interesting applications of the *QAP*.

In its simplest form, the *QAP* is the problem of assigning n facilities to n locations with cost proportional to the amount of material flow between the facilities multiplied by the distances between locations, the initial costs for placing the facilities at their respective locations. The objective is to find an allocation such that the total cost of allocating and operating all facilities is minimized. The *QAP* can be formally modeled by using three $n \times n$ matrices, A, B, and C.

$$A = (a_{ik}) \tag{2.1}$$

where a_{ik} is the flow amount from facility i to facility k.

$$B = (b_{jl}) \tag{2.2}$$

where b_{jl} is the distance from location j to location l.

$$C = (c_{ij}) \tag{2.3}$$

where c_{ij} is the cost of placing facility i at location j.

The Koopmans-Beckmann form of QAP can be written as:

$$min_{\phi \epsilon S_n}(\sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik} b_{\phi(i)\phi(k)} + \sum_{i=1}^{n} c_{i\phi(i)}) \tag{2.4}$$

where S_n is the set of all permutations of integers 1,2, ...,n. Each individual product $a_{ik}b_{\phi(i)\phi(k)}$ is the transportation cost caused by assigning facility i to location ϕ (i) and facility k to location ϕ (k). Each term $c_{i\phi(i)} + \sum_{k=1}^{n} a_{ik}b_{\phi(i)\phi(k)}$ is the total cost given, for facility i, the cost for installing it at location $\phi(i)$, plus the transportation costs to all other facilities k, installed at locations $\phi(1)$, $\phi(2)$ ... $\phi(n)$. An instance of the *QAP* with input matrices A, B, and C is denoted by *QAP* (A, B, C). If there is no C term, we can write it as *QAP* (A, B). Lawler [92] introduced a four-index cost array $D = (d_{ijkl})$ instead of the three matrices and obtained the general form of the *QAP* as:

$$min_{\phi \epsilon S_n}(\sum_{i=1}^{n}\sum_{k=1}^{n} d_{i\phi(i)k\phi(k)}) \tag{2.5}$$

The relationship with the Koopmans-Beckmann problem is

$$d_{ijkl} = a_{ik}b_{jl}(i, j, k, l = 1, 2, ..., n; i \neq k \text{ or } j \neq l) \tag{2.6}$$

$$d_{ijij} = a_{ii}b_{jj} + c_{ij}(i, j = 1, 2, ..., n) \tag{2.7}$$

The applicability of the *QAP* to the solution of many different problems has made it the subject of extensive research area for exhaustive and metaheuristic strategies. Small size *QAP* instances are appropriate for exact solutions but the larger instances cannot be solved in reasonable times due to the computational limits. Therefore, metaheuristic approaches have gained a reputation for their ability to produce high-quality solutions within the computational limitations. Simulated Annealing [40, 151], Neural Networks [23], Genetic Algorithms (*GA*s) [22, 145], GRASP [97], Tabu Search (*TS*) [15], and Ant Colony Optimization [67] are some of the well-known metaheuristics that have been successfully applied to the *QAP*.

We present an overview of the sophisticated studies in the area. Most are hybrid approaches in which, usually, a local search technique like *TS* is used. Taillard's Robust *TS* [141] is a very effective way of local searching and its computational time is so short that this property makes it a very attractive tool for the researchers using the *QAP*. In this thesis, we propose a new Parallel Hybrid Genetic *TS* Algorithm (*PHGETS*) for the solution of the *QAP*. It combines the metaheuristic of Parallel Genetic Algorithms (*PGA*s), with the local search technique of *TS*. We used the *QAPLIB* [20] benchmark during the tests and performed extensive experiments on these instances and reported their results. *PHGETS* can produce high-quality solutions in shorter times than most of the best algorithms in the literature.

2.1 Solution Methods for the *QAP*

The *QAP* has been studied extensively since it was introduced in 1957. It is proven to be NP-complete [129], so that no polynomial time algorithm is able to exactly solve this problem for larger data sets. Several algorithms have been proposed for both exact and approximate solutions to the problem. Exact algorithms are limited to solving small data sets of the *QAP* with massively parallel computers whereas metaheuristics can provide near-optimal solutions within reasonable optimization times. This property of metaheuristics has made them prominent for solving the *QAP* instances, therefore many researchers have proposed different heuristics or hybrid approaches to solve this problem. In this section, we present a summary of the successful approaches from the literature.

2.1.1 Exact Solution Methods

Branch and Bound (*BB*) algorithms are the most elegant approaches to solve the *QAP* exactly [92, 26, 72]. Pardalos *et al.* [122] gives a detailed overview of *BB* algorithms. It was not possible to find exact solutions to QAP problems of size 20 until 1990. Mautor and Roucairol [107] gave the exact solutions for nug16, els19, and the problem of the size-20 instance. Clausen and Perregaard [39] solved the instance of nug20 with a parallel *BB* algorithm. Marzetta and Brungger [106] solved the nug25 instance by parallel dynamic programming. Anstreicher and Brixius [9] developed a convex quadratic programming relaxation within a *BB* algorithm and this algorithm provided an exact solution for the nug25 instance after 13 days of CPU time using sequential processing. Hahn and Krarup [78] solved the kra30a after 99 days of work with a sequential workstation. Nystrom [114] gave the optimal solution for the ste36b and ste36c instances after 200 days of work in a distributed environment. Anstreicher *et al.* [8] reported the exact solution of the nug30, which required seven days with 650 processors roughly equal to seven years of computation on a single *CPU*.

2.1.2 Genetic Algorithms

*GA*s were first introduced by Holland [79] in the early 1970's and they have been widely used since then. The robustness and simplicity of *GA*s make them very attractive algorithms. They simulate the natural process of selection and evolution; surviving, reproducing and transferring their characteristics to new generations. A phenotype is considered as an individual in the search space and represented by strings called chromosomes. The atomic building blocks of chromosomes are called genes. Each gene encodes a specific character of a chromosome, forming a solution instance when they are brought together (the order of genes is significant). A simple *GA* randomly generates an initial population of solutions, then by applying selection, crossover, and mutation operations repetitively, creates new generations [74]. The individual having the best fitness value is returned as the best (discovered) solution to the problem. The termination condition may be defined in relation to the total execution time, the number of generations produced and as having no (or very little) improvement in the average fitness value of the population [147, 54, 132].

Different mechanisms are used by *GA*s to select the individuals for mating with tournament, roulette wheel, and truncation being the most frequently used methods. In the tournament method, the best one among the randomly selected individuals is used for mating. In the roulette wheel method, a random number is generated to choose an individual from solution pool for mating where the probability of selecting an individual depends on its fitness value thus, giving reproductive advantage to fitter individuals. In the truncation method, individuals with the lowest fitness values are eliminated from the population with a previously defined and constant percentage.Individuals are sorted by their fitness values. Parents are selected according to a threshold value (10% for our experiments). Only the individuals above the threshold are submitted to crossover and mutation operators. In *GA*'s, the best individuals of the population are guaranteed to survive for future generations. Recombination of the characteristics of the parents is achieved by the crossover operator which generates new individuals by mixing the genes of the parents. Basically, parent chromosomes are split into parts and used to generate the chromosome for the new offspring. Moreover, mutations can occur randomly, causing modifications to an individual's genetic code and effectively producing a new solution [132]. A new generation is produced after executing the genetic operators and this production of successive generations continues until a stopping condition is satisfied. A so called fitness function is used by *GA*s, in order to decide which solutions are better able to direct the search towards the optimum solution.

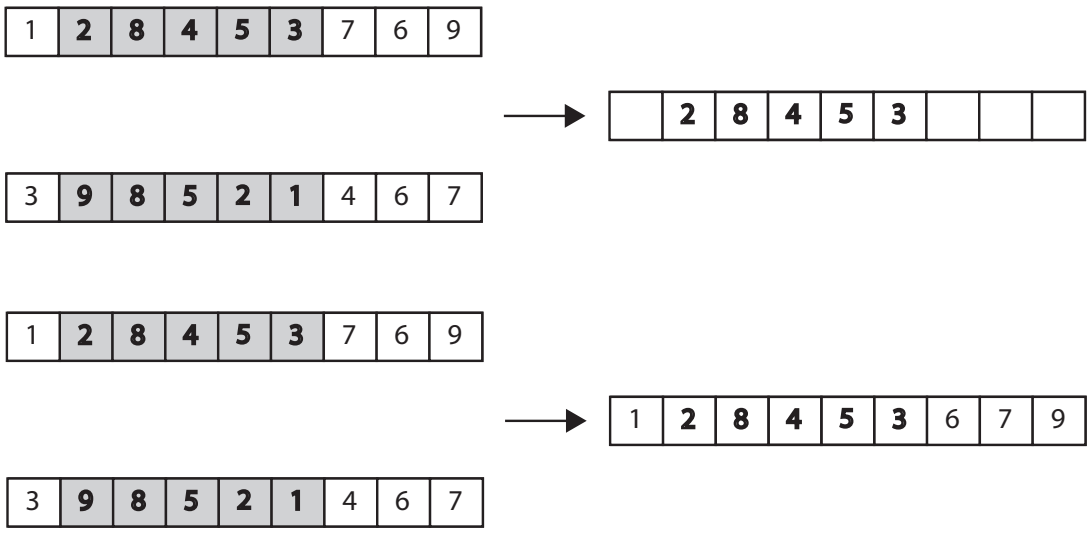

Figure 2.1: Steps of Order 1 crossover.

2.1.2.1 Order 1 Crossover

The idea of Order 1 crossover is to preserve the relative order that the genes occur in the parent chromosomes. An arbitrary part from the first parent is chosen and this part is copied to the first child with the same order as in the first parent. Then, the rest of the genes in the first parent are copied to the first child as in the order of the second parent. The second child is created analogously [59] (Figure 2.1).

2.1.2.2 Partially Mapped Crossover (*PMX*)

PMX chooses a random segment from Parent1 and copies it to the first child. Starting from the first crossover point, PMX looks for the elements in that segment of Parent2 that have not been copied. For each of these elements, say *i*, looks in the offspring to see what element *j* has been copied in its place from Parent1, PMX places *i* into the position occupied by *j* in Parent2, since we know that we will not be putting *j* there. If the place occupied by *j* in Parent2 has already been filled in the offspring by *k*, we put *i* in the position occupied by *k* in Parent2. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from Parent2. The second child is created similarly [59] (Figure 2.2).

2.1.2.3 Cycle Crossover

In cycle crossover, each gene comes from one parent together with its position. A cycle of genes from Parent1 is made by starting with the first gene, is ”1” in Figure 2.3, of Parent1. After that, the gene at the same position in Parent2, ”3”, is determined and we find the position of ”3” in Parent1, which has ”1” in its position in Parent2, thus completing the first cycle. The genes of the cycle $< 1, 3, 1 >$ are copied to the first child with their positions obtained from the first parent $< 1, \_, \_, \_, 3, \_, \_, \_ >$.

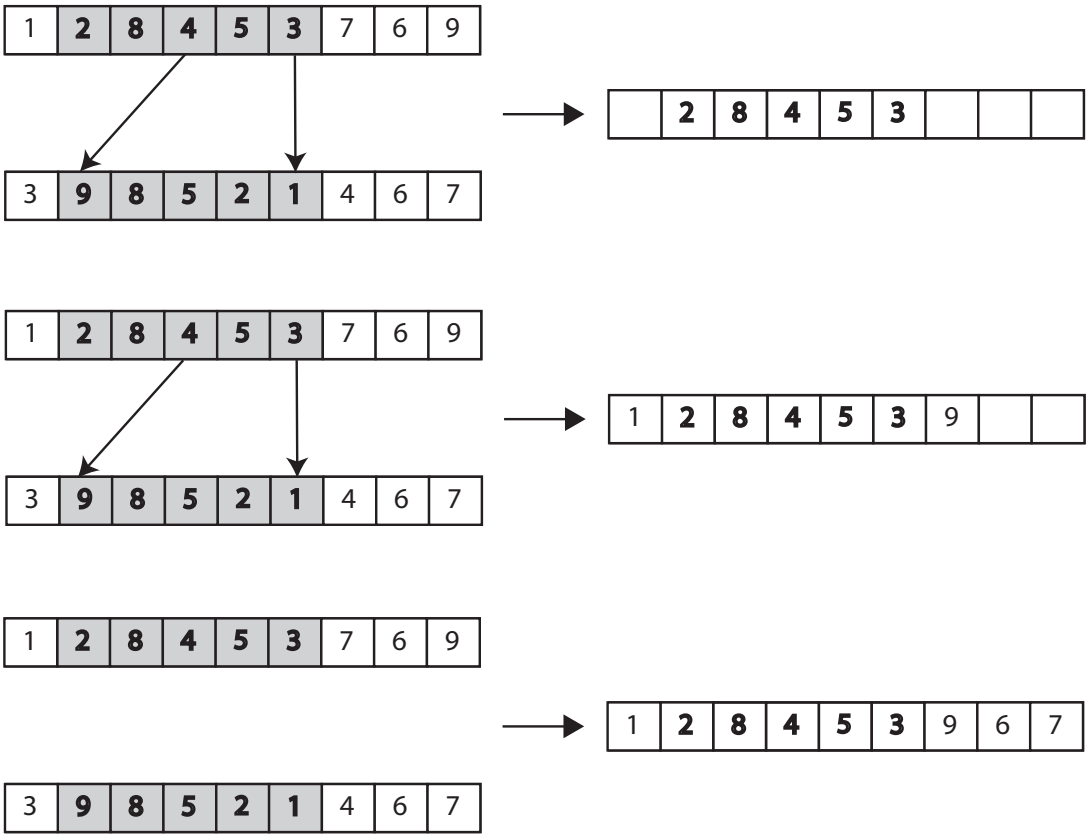

Figure 2.2: Steps of PMX crossover.

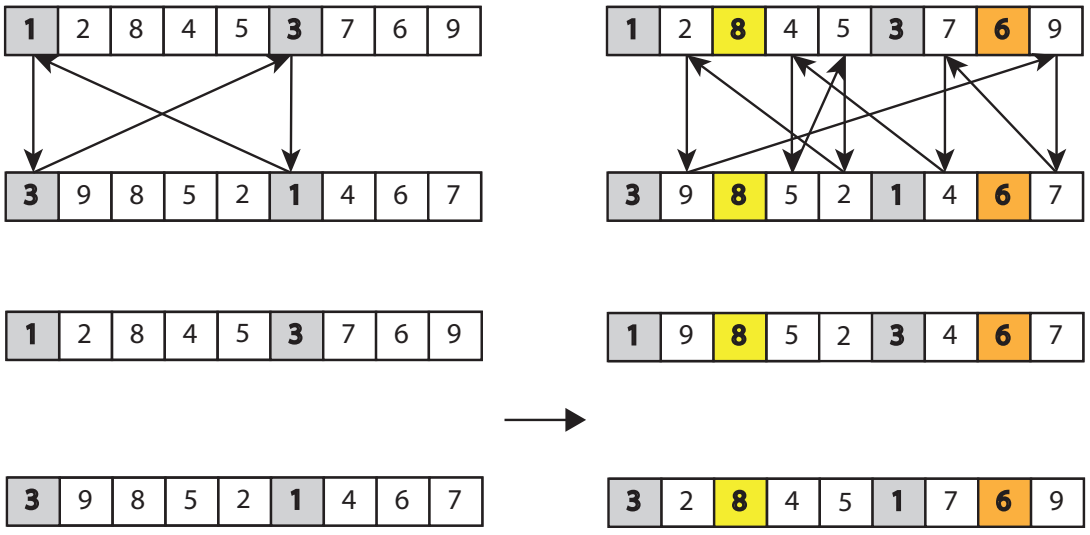

Figure 2.3: Steps of Cycle crossover.

The next cycle $< 2, 9, 7, 4, 5, 2 >$ is used to copy those genes using their positions in the second parent, $< \_, 9, \_, 5, 2, \_, 4, \_, 7 >$, $< 8 >$ and $< 9 >$ form cycles by themselves [59] (Figure 2.3).

2.1.3 Tabu Search and Multi-Start Methods

The *TS* has an adaptive memory to explore the search space and uses this memory to forbid returning to recently visited solutions. Variants of *TS* use different strategies such as diversification and intensification to focus on more promising search regions. *PHGETS* uses diversification in its *TS* diversification phase to increase performance. After designing appropriate intensification and diversification strategies to decide on a search direction, it explores the search space using *TS* which is the most time consuming phase of the *PHGETS* algorithm.

Changing the tabu list size is the basic strategy in *TS* variants. A smaller size tabu list enables the search in and around local minima, whereas bigger lists will help the search to escape from the local minima. Several aspiration criteria can be used with tabu restrictions to make a decision about a particular move. Taillard has given a *RTS* algorithm which has an efficient short-term memory and also employs multiple-level aspiration criteria. The short-term memory can help us find high quality solutions while, the longer term memory components also have promising results [86, 90]. There are many different *TS* algorithms exploring intensification and diversification strategies with a variety of parameter settings. The long-term memory generally keeps track of the frequency of the components that occur in high-quality solutions. For long-term intensifications, components with a high frequency are selected for future searches. Diversifications can eliminate these components so that the search can move into unexplored regions.

Exchanging the previous locations of two facilities with each other is a very appropriate way to generate new permutations for the neighborhood relations of the *QAP*. The computational simplicity of the swap makes it very appropriate when compared with the other larger exchange methods. In Taillard's *RTS*, these two-exchange moves save a great amount of execution time. In order to be able to calculate the cost of new permutations obtained by performing swap operations quickly, the cost of each swap operation is stored in a matrix. These costs in this matrix are calculated only once and stored, then the cost of new permutations can be calculated efficiently using these costs. The fast evaluation of the moves is an important issue that contributes to the efficiency of the search. The skeleton of the algorithm is similar to Algorithm 1.

Algorithm 1 Tabu Search Algorithm.

```
while number of failures < maximum number of failures do
  if (tabu but meets all aspiration criteria) or (not tabu and best cost ) then
    Store best exchange that meets all conditions
  end if
  Update Tabu List
  Make exchange on working solution
  if strictly improving then
    Increment the number of failures
  end if
end while
```

A variant of the simple *TS* algorithm the Robust Tabu Search (*RTS*) is one of the best known algorithms that can produce high quality solutions [141]. James *et al.* [84] introduced a multi-start *TS* algorithm (*JRG-DivTS*). When the search stagnates, *JRG-DivTS* restarts from a diversification of the best solution

found up to that point. *JRG-DivTS* uses a greedy local search that executes swap operations until a local minimum is found and also uses a multi-start approach modifying the search path by changing parameter settings while the search continues. An alternative method is to find a new solution different from the current solution and then restart the search at this new solution. This can be achieved by re-exploring a previous solution using a different search path, or restarting the search at a newly created solution. Two categories of multi start methods as *TS* variants were developed in [84]. The first category searches by modifying specific algorithm parameters, the so-called *continuous diversification*. The paper describes, a Restricted Descent *TS* (*RDTS*), and a Tabu Tenure Modification *TS* (*TTMTS*) which alter the tabu list matrix and the tabu tenure parameters. The tabu status of elements is stored in the tabu list matrix and informs the algorithm if an exchange operation is allowed or forbidden (tabu). The range of tabu tenure values applicable for an element are given by the tabu tenure parameters. The second category implements a discontinuous diversification *TS* process and contains the Random Restart TS (*RRTS*), the Best Solution Found using the *TS* (*BSFTS*) and the Diversified Best Solution was found using the *TS* (*DivTS*). The performance of the second category methods are reported to be better than the first category and *DivTS* is used to build the proposed Cooperative Parallel *TS* (*CPTS*) algorithm.

The *TS* proposed by Glover [73] starts with a steepest descent algorithm and continues by making upward and downward moves towards the solution. Earlier moves affect the future moves and moving in a reverse direction is forbidden for a certain number of moves. This way the process is forced to search through unexplored areas. When the search process gets stuck in a local minimum, it tries to find a better solution by escaping from it, even if it means moving to a worse solution (i.e. an upward move). Moves that force the search back into that local minimum are not allowed by the *TS*.

2.1.4 Sequential Metaheuristics

Many successful sequential algorithms have been proposed such as the Ant Colony Optimization [139, 67], Simulated Annealing [40] and Neural Networks [30]. The combination of population based metaheuristics and *TS* shows a hybrid behavior that may provide competitive high quality solutions.

The solution provided by *TS* procedure is combined with a so-called robust tabu search, *RTS* by Taillard [141], by Misevičius [110] while perturbing the solution via diversification operators. This algorithm effectively explores the symmetric and asymmetric instances given by Taillard from the *QAPLIB* [20]. Hybrid algorithms which exploit *RTS* like sequential metaheuristics, produce high quality solutions in combination with *GA* variants, as shown by Misevičius [110] where two *GA*s are combined with *RTS* based on diversification operators. The first algorithm performs a ruin-and-recreate strategy called *M-GA/TS* and recreates a solution by the ruin procedure. The second algorithm *M-GA/TS-I* apply a random ruin procedure to the solutions of the *GA* and creates a perturbation. Ahuja *et al.* [3] and Drezner [58] have also successfully incorporated *GA* variants into *TS*. The algorithms developed by Drezner [57, 58] perform well especially on instances given by Skorin-Kapov in the *QAPLIB*. The *D-GA/SD* algorithm developed by Drezner [57] implements a crossover operator for merging. It uses a greedy search to handle swaps to find a local optimum.

2.1.5 Parallel Metaheuristics

Sequential metaheuristics have achieved significant progress in the solution of the *QAP* and they can provide very good quality solutions. However, the complexity and computational requirements of intractable problems are very high. The granularity of the solution space, the repetitive behavior of the

metaheuristic methods, and the solution complexity make parallelization an attractive alternative for such problems. Parallelization can provide better solutions and reduce the search time required allowing new powerful algorithms to be tested and developed. As High Performance Computers (*HPC*s) become widely available, exploring the performance of parallel metaheuristics has become an important research area.

Crainic and Toulouse [48] illustrated the differences and the design choices of the parallel metaheuristic algorithms in detail. By looking at the work assigned to processors, they grouped these algorithms into three categories. A low level parallelization is deemed to be in Type 1. In this category, the master controls the task assignment among the processors and it collects the results. Type 2 reduces the exploration of solution alternatives by assigning variables to the processors for decisions. The resulting solutions are combined by a master process. Type 3 makes use of exchanging and aggregating the output that is obtained meanwhile or after the synchronous searching process. Different heuristics can be employed on different processors or the search process may start at the same or different initial solutions.

Tsutsui and Fujimoto [148] proposed a new parallel scheme using parallel graphic processing units (*GPU*s) to solve the *QAP* by combining the computational power of parallel machines with the *GA*s for solving instances up to 40 locations. James *et al.* [83] introduced a cooperative parallel *TS* algorithm (*CPTS*) for the *QAP*. The *CPTS* is shown to provide high quality solutions for all of the problem instances in the *QAPLIB* in acceptable computation times.

2.2 Distributed Database Design as Data Allocation Problem

FAP, *DAP*, and *QAP* are extensively studied well-known problems [88, 91]. There are static or dynamic allocation algorithms for the *DAP*. Static algorithms use predefined requirements, whereas dynamic algoritms take modifications into consideration [76]. Ceri and Plagatti presented a greedy algorithm for replicated and non-replicated data allocation design [27]. Bell showed that the *DAP* is NP-Hard [18]. Corcoran and Hale [41] and Frieder and Siegelmann [66] solved the *DAP* with genetic algorithms (*GA*). Ahmad and Karlapalem [2] solved the problem of non-redundant data allocation of fragments in distributed database systems by developing a query driven data allocation approach integrating the query execution strategy with the formulation of the data allocation problem [2]. Adl and Rankoohi have studied the data allocation problem without replication, but this time with site capacities limited so that only feasible allocations are considered [1]. In this study optimized execution strategies for queries and enforcement (i.e. fragment communication) costs caused by integrity constraints are also formulated into the DAP [1]. They took into consideration the query optimization and the integrity enforcement mechanisms in the formulation of the *DAP*. Many other NP-Hard problems are designed by using *QAP* [98, 54, 104].

The data allocation problem is specified with two kinds of dependencies between transactions and fragments as seen in Figure 2.4. The dependency of sites to fragments can be inferred from transaction-fragment and site-transaction dependencies. S represents sites, F represents fragments, T represents transactions in Figure 2.4. Similarly, freq is the number of requests for the execution of a transaction at a site, trfr is the direct dependency of a transaction on a fragment, and q is the indirect dependency of a transaction on two fragments.

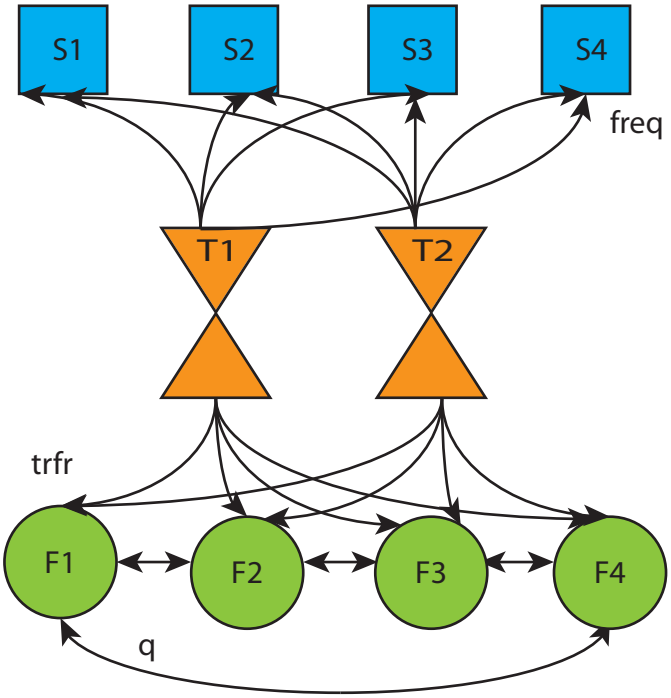

Figure 2.4: The dependencies of transactions on fragments and sites on transactions.

2.2.1 Genetic Algorithm

Genetic algorithms (GA) randomly generate an initial population of solutions, then by applying selection, crossover, and mutation operations repetitively, creates new generations [74]. The individual having the best fitness value is returned as the best solution of the problem. The termination condition may be defined depending on total execution time, number of generations produced and having no improvement in the average fitness value of the population [132], [147]. The chromosome structure of the solution can be seen in Figure 1. Facilities are placed on the array of locations. PMX crossover is used in the GA. It is one of the best performing operators for QAP solution when compared with the others. PMX copies a random segment from parent1 to the first child. It looks for the elements in that segment of parent2 that have not been copied starting from the initial crossover point. For each of these elements, say *i*, it looks in the offspring to see what element *j* has been copied in its place from parent1, PMX places *i* into the position occupied by *j* in parent2, since we know that we will not be putting *j* there. If the place occupied by *j* in parent2 has already been filled in the offspring by *k*, we put *i* in the position occupied by *k* in parent2. The rest of the offspring can be filled from parent2. The second child is created similarly [59] (Figure 2.5).

2.2.2 Fast Ant System (*FANT*)

Dorigo and colleagues proposed ACO as a method for solving difficult combinatorial problems [56]. ACO is a metaheuristic inspired by the behavior of real ants where individuals cooperate through self-organization. FANT is a method to incorporate diversification and intensification strategies [141]. It systematically reinforces the attractiveness of values corresponding to the best solution found so far the search, and on the other hand by clears the memory while giving less weight to the best solution if the process appears to be stagnating. The Ant process constructs new solutions by randomly choosing the location of facilities with a probability. Then the solutions are improved with a local search and

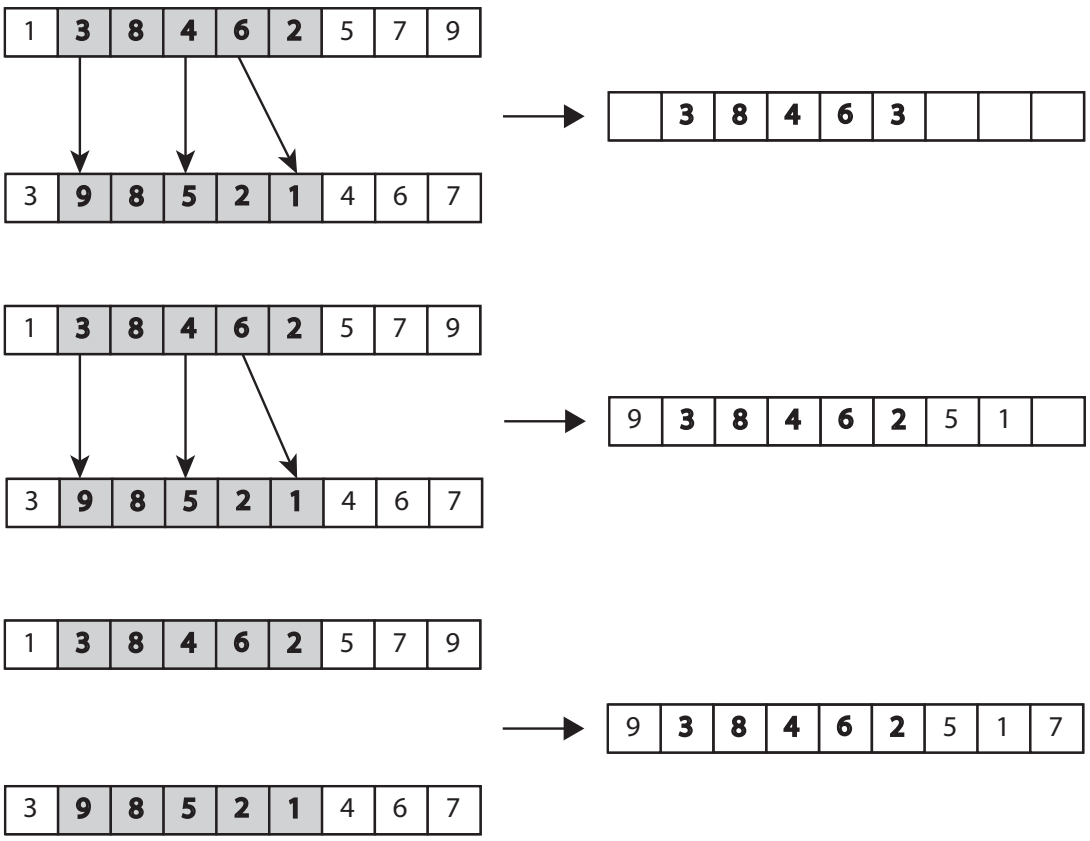

Figure 2.5: Stages of PMX Crossover.

sent to the Queen process. While implementing an automatic intensification and diversification, the Queen process requires a parameter for managing the traces. In addition to the memory matrix, the variable and the best solution found by the system so far are managed. The algorithm for a standard ant system is shown in Algorithm 2.

Algorithm 2 Standard Ant System

```
Pheromone trail is initialized
while stopping criterion is not met do
  for each ant in the colony do
    Construct a new solution with the current pheromone trail.
    Construct an evaluation of the partial solution
  end for
  Update pheromone trail.
end while
```

2.2.3 Robust Tabu Search

Robust Tabu Search (RTS) is one of the best known optimization algorithms that can produce high quality solutions [142]. It is a variant of the simple TS algorithm. Another version of the TS is the multi-start TS algorithm (JRG-DivTS) [84]. It uses a greedy local search that executes swap operations until a local minimum is found and works with a multi-start approach modifying the search path by changing parameter settings while exploration continues. An alternative method is to restart the search from a solution that differs from the current solution. This can be achieved by re-exploring a previous solution using a different search path, or restarting the search at a newly created solution. Two categories of multi start methods as TS variants are developed. The first category searches by modifying specific algorithm parameters, so-called continuous diversification. The second category implements a discontinuous diversification TS process and contains the random restart TS (RRTS), the best solution found TS (BSFTS) and the diversified best solution found TS (DivTS). The performance of the second category methods are reported to be better and DivTS is used for building the proposed Cooperative Parallel TS (CPTS) algorithm. The skeleton of the algorithm is similar to Algorithm 3.

Algorithm 3 Robust Tabu Search Algorithm [141]

Authorized: If a move is not tabu, it is authorized.
Aspired: Allow tabu moves if they are decided to be interesting.
Tabu List: A list to forbid reverse move.
Neighbor: Each location in the permutation is considered as neighbor.

```
RTS (FLOW, DIST, MaxIter, BestPerm,
      MinSize(<n×n/2), MaxSize(<n×n/2),
      Aspiration(>n×n/2)) {

TABU_LIST = {};
CurCost = QAP_Cost(BestPerm);
CurSol = BestPerm;
Delta[i][j] = ComputeDelta();// i = 0..n, j = 0..n
TABU_LIST[i][j] = - (n×i+j); // i = 0...n-1, j = 0...n-1
```

Algorithm 3 Robust Tabu Search Algorithm [141]

```
for (iteration = 1; iteration < MaxIter; iteration++){
    i_retained = infinite;
    MinDelta = infinite;
    Already_Aspired = false;
    for each Neighbor (i, j){
        current1 = TABU_LIST[i][CurSol[j]];
        current2 = TABU_LIST[j][CurSol[i]];
        Authorized = (current1 < iteration) || (current2 < iteration);
        Aspired =
                (current1 < iteration-Aspiration)||
                (current2 < iteration-Aspiration)||
                (CurCost + Delta[i][j] < BestCost);
        if (Aspired && Already_Aspired) ||
            (Aspired && Delta[i][j] < MinDelta) ||
            (!Aspired && !Already_Aspired && Delta[i][j] < MinDelta && Authorized){
            i_retained = i; j_retained = j;
            MinDelta = Delta[i][j];
            if (Aspired) Already_Aspired = true;
        }}
    if (i_retained != infinite){
        SWAP(CurSol[i_retained], CurSol[j_retained]);
        CurCost = CurCost + Delta[i_retained][j_retained];
        TABU_LIST[i_retained][CurSol[j_retained]] =
        iteration + getRandom(MinSize, MaxSize);
        TABU_LIST[j_retained][CurSol[i_retained]] =
        iteration + getRandom(MinSize, MaxSize);
        if (CurCost < BestCost)
            BestCost = CurCost;
    }
  UPDATE_MOVE_COSTS(FLOW, DIST, CurSol, Delta,
                                i, j, i_retained, j_retained);
}}
```

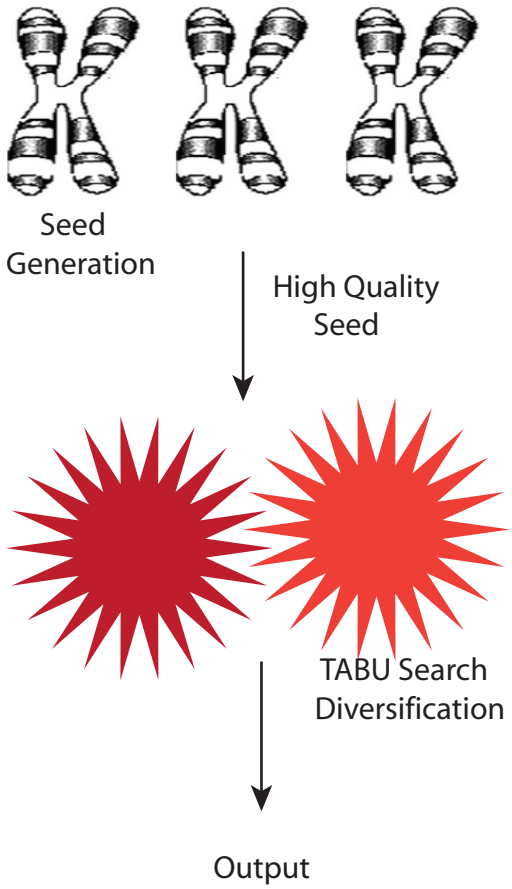

Figure 2.6: Stages of Hybrid Genetic Multi-Start Tabu Search Algorithm.

2.2.4 Hybrid Genetic Multi-Start Tabu Search Algorithm

Hybrid Genetic Multi-Start Tabu Search Algorithm (HG-MTS) is a hybrid of GA and multi-start RTS. It is a two step algorithm consisting of seed generation and TS diversification decision. Figure 2.6. shows the stages of the algorithm. The TS diversification phase uses the diversification operator of Cooperative Parallel Tabu Search (CPTS) [83]. After having decided a high quality seed, multi-start TS performs a stepwise procedure to decide the best diversification towards the solution.

2.3 Linear Programming

Linear programming is a mathematical framework to represent a set of constraints as linear relationships. It aims to produce best outcome while satisfying an objective function like maximum profit, minimum cost etc. Linear programming is also called linear programming (LP) and it is a ramification of mathematical optimization. Many problems in logistics, economics social and political sciences can be represented as linear programming problems. Knapsack-capital budgeting problem, warehouse location problem, travelling salesman problem, decreasing costs and machinery selection problem, maximum flow problems, set covering problems, matching problems, weighted matching problems, spanning trees problems can be represented as linear programming problems [35, 80, 101, 120]. Many of the operations research problems can be solved by linear programming. Network flow problems and commodity flow problems are the most common problems that can be modelled by linear program-

ming. Some algorithms solve linear programming problems as sub-problems while solving other types of problems. Linear programming is used in planning, production, transportation and basic company management activities such as minimizing the costs and resource usage. The general form of the linear programming problem is as follows:

$$Maximize \quad c^T x \tag{2.8}$$

$$subject \quad to : Ax \leq b \tag{2.9}$$

$$x \geq 0 \tag{2.10}$$

In this formulation illustrated in Equation 2.8, Equation 2.9 and Equation 2.10, x is the vector of variables to be determined, b and c are known coefficient vectors and A is the matrix of known coefficients.

2.4 A-star Algorithm for *MQO*

There are several variants of the A-Star algorithm [47, 133]. A-Star algorithm starts with an initial state having null values which means no queries have been assigned a plan. At each level of the tree, a plan is assigned to a query. Given a consistent and good cost estimation heuristic function A-Star algorithm is guaranteed to provide optimal results with a minimal number of expanded states. The number of expanded states during A-Star search determines the memory usage and the efficient and quick solution of *MQO* problem instances.

The set of tasks in the selected plans for queries Q_1 to Q_k is represented as $\bigcup_{(1\leq i\leq k)\wedge(1\leq j\leq Pi)}$. The cost estimation of task t_i is defined as $\text{estCost}(t_i) = \frac{realCost(t_i)}{m_i}$ where $t_i \in t_{sel}$ is not satisfied and zero otherwise. m_i is the number of queries waiting for a plan to be assigned and has at least one alternative plan that contains task t_i. The estimated cost for plan p_{ij} is $\text{estCost}(p_{ij}) = \sum_{t_i \in p_{ij}} \text{estCost}(t_i)$. Let Pi be the number of alternative plans for qi then the heuristic function used in A-Star algorithm is $h(Sk) = \sum_{t_x \in t_{sel}} \text{realCost}(t_x) + \sum_{k<i\leq Q} \min(\text{estCost}(p_{i,1}),...,\text{estCost}(p_{i,pi}))$.

Six alternative query ordering methods are defined to minimize the estimation error in the plan cost estimation function [130]. The selected query order remains constant throughout the search . The method ordering the queries in decreasing average query cost is experimentally shown to be the best performing [133]. This query ordering is used in this thesis too.

2.5 Dijkstra's Shortest Path Algorithm

Dijkstra's shortest path algorithm [51] is used to calculate the communication costs among sites when designing distributed databases with integer linear programming. Dijkstra's shortest path algorithm is a well known single source shortest path algorithm generally used in routing and graph based algorithms. It gets a graph with positive edge costs and produce the shortest path tree for the graph. The algorithm aims to find the path with lowest cost between two nodes namely the shortest path. The shortest path algorithm has wide usage areas in network protocols. It is also used to find costs of distances from one node to all nodes. The shortest path algorithm assigns temporary distance values first, then improve

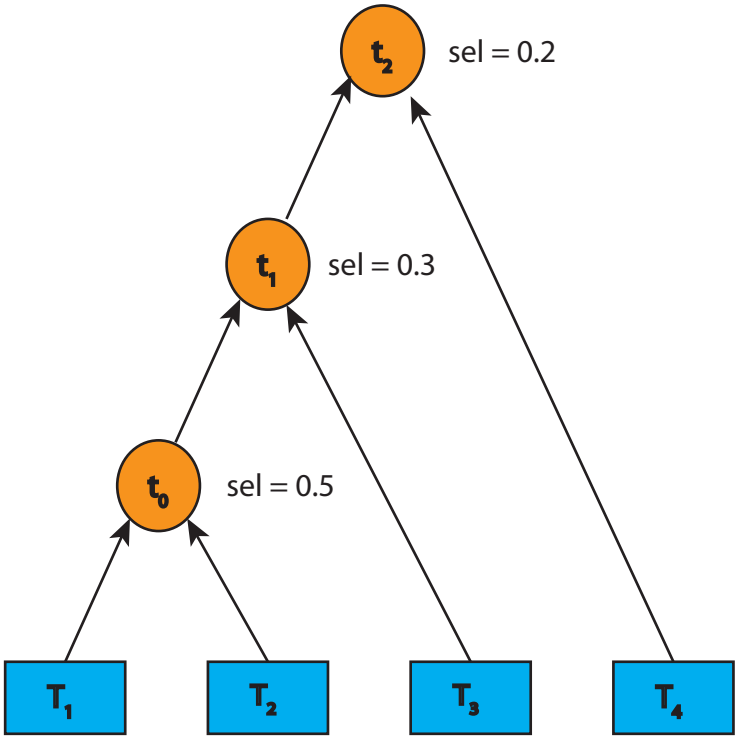

Figure 2.7: Illustration of a left deep query tree.

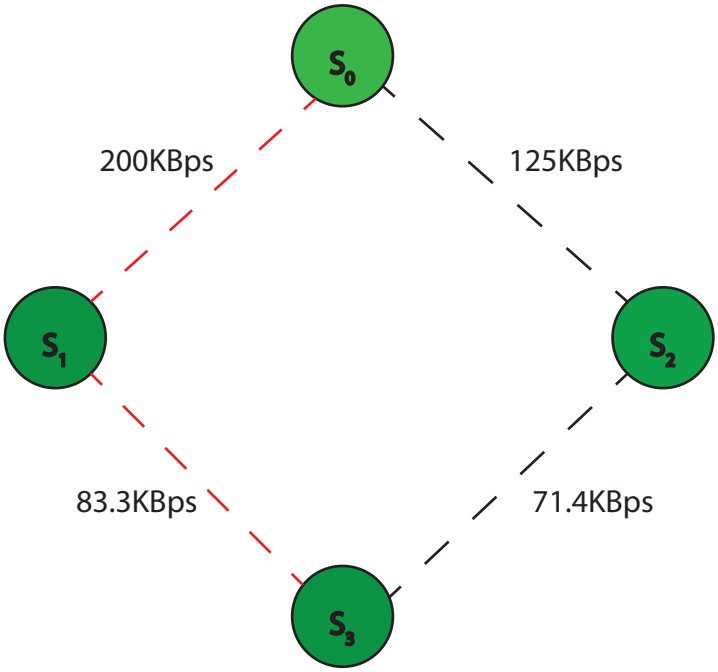

Figure 2.8: A Sample Network Topology used in our experiments.

them iteratively. Assume that we have the network topology shown in Figure 2.8 where S_0 is the initial node and S_3 is the node to calculate shortest path.

Initially, every node is assigned a temporary distance value where initial node S_0 is set to 0 and the other nodes are set to infinity. The nodes are all marked as unvisited and the initial node is set as current. A set of unvisited nodes consisting of all the nodes excluding the originating node is created. During the traversal, the temporary distances are calculated for the current node. In our example in Figure 2.8 if the current node S_3 has a distance value of 1MB / 125KBps + 1MB / 71.4KBps = 22sec. followed by the path $S_0 \rightarrow S_2$. Whenever the path $S_0 \rightarrow S_1$ is realized the distance value is set to 1MB / 200KBps + 1MB / 83.3KBps = 17sec. The new value is overwritten. A neighbor is not marked as visited unless all the neighbors are considered in the unvisited set. The prevailing node is stated to be visited and removed from unvisited node set if all the neighbors of the current node are visited. The algorithm does not again check a node which is marked as visited. The algorithm finishes when the smallest temporary distance in the unvisited node set is infinity or the destination node is marked is visited. If the algorithm is not finished, the unvisited node with smallest temporary distance is selected as the new current node. Algorithm 4 shows the pseudocode for shortest path used in our model.

Algorithm 4 Dijkstra's Shortest Path Algorithm for Graph *G* and Source *S* [51]

```
Dijkstra (G, S)
for each node N in Graph G do
   distance[N] = ∞
   previousNode[N] = unknown
end for
distance[S] = 0;
G^0 = G
while G^0 != Ø do
   g = smallest ϵ dist;
   G^0 = G^0 - g;
   if dist[g] == ∞ then
      break;
   end if
   for each g′ neighbor to g do
      alternative = dist[g] + distance{g′, g};
      if dist[g′] > alternative then
         dist[g′] = alternative;
         previousNode[g′] = g;
         order g′ in G^0
      end if
   end for
end while
```

In our distributed database design, the base tables and relations are provided as left deep trees. Figure 2.7 shows the left deep tree structure used in our model. Replica management and selectivity of data in requested amounts are possible. Site capacities exist to control distribution of data among sites. The network topology is an input to the tool. The communication costs of the source site to all sites are calculated first then cost model for the distributed database environment is generated. Dijkstra's shortest path algorithm [51] is used to calculate the distance of the query source to all sites.

2.6 Applications of Production Planning

The bullwhip effect has been a primary concern for many researchers in recent years. It was discovered by [65] and many researchers have studied to find the causes of it since then. He showed that indefinite sales occurring during the year can be converted to seasonal production activity while planning the inventory capacity efficiently. In an *SC* with many companies, the information flows through layers but only a few members of the *SC* know the primary information. Generally, the information passing through the layers is deteriorated. Inventory shortage, poor service quality, increased production costs, increased transportation costs, extra labor costs are the problems arising during this amplification of poor information [37]. According to [94], demand forecast processing, the rationing game, batch ordering and price variations are major causes of the bullwhip effect.

With the developments in the automotive industry, most of the car manufacturing companies aim to move from stock oriented production of cars through a built order production. There have been many improvements in recent years in terms of delivery times, reliability and responsiveness. Shortening the delivery times, increasing the reliability and the responsiveness perform the most of the efforts in car manufacturing optimization. Foreseen vulnerability of bullwhip effect for short-term and mid-term car manufacturing and supply chain planning are considered. The bullwhip effect occurring with respect to demand order fluctuations is transferred while moving up the supply chain. Information deterioration end to end in a supply chain causes drastic ineffectiveness and inefficiencies. Companies should understand the causes to overcome extra costs over the supply chain. A supply chain model based on the MIT Beer Game for automotive industry is discussed in the shadow of the parallel algorithms used to solve the *QAP*. Finally, a parallel genetic algorithm to reduce the bullwhip effect and cost in an automotive supply chain is proposed.

Some companies try to predict their orders with respect to sales of previous years. Forecasting data is used instead of actual demand. A major cause of using forecasting is privacy of company decisions. Most of the companies don't want to share their demand statistics publicly. However, this kind of purveyance causes the bullwhip effect as the information is distorted through amplification among the participants. Companies try to overwhelm this by using methods like exponential smoothing and moving average forecasting [36]. Companies may prefer batch ordering to provide orders less costly. A company orders large amount of supplies one month and it does not order anything for a few months. It is one of the primary causes of the bullwhip effect because forecast demand is distorted if forecasting data is used for actual demand [94, 34]. Even though this ordering strategy reduces the costs for the company, it affects other participants of the *SC*. Batch ordering may reduce the transportation costs, market risks or helps getting discounts. However, other participants will suffer because there are long no order intervals after a period of batch order.

It is important for companies to continue supply demand balance. Whenever the demand is much more than supply, manufacturer queues the customer demands [94]. Supply shortage arising due lack of production fluctuates the flow among layers of the *SC* [37]. Retailers boost orders to satisfy customer requests and they try to stock adequate amount of inventory. Majority of the problems occur when cancellation of orders occur. This causes the manufacturer to stay in a unsuitable situation. Redundant inventory stock arise due to previous batch order and cancelled customer request. Supply demand balance gets distorted due to the sudden change. Customers want to purchase products as bulk when they are cheapest. This causes amplifications in the *SC*. Promotions, discounts, tickets, gifts are the primary causes for bulk purchase orders. Companies may fix prices at a level to prevent these amplifications [94]. Discounts, gifts etc. boost demand and orders may not show the true demand. The increase in the usage of a product by a consumer because of discounts or promotions may cause

peaks in the purchase orders and these peaks may lead to amplifications through lower levels of the *SC*. Price stability may be important to persist ideal demand. These price variations due to the demand fluctuations may also trigger batch ordering.

CHAPTER 3

MULTIPLE QUERY OPTIMIZATION WITH INTEGER LINEAR PROGRAMMING

3.1 Multiple Query Optimization and Distributed Database Design

Multiple Query Optimization(*MQO*) [130] is an important database research area that has been investigated since 1980s. It aims to decrease query execution costs by evaluating the tasks shared by more than one query only once. Single query optimization processes queries solely and finds the optimum solution after evaluating alternative plans. *MQO* executes more than one queries concurrently while trying to share as many expensive tasks as possible with other queries. Processing queries one by one is inefficient when there exists a high number of common tasks, relations or predicates. *MQO* identifies common sub-expressions among queries and creates an integrated query execution plan in which common tasks are evaluated only once. There are many works since 1980s on (*MQO*) problem [130, 131, 31, 32, 64]. Depth first based approaches [75, 47] analyze common sub-expressions. A connection graph, which is an extension of a query graph [152], is used in [33]. It has been shown that *MQO* is NP-hard [130] and various levels of detail on the cost measure are criticized in [31]. A state space search formulation is defined in [130, 131]. A-star algorithm uses bounding functions and intelligent state space expansion. Several heuristics are used to persist an A-star search and cost prediction schemes were used to diminish state usage of A-star algorithm [45, 133]. Another research area has been on alternative plan generation trying to find plans that will have a higher number of common tasks [128, 49]. There are greedy heuristics for generating alternative plans [128] and heuristics using greedy algorithms with intermediate common task pipelining [49].

The *MQO* problem consists of two sub stages [31]. In the first stage, alternative plans are generated for each query respectively. Second stage exploits the plans produced in the first stage to select one optimal plan for each query. A directed acyclic graph of tasks arise to answer each query as a consequence of the second stage. It has been shown that up to ten queries near optimal plans can be generated efficiently [85]. It is possible to preserve plan quality in optimal multiple plans while generating alternative plans for a query. The second stage of the problem is more challenging since the multiple plans generated remain close to optimal as the number of queries increases. A global optimal execution plan is determined from the optimal plans for each query determined in the first stage. Using the formulation of [130] it is possible to work on both stages of the problem independently. Heuristic techniques such as A-star are very popular to solve the second stage [47, 133, 128].

The alternative plans are generated by multiple query optimizers [31] or by saving some of the best plans generated by the single query optimizer [32, 46]. Single query optimization algorithms intend to reduce the query response times after shared tasks and prediction of execution times are provided. *MQO* problem has been worked since 1980s [11, 60]. Relational query optimizers usually consider

only left-deep join trees. A single query execution plan is generated as a tree of relational algebra operations. Finding optimal plans for individual queries may not work for *MQO* since a higher cost plan might have more shared tasks with other queries, resulting in lower total execution cost for *MQO*.

We considered the below example as a group of queries to be processed to show how *MQO* processes common tasks.

Q_1: Find the names of employees older than 20 years old working in the accounting department.

Q_2: Find the names of employees having work experience of 5 years working in the accounting department.

Q_3: Find the names of employees earning more than 5000$ working in the accounting department.

The name of the department is common to all queries. Therefore, the join operation for DName = "Accounting" can be shared among whole queries. The select operations SALARY > 5000, EXPERIENCE = 5 and AGE > 20 should be delayed to share the join operation. The alternative plans for these queries are shown in Figure 3.1 and Figure 3.2. Single query optimization aims to perform select operations as soon as possible. However, it is possible to delay select operations in *MQO* unlike the conventions of single query optimization. Individual query costs increase when select operations are processed later. However, it becomes preferable to process the cost expensive common task instead of cheaper tasks once to generate alternative plans for individual queries of an *MQO* instance.

We believe that great improvements are possible in solving *MQO* problem by means of parallel processing environments and parallel heuristic algorithms [147]. High quality solutions may be obtained in shorter times by exploiting the power of parallel machines. Moreover, if *MQO* can be modelled as an integer linear programming problem, high quality solutions can be obtained with less effort by using prevailing linear programming tools. This cuts the effort and time spent for solving *MQO* problem. Linear programming is defined as the mathematical way of achieving the best results by using linear relationships. It aims to minimize an objective function subject to linear constraints. Using linear programming tools it is possible [111] to solve linear, mixed-integer linear, quadratic, mixed-integer quadratic, conic and convex nonlinear problems. It aims to solve large scale optimization problems with a high number of constraints.

3.2 General Formulation of *MQO*

An MQO problem consists of queries, plans and tasks. There is a set of n queries namely q_1 through q_n. Each query has m_i plans consisting of a set of relational algebra operations named tasks. An answer to a query is produced after executing these tasks. Tasks have individual cost values represented as positive integers. Tasks may be shared among alternative plans and queries. The solution of the *MQO* problem should minimize the total cost by selecting a set of task with a maximum number of common tasks. Only one alternative plan for each query is allowed and shared tasks are processed once during the execution. Figure 3.3 shows an example of the *MQO* problem. There are 3 queries namely q_1, q_2, q_3. Five different tasks exist in respective queries. Tasks are represented as t_i and their cost values are represented as c_i. In our example tasks are t_0, t_1, t_2, t_3 and t_4. They have corresponding costs of c_0 = 3, c_1 = 3, c_2 = 1, c_3 = 1 and c_4 = 3. There are seven different plans. Two of these plans are for q_1 and q_3. Three of these plans are for q_2. Each plan is represented as p_{ij} which means it is the j'th plan for i'th query. The plans for the first query are $p_{11} = \{t_0, t_1, t_2\}$ and $p_{12} = \{t_0, t_3\}$. The plans for the second query are $p_{21} = \{t_2, t_3\}$, $p_{22} = \{t_4\}$ and $p_{23} = \{t_0, t_1, t_3\}$. The plans for the third query are $p_{31} = \{t_0, t_1\}$ and $p_{32} = \{t_2\}$

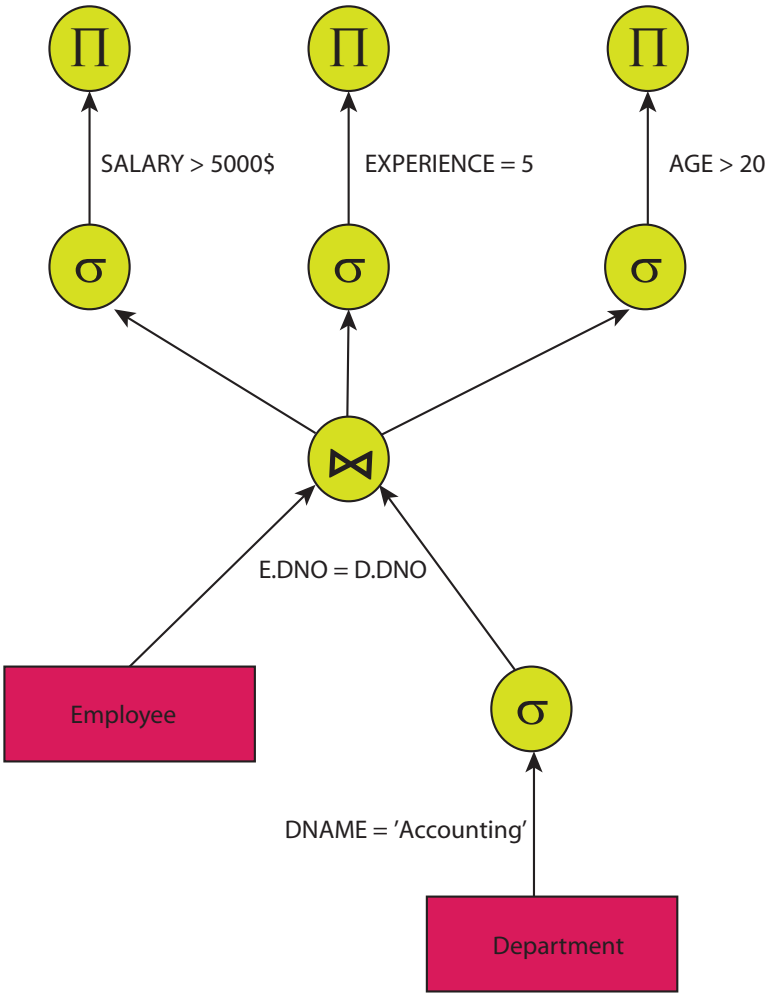

Figure 3.1: Multiplan 1 for processing queries Q_1, Q_2 and Q_3.

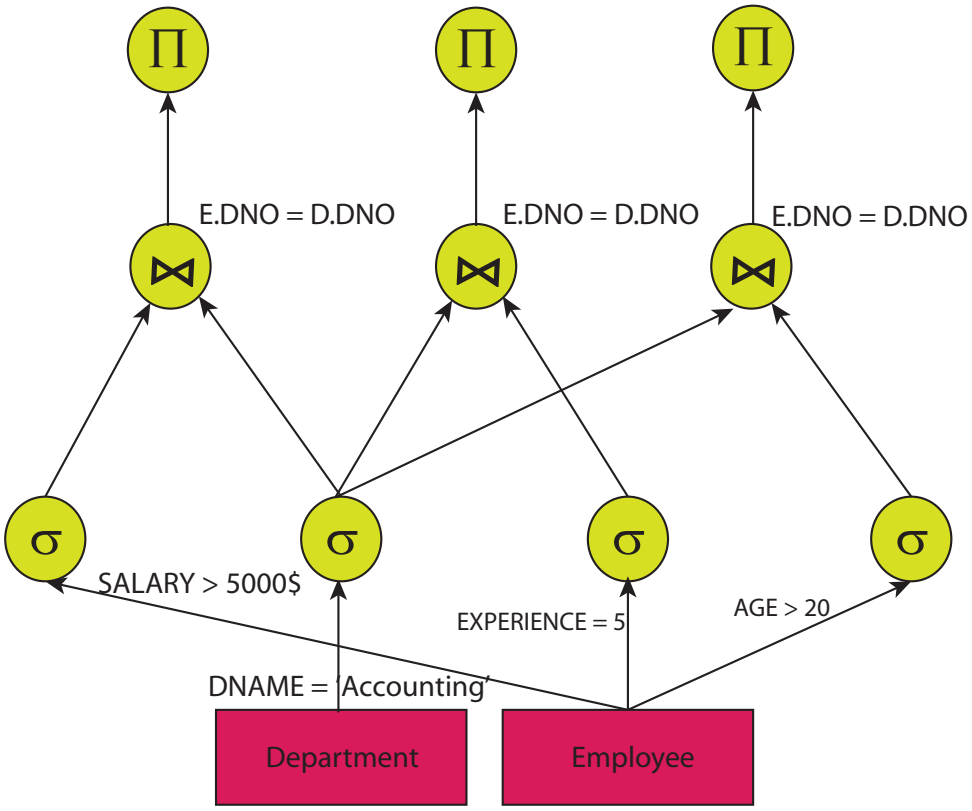

Figure 3.2: Multiplan 2 for processing queries Q_1, Q_2 and Q_3.

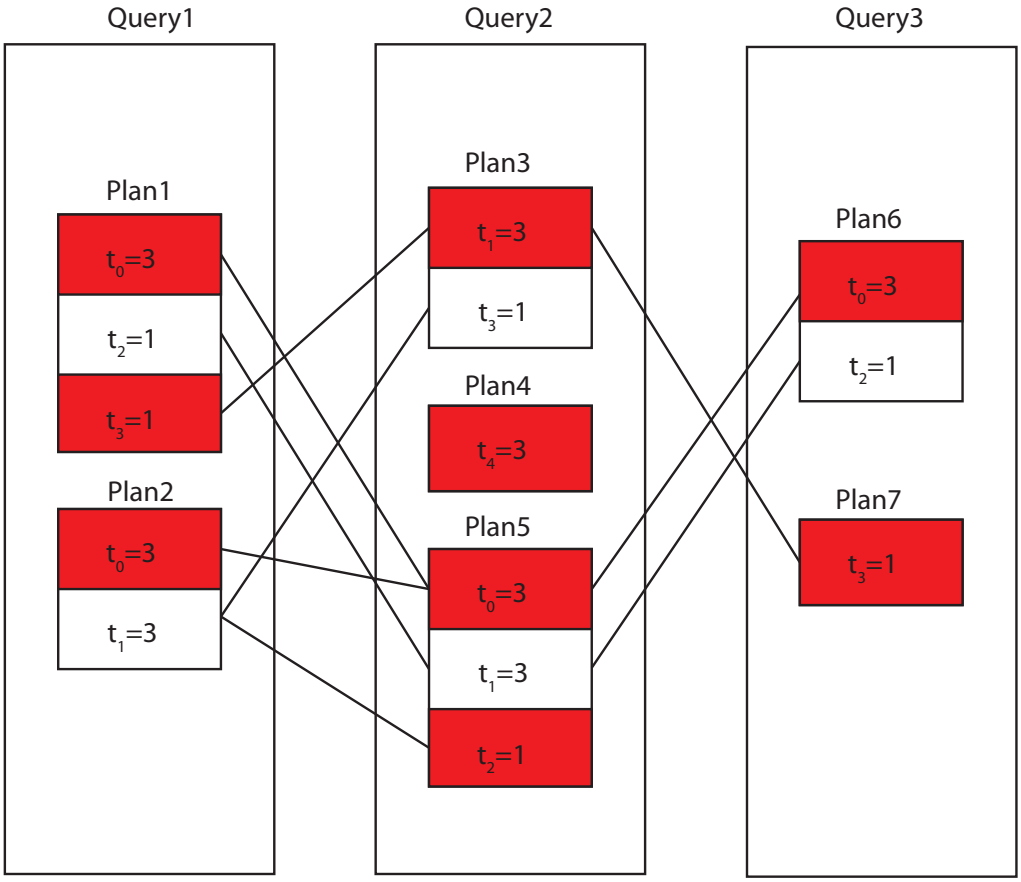

Figure 3.3: Sample *MQO* problem with 3 queries, 7 plans and 5 tasks.

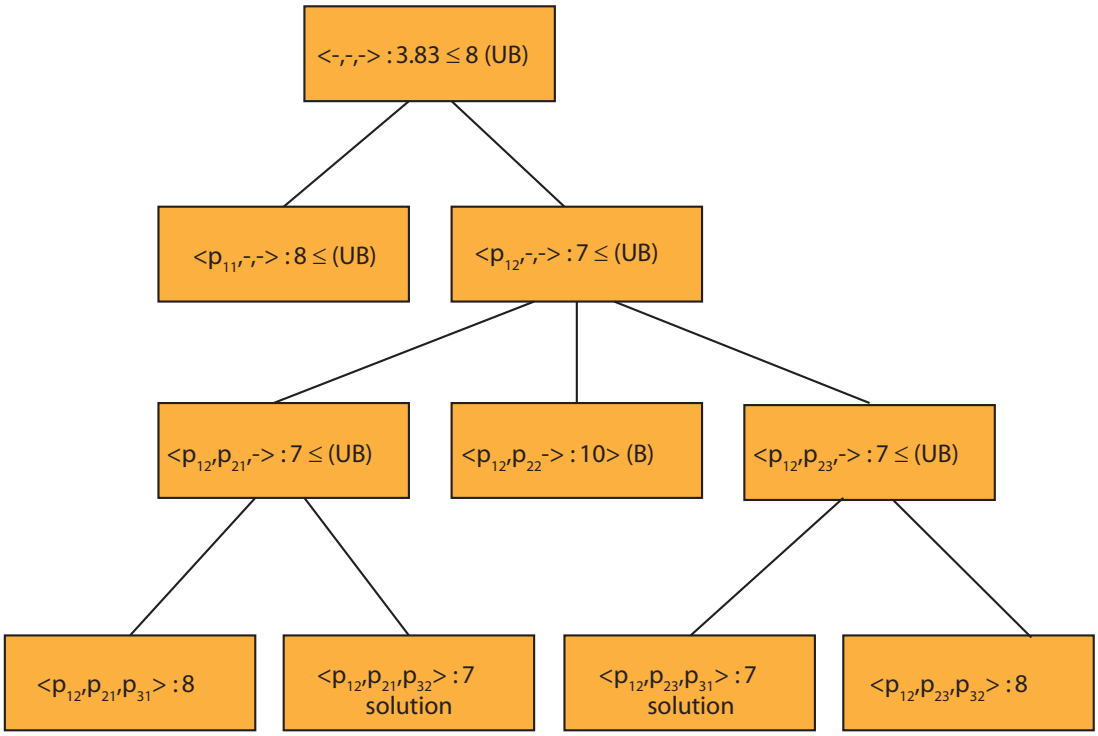

Figure 3.4: A-star Heuristic Search Tree [16]

3.3 A-star Algorithm for *MQO* (Sellis, 1998)

Table 3.1 shows the execution of the A-star heuristic for the example given in Figure 3.3. Estimation for the initial cost is 3.83. Initial estimated cost is determined as 3.83. For the example used the value of the upper bound is 8 and finding the minimum cost for the plans for each query is an upper bound of the cost for evaluating the whole plan. The upper bound should be less than or equal to the sum of task costs of the selected plans. If it is less than the sum, this means that plans share common tasks. A branch is pruned if an expanded state representing a partial or complete solution costs greater than 8. Figure 3.4 shows the corresponding search tree for solving our *MQO* instance.

3.4 Integer Linear Programming for *MQO*

The *MQO* problem instance in Figure 3.3 can be formulated as an integer linear programming problem as follows

$$x_1 + x_2 = 1 \tag{3.1}$$

$$x_3 + x_4 + x_5 = 1 \tag{3.2}$$

<-, -, ->	3.83	expanded
<p_{11}, -, ->	8	in expansion list
<p_{12}, -, ->	7	expanded
<p_{12}, p_{21}, ->	7	expanded
<p_{12}, p_{22}, ->	10	pruned
<p_{12}, p_{23}, ->	7	expanded
<p_{12}, p_{21}, p_{31}>	8	not-solution
<p_{12}, p_{23}, p_{32}>	7	solution
<p_{12}, p_{23}, p_{31}>	7	solution
<p_{12}, p_{23}, p_{32}>	8	not-solution

.

Table3.1: Execution of the A-star Heuristic Algorithm

$$x_6 + x_7 = 1 \tag{3.3}$$

$$-5x_1 + 3x_8 + 1x_{10} + 1x_{11} >= 0 \tag{3.4}$$

$$-6x_2 + 3x_8 + 3x_9 >= 0 \tag{3.5}$$

$$-4x_3 + 3x_9 + 1x_{11} >= 0 \tag{3.6}$$

$$-3x_4 + 3x_{12} >= 0 \tag{3.7}$$

$$-7x_5 + 3x_8 + 3x_9 + 1x_{10} >= 0 \tag{3.8}$$

$$-4x_6 + 3x_8 + 1x_{10} >= 0 \tag{3.9}$$

$$-1x_7 + 1x_{11} >= 0 \tag{3.10}$$

objective function:

$$3x_8 + 3x_9 + 1x_{10} + 1x_{11} + 3x_{12} \tag{3.11}$$

Plans are represented by variables x_1 through x_7 in the above equations. Q_1 has plan variables x_1 and x_2, Q_2 has plan variables x_3, x_4 and x_5, and finally Q_3 has plan variables x_6 and x_7. Equation 3.1, 3.2 and 3.3 represent the constraint that only one plan can be chosen for each query. Each task is assigned a variable x_8 through x_{12} to represent whether that task is included in the global query plan. Each plan has an associated constraint represented by the sum of the costs of that plan's tasks multiplied

by task variables minus the sum of task costs multiplied by the plan variable. Equations 3.4-3.10 represent the constraints that if a plan is selected for a query, all of the tasks in that plan must also be selected. For example, Equation 3.4 is for Plan1 of Q_1, and tasks of Plan1 are t_0 (cost=3), t_2 (cost=1), t_3 (cost=1) and are coded by variables x_8, x_{10}, and x_{11}, respectively. By multiplying each task cost by its variables we obtain $3x_8+1x_{10}+1x_11$, and this sum subtracted from $5x_1$ (meaning Plan1 is selected) must be equal to zero. Thus, for each plan in the *MQO* problem instance we produce an equation giving Equations 3.4-3.10.

Finally, Equation 3.11 is the objective function that will be used for minimizing the total cost of selected plans/tasks.

CHAPTER 4

DISTRIBUTED DATABASE DESIGN WITH QUADRATIC ASSIGNMENT AND INTEGER LINEAR PROGRAMMING

4.1 Distributed Database Design and Integer Linear Programming

In our model, we first calculate all the distances of the source nodes to all other nodes in the network by using Dijkstra's shortest path algorithm. Then, the next step is to find such an assignment of database tables/fragments to distributed nodes that the total cost of executing all queries becomes minimal. We assume that the input queries are all left deep trees containing base tables as leaves of the tree. The execution model is discussed using the query trees given in Figure 4.1. All the base tables are represented by capital letter T and in our examples we use T_0 to T_n. The relations are represented by t and the intermediate results of operating the relations are represented as t_0 through t_n. There exists a selectivity factor for each relation which represents the join output size after the operation. There also exists selectivity factors for the base tables prior to applying the join operations which are used for specifying a query which inputs only a subset of a table.

We considered two types of network models in our examples. The first one consists of different link communication speeds whereas the second one consists of uniform communication speeds. Figure 4.2 and Figure 4.3 show both of the topologies used in our examples. In these examples, there are four sites S_0, S_1, S_2 and S_3. The sites have capacities C_0 = 14MB, C_1 = 8MB, C_2 = 6MB and C_3 = 5MB. Links have communication speeds of 200KBps, 125KBps, 83.3KBps and 71.4KBps for Figure 4.2 and uniform link speeds of 200KBps for Figure 4.3. The queries to be executed are shown in Figure 4.1.

There are four sites and four base tables. We used MOSEK [111] linear programming tool to model network topology, communication costs, and assignment of table/fragments to distributed database nodes. Appendix D-Table D.7 shows the Integer Linear Programming model corresponding to the set of queries given in Figure 4.2. In order to represent the assignments of base tables to sites, we use the formalization in Table 4.2. The total number of variables is 16 for site-table assignments given for this example problem instance. There are 44 constraints and 8 of them are for the constraints stating whether replicas are allowed for each relation by giving the replica count as a constraint (e.g. 1 means no replication for the corresponding table). Next, 4 constraints are given to make sure that the total storage requirements for tables assigned to particular sites doesn't exceed the storage capacities of each site.

$$size(t') = size(T_0) * Selectivity(T_0, Query1) \tag{4.1}$$

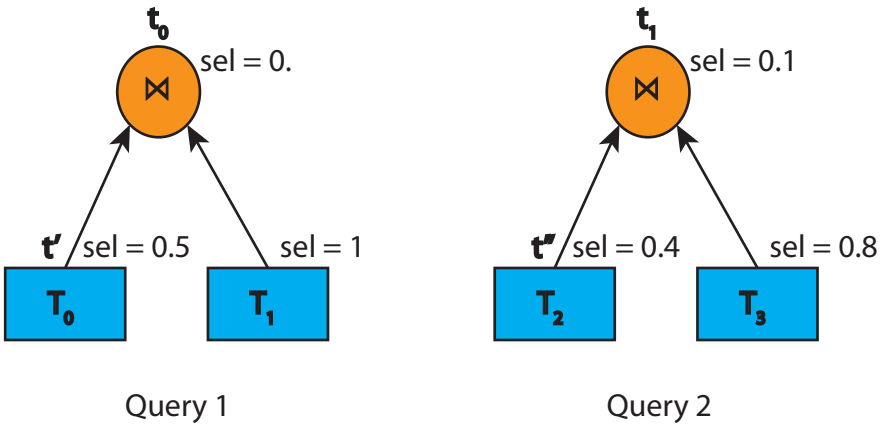

Figure 4.1: Query trees used in our examples.

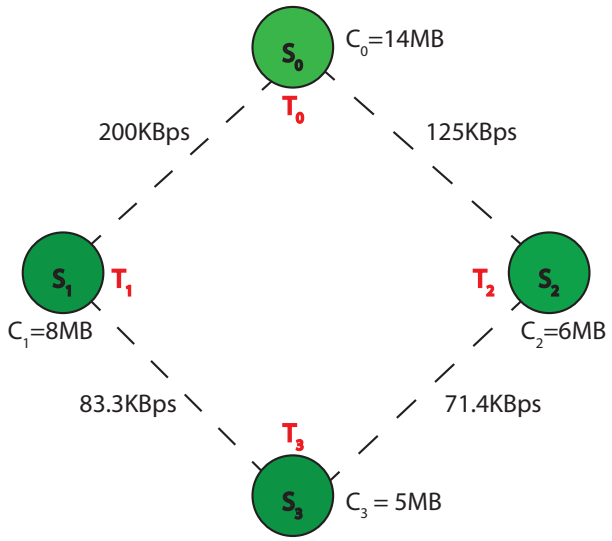

Figure 4.2: Distributed Database System with site capacities of 14MB, 8MB, 6MB and 5MB and communication links of 200KB, 83.3KB, 125KB, 71.4KB respectively.

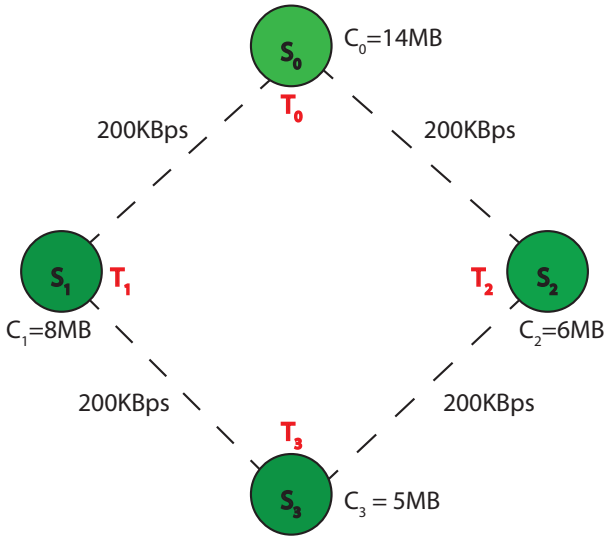

Figure 4.3: Distributed Database System with site capacities of 14MB, 8MB, 6MB and 5MB and communication links of 200KB respectively.

$$size(t_0) = size(t') * size(T_0) * joinSelectivity(T_0, T_1) \tag{4.2}$$

$$UpdateCost(T_i) = size(T_i) * UpdatePercentage(T_i) * UpdateCostRatio(T_i) \tag{4.3}$$

MB * % * seconds/ MB = seconds

There are 32 equations used for specifying the nodes that perform each operation of given queries and the objective function includes the communication cost for each possible selected path. There are 3 queries used in our examples. Query 1 executes 100 times from originating site S_0 whereas Query 1 executes 5 times from originating site S_3. Query 2 executes 20 times from site S_2. In our examples we consider at most 2 replicas for all of the tables. Update ratios' parameters are selected as 0.5, 0.2, 0.1 and 0.05 for tables T_0-T_3. Update costs are calculated according to Equation 4.3. The objective function of the optimization problem is to minimize the sum of costs of transmitting base tables and intermediate results used by queries to sites S_0, S_1, S_2 and S_3 while executing the queries. Table size for T_0 = 10MB and T_1 = 8MB which gives 10MB $\times$ 0.5 = 5MB for T_0, and 8MB $\times$ 1 = 8MB for T_1 where 0.5 and 1 are the table selectivity values. When performing the join operation, the resulting intermediate relation t_0 is calculated as 5MB $\times$ 8MB $\times$ 0.3 = 12MB where 0.3 is the join selectivity. Similar to Query 1, Query 2 has two tables T_2 = 6MB and T_3 = 5MB. This results 6MB $\times$ 0.4 = 2.4MB and 5MB $\times$ 0.8 = 4MB where 0.4 and 0.8 are the table selectivity values. When performing the join operation, size of the resulting intermediate relation t_1 is calculated as 2.4MB $\times$ 4MB $\times$ 0.1 = 0.96MB where 0.1 is the join selectivity. Equation 4.1 shows the size of intermediate result after base table selectivity and Equation 4.2 shows the size after join operation is applied. Table 4.1 summarizes the intermediate table size calculation, update cost calculation operations and formulates the total cost.

Equation 4.4 and Equation 4.5 show the replication formulation. There are a total of 96 variables used to represent the optimization problem as an Integer Linear Programming model. 16 of the values

Table4.1: Calculation of Optimization Parameters and Objective Function Calculation

Intermediate Table Sizes
t’ = 10MB×0.5 = 5MB t_0 = t’(5MB)×8MB×1×0.3 = 12MB t” = 6MB×0.4 = 2.4MB t_1 = t”(2.4MB)×5MB×0.8 × 0.1 = 0.96MB
Update Costs (ratio unit: seconds / MB)
0.5×10MB = 5 0.2×8MB = 1.6 0.1×6MB = 0.6 0.05×5MB = 0.25
Objective Function
$TotalCost = \sum_{i=0}(TableSize_i * UpdateCost_i) + \sum_{S_x}\sum_{Q_i}\sum_{T_y} CommCost(S_xQ_iT_y)$

Site Assignment Constraints: each table must be assigned to at least one site:

$$\left.\sum_{i=1}^{numSites} x_{(i+numSites*j)} \geq 1\right\} \qquad j\epsilon\{0, ..., numTables - 1\} \tag{4.4}$$

Number of Replication Constraints for each Table:

$$\left.\sum_{i=1}^{numSites} x_{(i+numSites*j)} \geq numReplicas(Table_j)\right\} \qquad j\epsilon\{0, ..., numTables - 1\} \tag{4.5}$$

Table4.2: Table to Site assignment variables used in our model.

Table \ Site	S_0	S_1	S_2	S_3
T_0	x_1	x_2	x_3	x_4
T_1	x_5	x_6	x_7	x_8
T_2	x_9	x_{10}	x_{11}	x_{12}
T_3	x_{13}	x_{14}	x_{15}	x_{16}

Table4.3: Table to Site Assignments for T_0 and t’

T_0 \ t’	S_0	S_1	S_2	S_3
S_0	x_{17}	x_{18}	x_{19}	x_{20}
S_1	x_{21}	x_{22}	x_{23}	x_{24}
S_2	x_{25}	x_{26}	x_{27}	x_{28}
S_3	x_{29}	x_{30}	x_{31}	x_{32}

Table4.4: Table to Site Assignments for t' and T_1

t' \ T_1	S_0	S_1	S_2	S_3
S_0	x_{33}	x_{34}	x_{35}	x_{36}
S_1	x_{37}	x_{38}	x_{39}	x_{40}
S_2	x_{41}	x_{42}	x_{43}	x_{44}
S_3	x_{45}	x_{46}	x_{47}	x_{48}

Table4.5: Table to Site Assignments for t' and T_1

t' \ T_1	S_0	S_1	S_2	S_3
S_0	x_{49}	x_{50}	x_{51}	x_{52}
S_1	x_{53}	x_{54}	x_{55}	x_{56}
S_2	x_{57}	x_{58}	x_{59}	x_{60}
S_3	x_{61}	x_{62}	x_{63}	x_{64}

Table4.6: Table to Site Assignments for T_2 and t"

T_2 \ t"	S_0	S_1	S_2	S_3
S_0	x_{65}	x_{66}	x_{67}	x_{68}
S_1	x_{69}	x_{70}	x_{71}	x_{72}
S_2	x_{73}	x_{74}	x_{75}	x_{76}
S_3	x_{77}	x_{78}	x_{79}	x_{80}

Table4.7: Table to Site Assignments for t" and T_3.

t" \ T_3	S_0	S_1	S_2	S_3
S_0	x_{81}	x_{82}	x_{83}	x_{84}
S_1	x_{85}	x_{86}	x_{87}	x_{88}
S_2	x_{89}	x_{90}	x_{91}	x_{92}
S_3	x_{93}	x_{94}	x_{95}	x_{96}

Site Capacities:

$$\left.\sum_{i=0}^{numTables-1} T_i x_{(i*numSites)+j+1} \leq C_j,\ j\epsilon\{0,...,numSites-1\}\right\} 1\ constraint\ for\ each\ site \quad (4.14)$$

represent table to site assignments as shown in Table 4.2 whereas 80 of the variables represent the communication costs as shown in tables Table 4.3, Table 4.5, Table 4.6 and Table 4.7.

There are 8 equations corresponding to replication formulation. Equation 4.6 through Equation 4.9 represent the minimum number of tables to be inserted to sites. We used these constraints since integer linear programming aims to set variables to 0 otherwise. Equation 4.10 through Equation 4.13 represent the maximum number of tables to be inserted to sites. Generally the system tries to use replicas when update costs are zero. Each query tries to exploit the tables it uses on its originating site.

$$x_1 + x_2 + x_3 + x_4 >= 1 \quad (4.6)$$

$$x_5 + x_6 + x_7 + x_8 >= 1 \quad (4.7)$$

$$x_9 + x_{10} + x_{11} + x_{12} >= 1 \quad (4.8)$$

$$x_{13} + x_{14} + x_{15} + x_{16} >= 1 \quad (4.9)$$

$$x_1 + x_2 + x_3 + x_4 <= 2 \quad (4.10)$$

$$x_5 + x_6 + x_7 + x_8 <= 2 \quad (4.11)$$

$$x_9 + x_{10} + x_{11} + x_{12} <= 2 \quad (4.12)$$

$$x_{13} + x_{14} + x_{15} + x_{16} <= 2 \quad (4.13)$$

4.1.1 Site Capacity Constraints

Site capacities are represented with Equation 4.15 to Equation 4.18. Equation 4.14 shows the site capacity formulation.

$$10x_1 + 8x_5 + 6x_9 + 5x_{13} <= 14 \quad (4.15)$$

$$10x_2 + 8x_6 + 6x_{10} + 5x_{14} <= 8 \quad (4.16)$$

$$10x_3 + 8x_7 + 6x_{11} + 5x_{15} <= 6 \quad (4.17)$$

$$10x_4 + 8x_8 + 6x_{12} + 5x_{16} <= 5 \quad (4.18)$$

4.1.2 Communication Cost Constraints

The communication costs are the most important part of the system. The most important issue with the communication cost variables is that the variable selection for the parts of the query should be consistent. When we consider Query 1, it originates from sites S_0 and S_3 thus only the first part is the same with each other. For Query 1 originating from S_0 is expressed by equations Equation 4.19 to Equation 4.30.

$$-x_1 + x_{17} + x_{18} + x_{19} + x_{20} = 0 \tag{4.19}$$

$$-x_2 + x_{21} + x_{22} + x_{23} + x_{24} = 0 \tag{4.20}$$

$$-x_3 + x_{25} + x_{26} + x_{27} + x_{28} = 0 \tag{4.21}$$

$$-x_4 + x_{29} + x_{30} + x_{31} + x_{32} = 0 \tag{4.22}$$

$$-x_{17} - x_{21} - x_{25} - x_{29} + x_{33} + x_{34} + x_{35} + x_{36} = 0 \tag{4.23}$$

$$-x_{18} - x_{22} - x_{26} - x_{30} + x_{37} + x_{38} + x_{39} + x_{40} = 0 \tag{4.24}$$

$$-x_{19} - x_{23} - x_{27} - x_{31} + x_{41} + x_{42} + x_{43} + x_{44} = 0 \tag{4.25}$$

$$-x_{20} - x_{24} - x_{28} - x_{32} + x_{45} + x_{46} + x_{47} + x_{48} = 0 \tag{4.26}$$

$$-x_5 + x_{33} + x_{37} + x_{41} + x_{45} = 0 \tag{4.27}$$

$$-x_6 + x_{34} + x_{38} + x_{42} + x_{46} = 0 \tag{4.28}$$

$$-x_7 + x_{35} + x_{39} + x_{43} + x_{47} = 0 \tag{4.29}$$

$$-x_8 + x_{36} + x_{40} + x_{44} + x_{48} = 0 \tag{4.30}$$

Query 1 originating from S_3 is expressed by Equation 4.19 to Equation 4.22 and Equation 4.31 to Equation 4.38. First part of the queries are the same thus first 4 equations are considered to be shared.

$$-x_{17} - x_{21} - x_{25} - x_{29} + x_{49} + x_{50} + x_{51} + x_{52} = 0 \tag{4.31}$$

$$-x_{18} - x_{22} - x_{26} - x_{30} + x_{53} + x_{54} + x_{55} + x_{56} = 0 \tag{4.32}$$

$$-x_{19} - x_{23} - x_{27} - x_{30} + x_{54} + x_{55} + x_{56} + x_{57} = 0 \tag{4.33}$$

$$-x_{20} - x_{24} - x_{28} - x_{31} + x_{55} + x_{56} + x_{57} + x_{58} = 0 \tag{4.34}$$

$$-x_5 + x_{49} + x_{53} + x_{57} + x_{61} = 0 \tag{4.35}$$

$$-x_6 + x_{50} + x_{54} + x_{58} + x_{62} = 0 \tag{4.36}$$

$$-x_7 + x_{51} + x_{55} + x_{59} + x_{63} = 0 \tag{4.37}$$

$$-x_8 + x_{52} + x_{56} + x_{60} + x_{64} = 0 \tag{4.38}$$

Similar to Query 1, Query 2 is represented by Equation 4.39 through Equation 4.50

$$-x_9 + x_{65} + x_{66} + x_{67} + x_{68} = 0 \tag{4.39}$$

$$-x_{10} + x_{69} + x_{70} + x_{71} + x_{72} = 0 \tag{4.40}$$

$$-x_{11} + x_{73} + x_{74} + x_{75} + x_{76} = 0 \tag{4.41}$$

$$-x_{12} + x_{77} + x_{78} + x_{79} + x_{80} = 0 \tag{4.42}$$

$$-x_{65} - x_{69} - x_{73} - x_{77} + x_{81} + x_{82} + x_{83} + x_{84} = 0 \tag{4.43}$$

$$-x_{66} - x_{70} - x_{74} - x_{78} + x_{85} + x_{86} + x_{87} + x_{88} = 0 \tag{4.44}$$

$$-x_{67} - x_{71} - x_{75} - x_{79} + x_{89} + x_{90} + x_{91} + x_{92} = 0 \tag{4.45}$$

$$-x_{68} - x_{72} - x_{76} - x_{80} + x_{93} + x_{94} + x_{95} + x_{96} = 0 \tag{4.46}$$

$$-x_{13} + x_{81} + x_{85} + x_{89} + x_{93} = 0 \tag{4.47}$$

$$-x_{14} + x_{82} + x_{86} + x_{90} + x_{94} = 0 \tag{4.48}$$

$$-x_{15} + x_{83} + x_{87} + x_{91} + x_{95} = 0 \tag{4.49}$$

$$-x_{16} + x_{84} + x_{88} + x_{92} + x_{96} = 0 \tag{4.50}$$

The objective function which consists of update costs and communication costs are calculated as follows for Figure 4.2. The communication links have costs 5 sec. for S_0-S_1, 12 sec. for S_1-S_3, 8 sec. for S_0-S_2 and finally 14 sec. for S_2-S_3 links. These costs are average costs to transfer 1 MB of data between two sites. We know that the update ratios for the respective tables are 0.5, 0.2, 0.1 and 0.05.

Table4.8: Update Costs in the example (Appendix D-Table D.1).

Table \ Site	S_0	S_1	S_2	S_3
T_0	5sec	5sec	5sec	5sec
T_1	1.6sec	1.6sec	1.6sec	1.6sec
T_2	0.6sec	0.6sec	0.6sec	0.6sec
T_3	0.25sec	0.25sec	0.25sec	0.25sec

Table4.9: Calculated Communication Costs for Assignments of t' and T_0 to Sites S_0-S_3 (Appendix D-Table D.2).

T_0 \ t'	S_0	S_1	S_2	S_3
S_0	0sec	2625sec	4200sec	8925sec
S_1	2625sec	0sec	6825sec	6300sec
S_2	4200sec	6825sec	0sec	7350sec
S_3	8925sec	6300sec	7350sec	0sec

Table4.10: Query Result Communication Cost of Query-1 originating from S_0 (assumed to be performed at site(T_1)) (Appendix D-Table D.3).

t' \ T_1	S_0	S_1	S_2	S_3
S_0	0sec	8500sec	13600sec	28900sec
S_1	2500sec	6000sec	16100sec	26400sec
S_2	4000sec	12500sec	9600sec	27400sec
S_3	8500sec	12000sec	16600sec	20400sec

Table4.11: Query Result Communication Cost of Query-1 originating from S_3 (assumed to be performed at site(T_1))(Appendix D-Table D.4).

t' \ T_1	S_0	S_1	S_2	S_3
S_0	1020sec	845sec	1040sec	425sec
S_1	1145sec	720sec	1165sec	300sec
S_2	1220sec	1045sec	840sec	350sec
S_3	1445sec	1020sec	1190sec	0 sec

Table4.12: Calculated Communication Costs for Assignments of t" and T_2 to Sites S_0-S_3 (Appendix D-Table D.5).

T_2 \ t"	S_0	S_1	S_2	S_3
S_0	0sec	240sec	384sec	816sec
S_1	240sec	0sec	624sec	576sec
S_2	384sec	624sec	0sec	672sec
S_3	816sec	576sec	672sec	0sec

Table4.13: Query Result Communication Cost of Query-2 originating from S_2 (assumed to be performed at site(T_3)) (Appendix D-Table D.6).

T_3 / t”	S_0	S_1	S_2	S_3
S_0	153.6sec	489.6sec	384sec	1084.8sec
S_1	393.6sec	249.6sec	624sec	844.8sec
S_2	537.6sec	873.6sec	0sec	940.8sec
S_3	969.6sec	825.6sec	672sec	268.8sec

Table 4.8 through Table 4.13 show the cost representations for the communication for the topology of Figure 4.2. Finally, the objective function for Figure 4.2 is Equation 4.51. The objective function for Figure 4.3 is expressed similarly and the objective function is shown in Equation 4.52.

$$
\begin{aligned}
&5x_1 + 5x_2 + 5x_3 + 5x_4 + 1.6x_5 + 1.6x_6 + 1.6x_7 + 1.6x_8 + 0.6x_9 + \\
&0.6x_{10} + 0.6x_{11} + 0.6x_{12} + 0.25x_{13} + 0.25x_{14} + 0.25x_{15} + 0.25x_{16} + \\
&0x_{17} + 2625x_{18} + 4200x_{19} + 8925x_{20} + 2625x_{21} + 0x_{22} + 6825x_{23} + \\
&6300x_{24} + 4200x_{25} + 6825x_{26} + 0x_{27} + 7350x_{28} + 8925x_{29} + 6300x_{30} + \\
&7350x_{31} + 0x_{32} + 0x_{33} + 8500x_{34} + 13600x_{35} + 28900x_{36} + 2500x_{37} + \\
&6000x_{38} + 16100x_{39} + 26400x_{40} + 4000x_{41} + 12500x_{42} + 9600x_{43} + \\
&27400x_{44} + 8500x_{45} + 12000x_{46} + 16600x_{47} + 20400x_{48} + 1020x_{49} + \\
&845x_{50} + 1040x_{51} + 425x_{52} + 1145x_{53} + 720x_{54} + 1165x_{55} + 300x_{56} + \\
&1220x_{57} + 1045x_{58} + 840x_{59} + 350x_{60} + 1445x_{61} + 1020x_{62} + 1190x_{63} + \\
&0x_{64} + 0x_{65} + 240x_{66} + 384x_{67} + 816x_{68} + 240x_{69} + 0x_{70} + 624x_{71} + \\
&576x_{72} + 384x_{73} + 624x_{74} + 0x_{75} + 672x_{76} + 816x_{77} + 576x_{78} + \\
&672x_{79} + 0x_{80} + 153.6x_{81} + 489.6x_{82} + 384x_{83} + 1084.8x_{84} + \\
&393.6x_{85} + 249.6x_{86} + 624x_{87} + 844.8x_{88} + 537.6x_{89} + 873.6x_{90} + \\
&0x_{91} + 940.8x_{92} + 969.6x_{93} + 825.6x_{94} + 672.0x_{95} + 268.8x_{96}
\end{aligned}
\tag{4.51}
$$

Finally, for both of the network topologies MOSEK [111] finds out the optimum result by selecting the consistent variables in a consistent way. For the examples of Figure 4.2 and Figure 4.3 we obtain the same table-site distributions even though the communication costs of the links are uniform for Figure 4.2 and randomly selected for Figure 4.3. The model settles the tables T_0 to S_0, T_1 to S_1, T_2 to

S_2 and T_3 to S_3 for both of the examples.

$$
\begin{aligned}
&5x_1 + 5x_2 + 5x_3 + 5x_4 + 41426x_5 + 41426x_6 + 41426x_7 + 41426x_8 + \\
&0.6x_9 + 0.6x_{10} + 0.6x_{11} + 0.6x_{12} + 0.25x_{13} + 0.25x_{14} + 0.25x_{15} + \\
&0.25x_{16} + 0x_{17} + 2625x_{18} + 2625x_{19} + 5250x_{20} + 2625x_{21} + 0x_{22} + \\
&5250x_{23} + 2625x_{24} + 2625x_{25} + 5250x_{26} + 0x_{27} + 2625x_{28} + \\
&5250x_{29} + 2625x_{30} + 2625x_{31} + 0x_{32} + 0x_{33} + 8500x_{34} + 8500x_{35} + \\
&17000x_{36} + 2500x_{37} + 6000x_{38} + 11000x_{39} + 14500x_{40} + 2500x_{41} + \\
&11000x_{42} + 6000x_{43} + 14500x_{44} + 5000x_{45} + 8500x_{46} + 8500x_{47} + \\
&12000x_{48} + 600x_{49} + 425x_{50} + 425x_{51} + 250x_{52} + 725x_{53} + \\
&300x_{54} + 550x_{55} + 125x_{56} + 725x_{57} + 550x_{58} + 300x_{59} + 125x_{60} + \\
&850x_{61} + 425x_{62} + 425x_{63} + 0x_{64} + 0x_{65} + 240x_{66} + 240x_{67} + \\
&480x_{68} + 240x_{69} + 0x_{70} + 480x_{71} + 240x_{72} + 240x_{73} + 480x_{74} + \\
&0x_{75} + 240x_{76} + 480x_{77} + 240x_{78} + 240x_{79} + 0x_{80} + 96x_{81} + \\
&432x_{82} + 240x_{83} + 576x_{84} + 336x_{85} + 192x_{86} + 480x_{87} + 336x_{88} + \\
&336x_{89} + 672x_{90} + 0x_{91} + 336x_{92} + 576x_{93} + 432x_{94} + 240x_{95} + 96x_{96}
\end{aligned}
\tag{4.52}
$$

4.1.3 Site Capacity

We analyzed the behavior of the model with several examples. Figure 4.7 shows an example where neighbor S_0 can handle all capacities and the other sites do not have enough capacities. Similarly, Figure 4.8 shows an example where neighbor S_2 can handle all capacities and other sites do not have enough capacity. Site capacities are represented with Equation 4.53 to Equation 4.56 for Figure 4.7 and with Equation 4.57 to Equation 4.60 for Figure 4.8. The communication costs are the same as network topologies of Figure 4.2 and Figure 4.3 since the changes do not effect cost of the communication. The update costs of the objective functions are also the same with Figure 4.2 and Figure 4.3 as in Equation 4.61

$$10x_1 + 8x_5 + 6x_9 + 5x_{13} <= 40 \tag{4.53}$$

$$10x_2 + 8x_6 + 6x_{10} + 5x_{14} <= 3 \tag{4.54}$$

$$10x_3 + 8x_7 + 6x_{11} + 5x_{15} <= 3 \tag{4.55}$$

$$10x_4 + 8x_8 + 6x_{12} + 5x_{16} <= 2 \tag{4.56}$$

$$10x_1 + 8x_5 + 6x_9 + 5x_{13} <= 3 \tag{4.57}$$

$$10x_2 + 8x_6 + 6x_{10} + 5x_{14} <= 4 \tag{4.58}$$

$$10x_3 + 8x_7 + 6x_{11} + 5x_{15} <= 40 \tag{4.59}$$

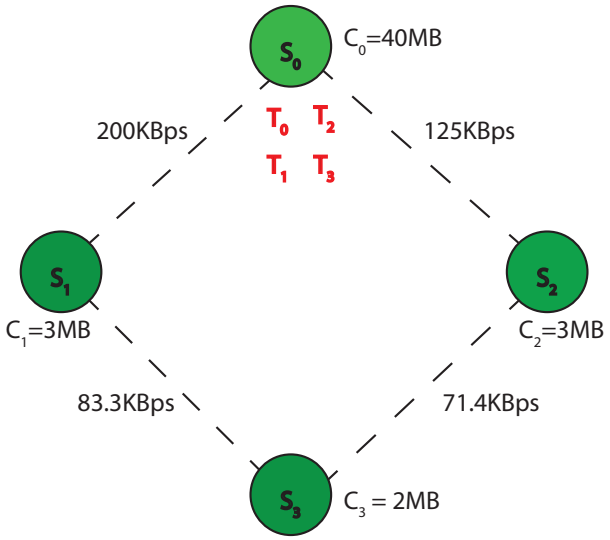

Figure 4.7: Distributed Database System with site capacities of 40MB, 3MB, 3MB and 2MB and capacity of S_0 can handle all base tables.

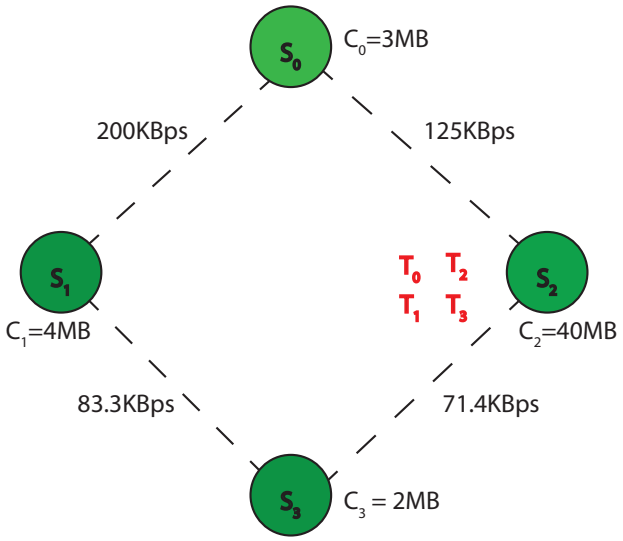

Figure 4.8: Distributed Database System with site capacities of 3MB, 4MB, 40MB and 2MB and S_2 can handle all base tables.

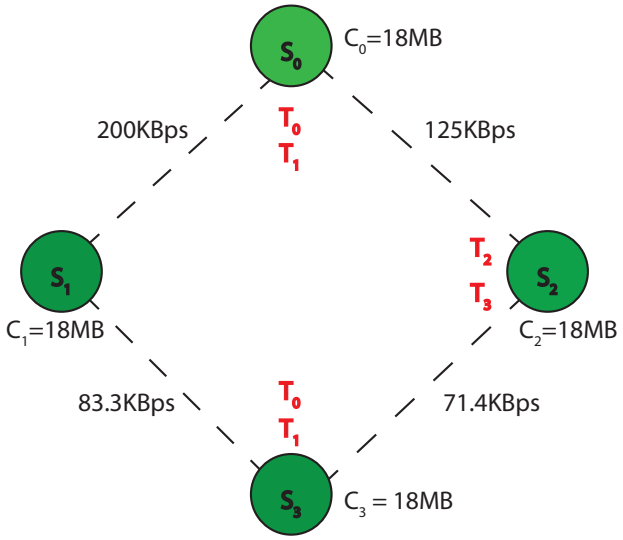

Figure 4.9: Distributed Database System with site capacities of 18MB and update costs are zero.

$$10x_4 + 8x_8 + 6x_{12} + 5x_{16} <= 2 \tag{4.60}$$

$$\begin{aligned} &5x_1 + 5x_2 + 5x_3 + 5x_4 + 1.6x_5 + 1.6x_6 + 1.6x_7 + 1.6x_8 + 0.6x_9 \\ &+0.6x_{10} + 0.6x_{11} + 0.6x_{12} + 0.25x_{13} + 0.25x_{14} + 0.25x_{15} + 0.25x_{16} \end{aligned} \tag{4.61}$$

Final settlements of the tables are T_0, T_1, T_2 and T_3 are in S_0 for Figure 4.7 and T_0, T_1, T_2 and T_3 are in S_2 for Figure 4.8

4.1.4 Replication

Replication is important to reduce the communication costs among the sites when communication costs dominate the update costs. If we choose the update costs as zero for our examples, there exists a possibility for replicas of the tables to be kept in sites. Figure 4.9 shows an example of replication where update costs are zero for all tables. Site capacities are selected as 18MB since all the sites can handle more than one tables with these capacities. The equations for the replication are shown through Equation 4.62 to Equation 4.97. The first 8 equations are for representing the replicas, the following 4 Equations represent the site capacities and the others are the communication costs.

$$x_1 + x_2 + x_3 + x_4 >= 1 \tag{4.62}$$

$$x_5 + x_6 + x_7 + x_8 >= 1 \tag{4.63}$$

$$x_9 + x_{10} + x_{11} + x_{12} >= 1 \tag{4.64}$$

$$x_{13} + x_{14} + x_{15} + x_{16} >= 1 \tag{4.65}$$

$$x_1 + x_2 + x_3 + x_4 <= 2 \tag{4.66}$$

$$x_5 + x_6 + x_7 + x_8 <= 2 \tag{4.67}$$

$$x_9 + x_{10} + x_{11} + x_{12} <= 2 \tag{4.68}$$

$$x_{13} + x_{14} + x_{15} + x_{16} <= 2 \tag{4.69}$$

$$10x_1 + 8x_5 + 6x_9 + 5x_{13} <= 18 \tag{4.70}$$

$$10x_2 + 8x_6 + 6x_{10} + 5x_{14} <= 18 \tag{4.71}$$

$$10x_3 + 8x_7 + 6x_{11} + 5x_{15} <= 18 \tag{4.72}$$

$$10x_4 + 8x_8 + 6x_{12} + 5x_{16} <= 18 \tag{4.73}$$

$$-x_1 + x_{17} + x_{18} + x_{19} + x_{20} = 0 \tag{4.74}$$

$$-x_2 + x_{21} + x_{22} + x_{23} + x_{24} = 0 \tag{4.75}$$

$$-x_3 + x_{25} + x_{26} + x_{27} + x_{28} = 0 \tag{4.76}$$

$$-x_4 + x_{29} + x_{30} + x_{31} + x_{32} = 0 \tag{4.77}$$

$$-x_{17} - x_{21} - x_{25} - x_{29} + x_{33} + x_{34} + x_{35} + x_{36} + x_{49} + x_{50} + x_{51} + x_{52} = 0 \tag{4.78}$$

$$-x_{18} - x_{22} - x_{26} - x_{30} + x_{37} + x_{38} + x_{39} + x_{40} + x_{53} + x_{54} + x_{55} + x_{56} = 0 \tag{4.79}$$

$$-x_{19} - x_{23} - x_{27} - x_{31} + x_{41} + x_{42} + x_{43} + x_{44} + x_{54} + x_{55} + x_{56} + x_{57} = 0 \tag{4.80}$$

$$-x_{20} - x_{24} - x_{28} - x_{32} + x_{45} + x_{46} + x_{47} + x_{48} + x_{55} + x_{56} + x_{57} + x_{58} = 0 \tag{4.81}$$

$$-x_5 + x_{33} + x_{37} + x_{41} + x_{45} + x_{49} + x_{53} + x_{57} + x_{61} = 0 \tag{4.82}$$

$$-x_6 + x_{34} + x_{38} + x_{42} + x_{46} + x_{50} + x_{54} + x_{58} + x_{62} = 0 \tag{4.83}$$

$$-x_7 + x_{35} + x_{39} + x_{43} + x_{47} + x_{50} + x_{54} + x_{58} + x_{62} = 0 \tag{4.84}$$

$$-x_8 + x_{36} + x_{40} + x_{44} + x_{48} + x_{52} + x_{56} + x_{60} + x_{64} = 0 \tag{4.85}$$

$$-x_9 + x_{65} + x_{66} + x_{67} + x_{68} = 0 \tag{4.86}$$

$$-x_{10} + x_{69} + x_{70} + x_{71} + x_{72} = 0 \tag{4.87}$$

$$-x_{11} + x_{73} + x_{74} + x_{75} + x_{76} = 0 \tag{4.88}$$

$$-x_{12} + x_{77} + x_{78} + x_{79} + x_{80} = 0 \tag{4.89}$$

$$-x_{65} - x_{69} - x_{73} - x_{77} + x_{81} + x_{82} + x_{83} + x_{84} = 0 \tag{4.90}$$

$$-x_{66} - x_{70} - x_{74} - x_{78} + x_{85} + x_{86} + x_{87} + x_{88} = 0 \tag{4.91}$$

$$-x_{67} - x_{71} - x_{75} - x_{79} + x_{89} + x_{90} + x_{91} + x_{92} = 0 \tag{4.92}$$

$$-x_{68} - x_{72} - x_{76} - x_{80} + x_{93} + x_{94} + x_{95} + x_{96} = 0 \tag{4.93}$$

$$-x_{13} + x_{81} + x_{85} + x_{89} + x_{93} = 0 \tag{4.94}$$

$$-x_{14} + x_{82} + x_{86} + x_{90} + x_{94} = 0 \tag{4.95}$$

$$-x_{15} + x_{83} + x_{87} + x_{91} + x_{95} = 0 \tag{4.96}$$

$$-x_{16} + x_{84} + x_{88} + x_{92} + x_{96} = 0 \tag{4.97}$$

Finally, tables T_0 and T_1 exist both in sites S_0 and S_3 which means that both of the sites exploit replicas of T_0 and T_1 as a result of the capacity constraints and update costs.

4.1.5 Query Frequencies

Query frequency is an important factor that dominates the total costs for the same query originating from different sites. The tables are to be placed into sites with higher query frequencies. Figure 4.10 and Figure 4.11 show the scenario where query frequencies effect the table allocation.

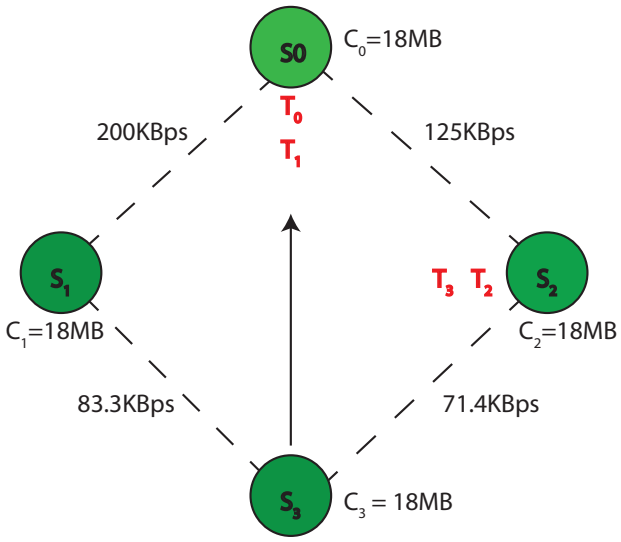

Figure 4.10: Distributed Database System with site capacities of 18MB and S_0 has a frequency of 100 and S_3 has a frequency of 5.

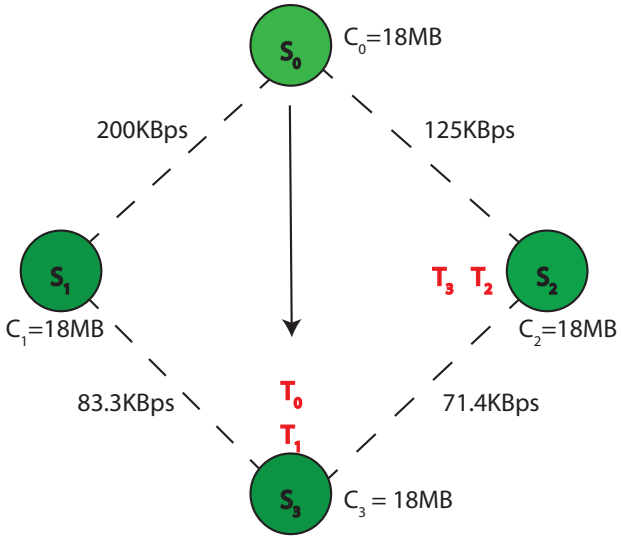

Figure 4.11: Distributed Database System with site capacities of 18MB and S_0 has a frequency of 5 and S_3 has a frequency of 100.

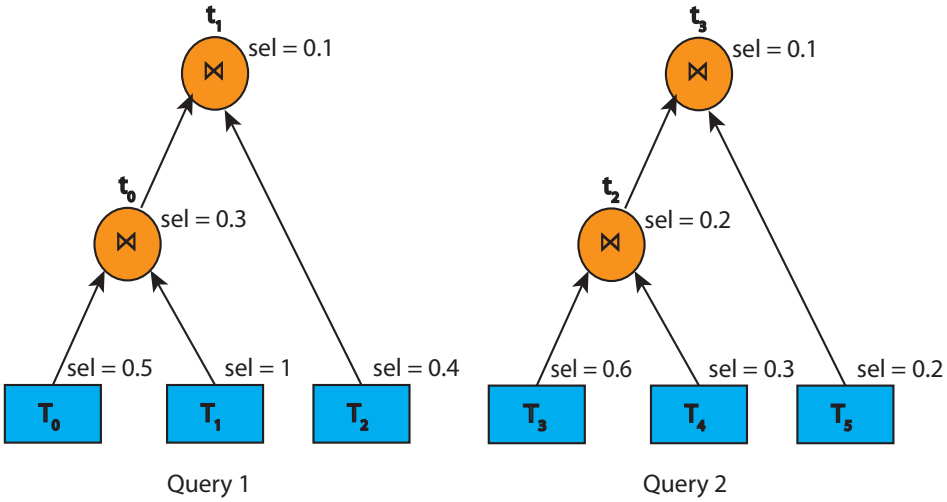

Figure 4.12: Left deep query trees with 2 relations used in our examples.

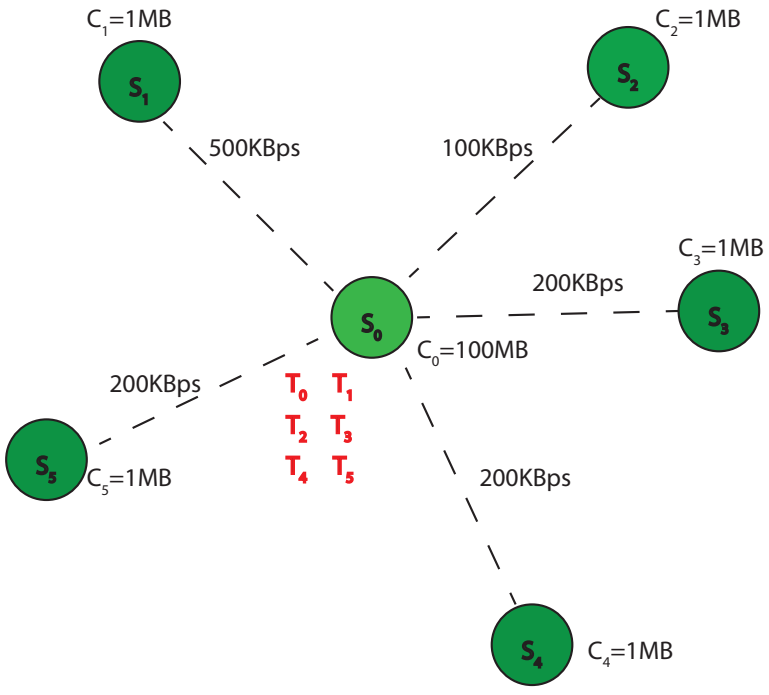

Figure 4.13: Star Network Topology with 6 sites and 6 tables.

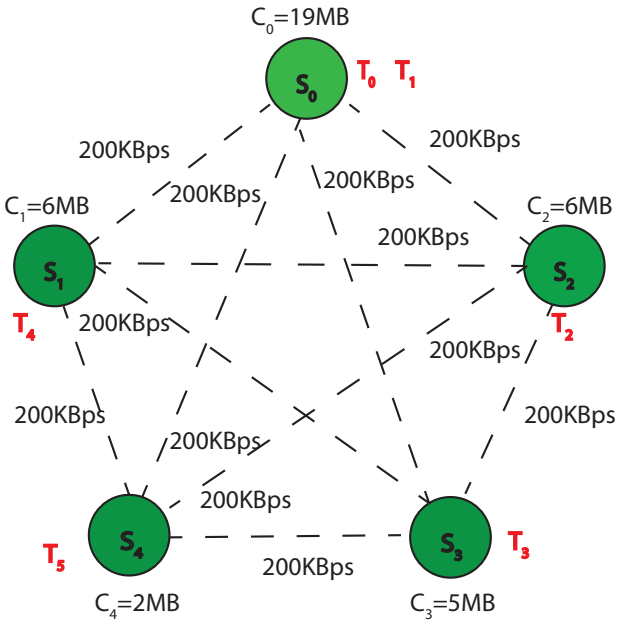

Figure 4.14: Link Network Topology with 6 sites and 6 tables.

.

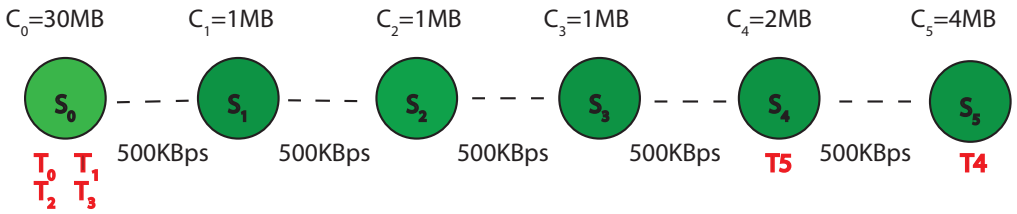

Figure 4.15: Link Network Topology with 6 sites and 6 tables.

.

4.1.6 Modelling Distributed Database Design Problem on Various Network Topologies

We tried three examples with different topologies namely star, link and mesh. For our examples with star and link topologies, we have chosen 6 sites and 2 queries with 2 relations. For the example with the mesh topology we have chosen 5 sites and 2 queries with 2 relations. There are 6 tables to allocate for all the examples used in this section. Tables have sizes of T_0 = 10MB, T_1 = 8MB, T_2 = 6MB, T_3 = 5MB, T_4 = 4MB and T_5 = 2MB respectively. Figure 4.12 shows the queries used in our examples. Both Query 1 and Query 2 are executed only once. In the first example in Figure 4.13, there are 6 sites with capacities C_0 = 100MB, C_1 = 1MB, C_2 = 1MB, C_3 = 1MB, C_4 = 1MB and C_5 = 1MB. The communication costs for Figure 4.13 are 500KBps for S_0 to S_1, 100KBps for S_0 to S_2, 200KBps for S_0 to S_3, 200KBps for S_0 to S_4, 200KBps for S_0 to S_5. The tables are gathered in S_0 after our tool finishes table to site assignments. In the second example in Figure 4.14, there are 5 sites with capacities C_0 = 19MB, C_1 = 6MB, C_2 = 6MB, C_3 = 5MB and C_4 = 2MB. The communication costs for Figure 4.14 are uniform and 200KB for all the links. The tables T_0 and T_1 are in S_0, T_4 is in S_1, T_2 is in S_2, T_3 is in S_3, T_5 is in S_4 after our tool finishes table to site assignments. In the last example in Figure 4.15, there are 6 sites with capacities C_0 = 30MB, C_1 = 1MB, C_2 = 1MB, C_3 = 1MB, C_4 = 2MB and C_5 = 4MB. The communication costs for Figure 4.14 are uniform and 500KB for all the links. The tables T_0, T_1, T_2 and T_3 are in S_0, T_5 is in S_4, T_4 is in S_5 after our tool finishes table to site assignments.

4.2 Distributed Database Design as a Quadratic Assignment Problem

In [1] Data Allocation Problem is formulated as a QAP formulated using sum of costs associated with ”direct” and ”indirect” fragment dependencies. The dependency between a query q and a fragment f is called *direct* if there is a data transmission from the site containing f for each execution of q. If there is some data to be transferred to a site different from the originating site of transaction, the dependency is considered as *indirect*. The total cost of data allocation *Cst* is the sum of two costs *Cst*1 and *Cst*2 (Equation 4.98) [1].

$$Cst(\Phi) = Cst1(\Phi) + Cst2(\Phi) \tag{4.98}$$

In Equation 4.98, Φ represents the m element vector where Φ_j specifies the site to which f_j is allocated. *Cst*1 is represented by the amount of site-fragment dependencies. It is expressed by the multiplication of two matrices *STFR* and *UC*, where *STFR* stores the site fragment dependencies and *UC* stores the unit communication cost among the sites. The cost of storing a fragment f_j in site s_i is represented by partial cost matrix $PCST1_{n\times m}$. The unit partial cost matrix is formulated in Equation 4.102 [1].

$$pcst1_{ij} = \sum_{q=1}^{n} uc_{iq} \times stfr_{qj} \tag{4.99}$$

*Cst*1 can be represented as in Equation 4.100 after having calculated the unit partial cost $pcst1_{ij}$ for each i and j.

$$Cst1(\Phi) = \sum_{j=1}^{m} pcst1_{\Phi_j j} \tag{4.100}$$

The inter-fragment dependency matrix *FRDEP* is defined as the multiplication of the matrices $QFR_{l\times m\times m}$ and $Q_{l\times m\times m}$. The matrix *QFR* denotes the execution frequencies of the transactions. The *FRDEP* matrix representing the inter-fragment dependency is the multiplication of the matrix *QFR* with the matrix *Q*. *Q* represents the indirect transaction fragment dependency. Equation 4.101 formulates the indirect transaction-fragment dependency *Cst*2. *Cst*2 is a form of the *QAP* as seen in the equation.

$$Cst2(\Phi) = \sum_{j_1=1}^{m}\sum_{j_2=1}^{m} frdep_{j_1 j_2} \times uc_{\Phi_{j_1}\Phi_{j_2}} \tag{4.101}$$

$$pcst1_{ij} = \sum_{q=1}^{n} uc_{iq} \times stfr_{qj} \tag{4.102}$$

Algorithm 5 DDB Design with QAP

```
minCost = infinity;
perm = NULL;
validpermFound = false;
sum = 0;
TotalComCostMatrix = 0;
for each query Q_i do
    TotalComCostMatrix += freq_i * ComCostMatrix(Q_i);
end for
for each permutation perm_j do
    sum = TotalComCostMatrix * AllocationMatrix + UpdateCost;
    for each query Q_i do
        sum += DynamicCost(Q_i, Q_s, perm)
    end for
    if sum < minCost and isValid(capacity(S_0, S_1, S_2, S_3))) then
        minCost = sum;
        bestPerm = perm;
        validpermFound = true;
    end if
end for
if validpermFound == true then
    print minCost;
    print perm;
end if
```

4.2.1 DDB Design Example with QAP

Figure 4.2 can be modeled with QAP as follows with Algorithm 5:

T_0 = 10 * 0.5 =5MB, T_1 = 8 * 1 = 8MB, T_2 = 6 * 0.4 = 2.4MB, T_3 = 5 * 0.8 = 4MB. Table 4.14 shows the communication costs among sites. Table 4.15 and Table 4.16 show the facility matrices for respective queries. Dynamic costs are added to the cost function after the permutation is determined for *ComCostMatrix* $\times$ *AllocationMatrix*. Equation 103 shows the fitness value for each permutation value. Update Table 4.17 shows the update cost values and cost calculations for all permutations are in Table 4.18. Details of the cost calculations can be found in Appendix E.

$$QResultTransferCost = DynamicCost(Q_i, Q_s, perm) \tag{4.103}$$

Table4.14: Flow Matrix

	F_0	F_1	F_2	F_3
F_0	0	5	8	17
F_1	5	0	13	12
F_2	8	13	0	14
F_3	17	12	14	0

Table4.15: Query Result Transfer Matrix(Query 1)

	T0	T1	T2	T3
T0	0	5	0	0
T1	0	0	0	0
T2	0	0	0	0
T3	0	0	0	0

Table4.16: Query Result Transfer Matrix(Query 2)

	T0	T1	T2	T3
T0	0	0	0	0
T1	0	0	0	0
T2	0	0	0	2.4
T3	0	0	0	0

4.3 Island Parallel Genetic Algorithm for *QAP*

Parallel computation performs many calculations simultaneously using the divide and conquer paradigm. The larger problem instances can often be divided into smaller ones, which are then solved on multiple

Table4.17: Update Costs in the DDB Design Example (Appendix D-Table D.1).

Table \ Site	S_0	S_1	S_2	S_3
T_0	5sec	5sec	5sec	5sec
T_1	1.6sec	1.6sec	1.6sec	1.6sec
T_2	0.6sec	0.6sec	0.6sec	0.6sec
T_3	0.25sec	0.25sec	0.25sec	0.25sec

Table4.18: Cost Calculations for All Permutations

0123	24147	1212,05	3281,48	28640,53	**SOLUTION FOUND (0,1,2,3)**
0132	26067	1308,05	3665,48	31040,53	
0213	22948,2	1152,85	3029	27130,05	Site Capacity Eliminates
0231	22948,2	1152,85	3029	27130,05	Site Capacity Eliminates
0312	26071,8	1303,25	3666,44	31041,49	
0321	24151,8	1207,25	3282,44	28641,49	
1032	26065	1310,05	3660,68	31035,73	
1023	24145	1214,05	3276,68	28635,73	
1230	29448,2	1477,85	3653	34579,05	
1203	25448,2	1277,85	3269	29995,05	
1320	30651,8	1532,25	3906,44	36090,49	
1302	28571,8	1428,25	3906,44	33906,49	
2013	22945	1154,05	3036,68	27135,73	Site Capacity Eliminates
2031	22945	1154,05	3036,68	27135,73	Site Capacity Eliminates
2103	25447	1277,05	3281,48	30005,53	
2130	29447	1477,05	3665,48	34589,53	
2301	25451,8	1272,25	3282,44	30006,49	
2310	29451,8	1472,25	3666,44	34590,49	
3021	24145	1214,05	3276,68	28635,73	Site Capacity Eliminates
3012	26065	1310,05	3660,68	31035,73	
3120	30647	1537,05	3905,48	36089,53	
3102	28567	1433,05	3905,48	33905,53	
3210	29448,2	1477,85	3653	34579,05	
3201	25448,2	1277,85	3269	29995,05	

*CPU*s and/or computers in parallel [5]. *GA*s are very suitable for parallelization in terms of granularity since the populations are divisible and the fitness values of the individuals can be computed in parallel.

The *QAP-IPGA* is an extension of the *GA* [147]. It locates different sub-populations to evolve in diverse directions concurrently while speeding up the search process. The *QAP-IPGA* requires the distribution of sub-populations to many sites, calculation of fitness values in parallel, and the migration of best individuals to other nodes. The *QAP-IPGA* distributes subsets of the population among parallel processing nodes. There exists an occasional exchange of the best individuals in order to increase the average quality of the sub-populations in all processing nodes.

Each slave node has a non-redundant initial population. The initial subsets of population are created randomly with respect to assigned boundaries of the defined search space. The possible set of all permutations is divided into subsets and populations are selected respectively from these subsets for each slave node. Non-redundancy of the populations is guaranteed by dividing the problem into subsets which minimize the interconnection among these subsets by Dijkstra's next permutation algorithm [52].

Each slave generates as many new individuals as half of its current population size. The best available solution for each slave is sent to the master at the end of every generation. These best solutions are collected in the master node and are re-broadcasted to all other slaves. The newly arriving best individuals are added to sub-populations in all slaves. Since they do not replace other individuals the population size in each node increases, until the number of new individuals exceeds a fixed ratio of the initial slave populations (to prevent overcrowding).

Parallel computation performs many calculations simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved on multiple *CPU*s and/or computers in parallel [5]. The need for expensive fitness calculations in genetic algorithms makes them very compatible with this principle of the parallel computers. Divisibility of the population and the computation of the fitness values of the individuals in parallel are the main motivations for the parallelization of *GA*s.

*GA*s have been frequently used to solve search and optimization problems since they were introduced by [79]. *GA*s gained this prominence by the robustness and simplicity they offer. They use a computational model that simulates the natural processes of selection and evolution. *GA*s can perform efficient search operations in problem areas where it is not easy to understand the nature of the problem. Each potential solution existing in the search space is considered as an individual (phenotype) and represented by strings called chromosomes. Genes are the atomic parts of chromosomes and they codify a specific characteristic of a chromosome. There are several ways to encode individuals for a variety of applications. A random population is generated in the first step of the algorithm and by applying selection, crossover, and mutation operations cyclically, *GA*s create new generations [74]. The individual having the best fitness value in the population is the solution of the problem. *GA*s can terminate depending on the time spent running the algorithm, the number of generations produced, or having no (or very little) improvement in the average fitness value of the population. *GA*s select the individuals to crossover, or mutate by using different selection mechanisms. *Tournament, roulette wheel*, and *truncation* are the most frequently used methods. In our study, we evaluated all of these selection methods. In tournament, the best one among the selected individuals is used for mating. In roulette wheel, a random number is generated to choose an individual for mating where probability of selection of an individual depends on its fitness value, giving reproductive advantage to fitter individuals. In truncation, individuals with the lowest fitness values are eliminated from the population with a ratio. Selected individuals are submitted to *crossover* and *mutation* operators.

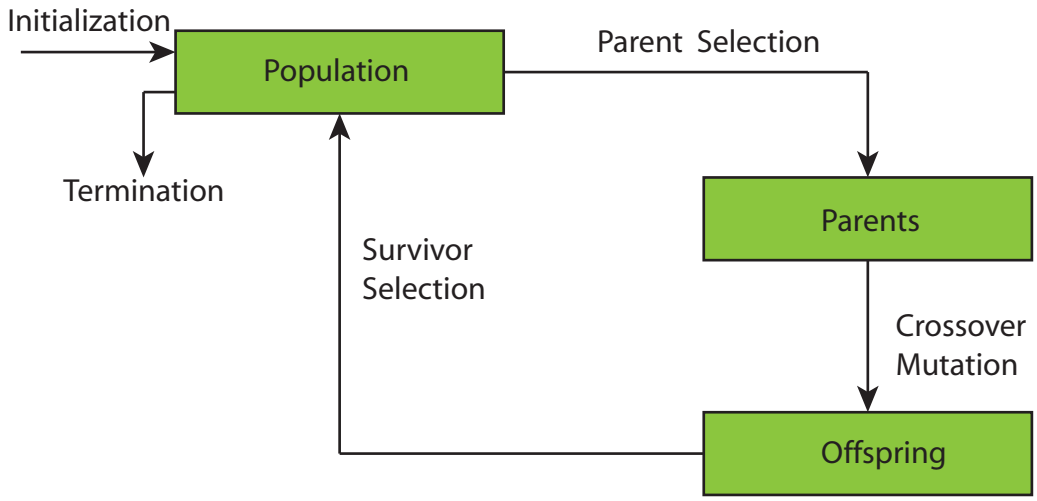

Figure 4.16: Data flow-diagram of a genetic algorithm.

*GA*s generate new individuals by guaranteeing that the best individuals will not be discarded in future generations. In this algorithm, the fittest individuals are copied into the next generations. The crossover operator generates new individuals by recombining the characteristics of parents. For each pair of individuals, the parent chromosomes are split into parts and genes are exchanged to generate new chromosomes. Individuals not subjected to any operation are copied into the next generation. If a mutation happens, it modifies an individual's genetic code [132, 16].

After executing genetic operators a new population is generated, and generations continue to be produced until the termination condition is reached. Figure 4.16 explains the general scheme of *GA*s. *GA*s work by using the objective function and do not consider any other auxiliary knowledge about the search space. Genetic Algorithm, Particle Swarm and Ant Colony based algorithms have been successfully used for optimizing distributed database query optimization problem. By solving the *QAP* efficiently the distributed database design problem can be formulated as a *QAP* and solved efficiently.

QAP-IPGA is an extension of *GA*s. Its advantage is to facilitate different subpopulations to evolve in diverse directions concurrently, dramatically speeding up the search process, thus potentially producing higher quality solutions by producing a larger number of individuals and generations. Parallelization of *GA*s will require decisions to be made on distribution of populations to many sites, calculation of fitness values in parallel, and the migration of best individuals to other nodes. There are three main types of *PGA*s according to [25]. The *QAP-IPGA* described in this work contains a number of subpopulations, which can occasionally exchange individuals and this exchange of individuals helps to improve the average quality of the subpopulations. Each subpopulation is assigned to a slave processor on which genetic operations are performed independently of the other processors [99].

The proposed *QAP-IPGA* generates non-redundant initial populations at each slave node. The individuals of each slave node are generated randomly between the assigned boundaries of the defined search space. The non-redundancy of the populations is guaranteed by dividing the solution space into subdivisions [52].

In *QAP-IPGA*, each slave generates half as many new individuals as the current population size and sends the best available solution to the master at every generation. Master node collects and sends the best received individuals to all other slaves until the number of these new individuals exceeds the defined ratio (10% in our experiments) of the initial populations of the slaves. Slaves add these

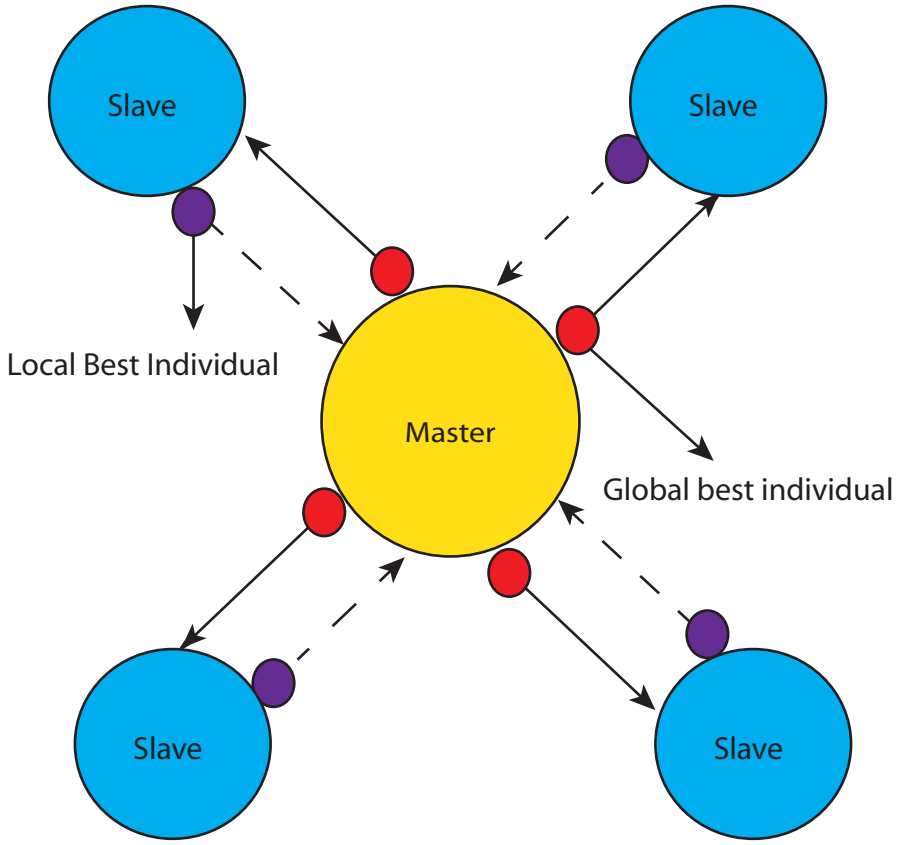

Figure 4.17: Fully informed parallel genetic algorithm migration topology.

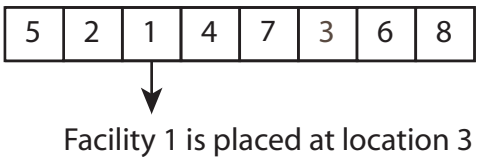

Figure 4.18: Representation scheme of the chromosome structure.

newly arriving best individuals of other nodes to their populations. They do not replace any of the individuals in the initial populations causing an increase in the population size. In order to prevent the slave populations from increasing too much, we defined a ratio of the initial population size so that exchange of best individuals will stop when the initial population of slaves is increased by that amount.

Communication topology of this type is known as fully informed, as shown in Figure 4.17. Details of communication topologies of *IPGA*s can be found in [103]. The detailed flow of the *QAP-IPGA* is explained in Algorithm 6. *QAP-IPGA* terminates its execution when slaves complete the number of generations defined by the master node. Generation based termination conditions are very appropriate for the *QAP-IPGA*, so we preferred parallelizing the entire slave nodes using this method.

We have developed 9 different variants of *QAP-IPGA*. Table 4.19 shows the corresponding *QAP-IPGA*s with three different crossover and three different selection mechanisms.
The structure of the *QAP-IPGA* chromosome is as shown in Figure 4.18.

Algorithm 6 *QAP-IPGA* Algorithm

```
if Node is the Master then
    Send The Generations' Termination Limit
    while Number of Generations Elapsed ≤ Termination Limit do
        Receive the Best Results from Slaves at Each Generation
        Update the Best Configuration
        if ratio < limit then
            Send best individuals of this generation to all slaves
        end if
        Show the Best Configuration
    end while
end if
if Node is a Slave then
    Receive The Generations' Termination Limit from the Master
    P ← Generate non-redundant Population at each site
    while Number of Generations Elapsed ≤ Termination Limit do
        Receive the best individuals coming from the Master
        (p1, p2) ← SelectParentsPair (P)
        s ← Crossover (p1, p2)
        P ← PopulationUpdate (P,s)
        Send the Best Configuration to Master Node
    end while
end if
```

Table4.19: Variants of *QAP-IPGA* with different crossover and selection mechanisms.

Algorithm Name	Crossover	Selection Method
$QAP-IPGA_1$	$Order1$	$Truncation$
$QAP-IPGA_2$	$Order1$	$Tournament$
$QAP-IPGA_3$	$Order1$	$RouletteWheel$
$QAP-IPGA_4$	PMX	$Truncation$
$QAP-IPGA_5$	PMX	$Tournament$
$QAP-IPGA_6$	PMX	$RouletteWheel$
$QAP-IPGA_7$	$Cycle$	$Truncation$
$QAP-IPGA_8$	$Cycle$	$Tournament$
$QAP-IPGA_9$	$Cycle$	$RouletteWheel$

Algorithm 7 Parallel Exhaustive Algorithm for *QAP*

```
if Node is the Master then
    Divide the Solution Space into subsolution Spaces
    Send the Solution Space Limits to Slaves
    Send Matrices to Slave Nodes
    Receive the Results from Slaves
end if
if Node is a Slave then
    Receive the Matrices and the Solution Space Limits
    while Number of Iterations ≤ Termination Limit do
        Compute the Next Solution
        if new solution is better than the existing then
            Update the Best Configuration
        end if
    end while
end if
```

Algorithm 8 Dijkstra's Get Next Permutation Algorithm

```
Value: The Permutation Vector
N: The Length of the Value
Set i = N - 1
while Value[i-1] >= Value[i] do
    Set i = i-1
end while
Set j = N
Swap values at positions (i-1) and (j-1)
Set i = i+1
Set j = N
while i < j do
    Swap values at positions (i-1) and (j-1)
    Set i = i+1
    Set j = j-1
end while
```

Algorithm 9 Load Balancer Algorithm

```
Loadbalancer (initialPerm,n,len)
Currperm = initialPerm
Length =currPerm.length
if length = 1 then
    return initialPerm
end if
residual = n
number of suffix_permutations = factorial(len - 1)
digit_index = 0
if number of suffix_permutations < residual then
    digit_index = residual / number of suffix_permutations
    if residual mod numSuffixPermutations = 0 then
        digit_index -= 1
        residual -= (digit_index × number of suffix_permutations)
    end if
end if
indexDigit = currPerm[digit_index]
length= 0
for i = 0 to len do
    if currPerm[i] != indexDigit then
        temp = temp + currPerm[i]
        length = length + 1
    end if
end for
currPermutation = temp
return indexDigit + Load Balancer (currPerm, residual, length)
```

4.4 Parallel Exhaustive Algorithm (*PEA*)

We developed the *PEA* to find the optimum solutions for the *QAP* instances as large as possible we can, so that we could compare these solutions with those found by the *QAP-IPGA*s and determine their quality with respect to the optimal solutions. The *PEA* is a Master and Slave distributed algorithm model, details of which are shown in Algorithm 7. The master node assigns search space intervals to the slave processors and each slave finds the best solution and sends it back to the master. The master selects the best among those received solutions, determining the global optimum.

In parallel algorithms load of each processor should be equally distributed in order to achieve the best speedup possible. Therefore, we needed to use a load balancer to distribute the search space. Each processor learns its range of exploration before it starts executing. After the limits for the search space are decided, nodes incrementally find the limits of their permutations using Dijkstra's lexicographic permutation generator algorithm as shown in Algorithm 8 [52]. We used a balancer algorithm as in Algorithm 9 which recursively fills the permutation vector with all (n-1)! permutations for each length value n.

4.5 Parallel Hybrid Genetic Tabu Search Algorithm (*PHGETS*)

PHGETS combines the *IPGA* paradigm [147] with Taillard's well known *RTS* algorithm [141]. It contains three steps; seed generation, *TS* diversification decision, and *RTS*. The *TS* diversification phase of *PHGETS* uses an enhanced version of the diversification operator of the Cooperative Parallel Tabu Search (*CPTS*) [83]. Even though it possesses the multi-start characteristics of the *CPTS*, the *TS* diversification phase performs the diversification procedure for all step sizes smaller than the problem size. The seeds are determined according to the step size. After having decided on a high quality seed, it performs a stepwise procedure to find the best diversification for the solution. Finally, this intermediate diversification result is fed to the *RTS* algorithm. The experimental results show that the final discovered solutions are very close to the optimal (or best known) results. Algorithm 10 shows the steps of *PHGETS*.

4.5.1 Generating the Seed Permutation

The *PHGETS* algorithm generates a high quality seed by using the *QAP-IPGA_4* algorithm. *QAP-IPGA_4* selects the population via truncation and uses the *PMX* algorithm for crossover. *QAP-IPGA_4* is shown to have the best performance among the 9 proposed *IPGA* algorithms [147]. Thus, *QAP-IPGA_4* is used as the seed generator of the *PHGETS* algorithm. The slave nodes of *QAP-IPGA_4* use different sub-populations to search in diverse directions concurrently while speeding up the search process. The slave nodes exchange individuals and this exchange of individuals improves the average quality of the sub-populations.

4.5.2 Tabu Search Diversification

Glover's diversification operator [73] is used in *CPTS*. Algorithm 11 shows the basic diversification procedure used in *CPTS* [83]. The diversification operator uses a fixed step parameter to create a new shuffle of the permutation vector as the seed for a new search. *CPTS* stores each reference set item and

Algorithm 10 PHGETS Algorithm Master Node.

```
Vector best_results;

BROADCAST_TO_ALL_SLAVES(IterationLimit);

IPGAIteration = IterationLimit/10;
// IPGA initialization for slave pools.
for (i = 0; i < IPGAIteration; i++) {
    RECEIVE_FROM_ALL_SLAVES(perm[slave]);
    best_results = best_results UNION perm[slave];
    BROADCAST_TO_ALL_SLAVES(best_results);
    best_results = {};
}

RECEIVE_FROM_ALL_SLAVES(SEED[slave]);
Best_Permutation= MinCostPermutation(SEED[0..N-1]);
PRINT "BestPermutation=", BestPermutation,
        " cost=", QAPCost(BestPermutation);
```

increments the step parameter (initial value 2) each time the diversification procedure is called. An example is given below for step size 3:

Let *SEED* = { 3, 5, 7, 8, 9, 6, 2, 12 }

The algorithm sets the start variable to step size, 3.

The result of each iteration of the outer for-loop is shown below:

It starts copying values from position 3 and the initial step size is 3.

PERM = {7, 6, _ ,_ ,_ ,_ ,_ ,_ }

The next part of the permutation is constructed with the start value 2 (3 is decremented by 1) and step size is again 3.

PERM = {7, 6, 5, 9, 12, _, _ }

Finally, for start = 1 and step size 3 we obtain:

PERM = {7, 6, 5, 9, 12, 3, 8, 2}

PHGETS tries every step of the diversification operator on the given seed of the *IPGA* as seen in Algorithm 12. It evaluates the convergence speed of the seed in a stepwise manner by looking at different trials of the diversification operator. The trials continue until no improvement can be achieved.

The diversification operator of *PHGETS* finds a permutation with the highest improvement rate. The tabu diversification phase of the algorithm consists of the enhanced diversification operator and it prepares the best convergence rate permutation vector for *RTS*.

Algorithm 10 PHGETS Algorithm Slave Node.

```
Vector best_results;

RECEIVE_FROM_MASTER(IterationLimit);

P = GENERATE_POOLS(N, i, PoolSize, NSlaves);

Population_Limit = sizeof(P)×1.5;
generation_count = 0;
IPGAIteration = IterationLimit/10;

while (generation_count < IterationLimit){
    if (generation_count < IPGAIteration) {
        SEND_TO_MASTER(best_individual);
        RECEIVE_FROM_MASTER(best_results);
        P = P UNION best_results;
    }

    while (sizeof(P) < PopulationLimit){
        (p1, p2) ← SelectParentsPair(P);
        child = CrossoverAndMutation(p1, p2);
        P = P UNION child;
    }

    truncate(P, PopulationLimit/1.5);
    generation_count++;
}

SEED = BestIndividual ∈ P;
N = dimension_of_permutation_vector;
PHGETS_DIVERSIFY(SEED, N);
RTS (FLOW, DIST, MaxIter, SEED,
      MinSize, MaxSize, Aspiration);

SEND_TO_MASTER(SEED);
```

Algorithm 11 CPTS Diversification Operator [83].

```
CPTS_DIVERSIFY(SEED, N, STEP) {
    PERM = {0}
    for (position = 0, start = STEP; start >= 0; start--)
        for (i = start; i < N; i+=STEP)
            PERM[position++] = SEED[i]
    SEED = PERM
}
```

Algorithm 12 PHGETS Diversification Operator.

```
PHGETS_DIVERSIFY(SEED, N)
{
    PERM = {0}
    while (there is improvement)
      for (STEP = N; STEP >= 1; STEP--)
        for (position = 0,start = STEP;start >= 0;start--)
            for (i = start; i<N; i+= STEP){
              PERM[position++] = SEED[i];
              if (QAPCost(SEED) > QAPCost(PERM))
                  SEED = PERM
            }
}
```

4.5.3 Final Phase with Robust Tabu Search

The *TS* proposed by Taillard [141] starts with a steepest descent algorithm and continues by making upward and downward moves towards the solution. Earlier moves affect the future moves and a tabu list prevents moving in a reverse direction for a certain number of moves. This way the process is forced to search through unexplored areas. When the search process gets stuck in a local minimum, it tries to find a better solution by escaping from it, even if it means moving to a worse solution (i.e. an upward move). Moves that force the search back into a local minimum are not allowed by the tabu list of *TS*.

The *TS* has an adaptive memory to explore the search space and uses this memory to forbid returning to recently visited solutions. Variants of *TS* use different strategies such as diversification and intensification to focus on more promising search regions. *PHGETS* uses diversification in its *TS* diversification phase to increase performance. After designing appropriate intensification and diversification strategies to decide on a search direction, it explores the search space using *TS* which is the most time consuming phase of the *PHGETS* algorithm.

Changing the tabu list size is the basic strategy in *TS* variants. A smaller size tabu list enables the search in and around local minima, whereas bigger lists will help the search to escape from the local minima. Several aspiration criteria can be used with tabu restrictions to make a decision about a particular move. There are many different *TS* algorithms exploring intensification and diversification strategies with a variety of parameter settings. The long-term memory generally keeps track of the frequency of the components that occur in high-quality solutions. For long-term intensifications, components with a high frequency are selected for future searches.

Exchanging the previous locations of two facilities with each other is a very appropriate way to generate new permutations for the neighborhood relations of the *QAP*. The computational simplicity of the swap makes it very appropriate when compared with the other larger exchange methods. In Taillard's *RTS*, these two-exchange moves save a great amount of execution time. The fast evaluation of the moves is an important issue that contributes to the efficiency of the search.

RTS seeks a neighbor solution with the best evaluation. This evaluation forces the choice of a move that improves the objective function most. In order to avoid returning to the local optimum just visited, the reverse move is forbidden. There is a tabu list to define forbidden moves. There may be some moves exempt by the tabu criterions. These moves may lead to better solutions that the best one found

so far. Aspiration criterion is introduced to allow tabu moves whenever a move may have a chance to come up with a better result. Moreover, *RTS* uses tabu list size and number of failures as parameters. Number of failures define the range in which no improving solution is found in terms of number of iterations. *PHGETS* uses *RTS* in its decision phase. Rather than using the diversification operator after a number of failures, it searches for the best search direction for its RTS phase.

4.6 Application Areas of the Parallel Algorithms in Production Research

A Supply Chain (*SC*) is a system of members and processes which illustrates the activities in moving a product from supplier to a customer. *SC* transforms natural resources and raw materials into a complete product. It covers all the steps to deliver this product end-to-end from supplier to customer. In *SC*s where residual value is recycled, products may enter at any level [113]. Recent companies all work in one *SC* successfully while satisfying high quality service and products. Respective levels of *SC* communicate and improve the quality of products and services. The main focus of *SC* is to increase profitability while providing high quality service to customer. The *SC* operations start with a customer request and end with a successful purchase [37]. Supply Chain Management (*SCM*) is a group of activities to coordinate retailers, wholesalers, distributors and factories in order to produce and consume at equilibrium while minimizing the total costs and increasing service quality [96].

In an *SC* each layer can transfer data in an up and down direction from the prevailing layer. The *SC* involves all the members and processes participating in the service to customer demands. Members can be categorized as retailers, wholesalers, distributors and factories. The processes are like stock, transport, store, order and delivery. The *SC*s are complex systems to be managed taking into account the members and processes, managerial decision coordination, indistinct material and knowledge flow. The disorder in the *SC* processes may arise with numerous side effects. The bullwhip effect is one of these side effects. It is the primary cause of production-demand mismatch. Companies waste plenty amount of money every year because of it. Fluctuations in layers of the *SC* define the bullwhip phenomena. The information deterioration in each layer of the *SC* may cause problems like lack of inventory, longer service time, raw material shortage, poor quality and missing deadlines [94, 34]. Information deterioration increases while moving up the *SC*. [112] created a game named Beer Game to show how an *SC* operates and how the bullwhip effect occurs. Customer demands usually change in a way difficult to foresee. Companies should predict the unstable customer demand to urgently manage resource positions. Generally predictions about the customer demands deflects even though they are statistically defined. Due to the deviations in the supply and demand balance, some of the companies buffer safety stocks. The need for safety stock increases more at higher levels of the *SC* when rising demands increase purchase orders. Each level of the *SC* observes more variation in a bottom-up traversal. There is also a risk of falling demand which causes excessive inventory at each level in case safety stock is used. Parallel algorithms used for *QAP* can be applied to reducing bullwhip effect in *SC*.

A Supply Chain (*SC*) is a system of members and processes which illustrates the activities in moving a product from supplier to a customer. *SC* transforms natural resources and raw materials into a complete product. It covers all the steps to deliver this product end-to-end from supplier to customer. In *SC*s where residual value is recycled, products may enter at any level [113]. Recent companies all work in one *SC* successfully while satisfying high quality service and products. Respective levels of *SC* communicate and improve the quality of products and services. The main focus of *SC* is to increase profitability while providing high quality service to customer. The *SC* operations start with a customer request and end with a successful purchase [37]. Supply Chain Management (*SCM*) is a

group of activities to coordinate retailers, wholesalers, distributors and factories in order to produce and consume at equilibrium while minimizing the total costs and increasing service quality [96].

In an *SC* each layer can transfer data in an up and down direction from the prevailing layer. The *SC* involves all the members and processes participating in the service to customer demands. Members can be categorized as retailers, wholesalers, distributors and factories. The processes are like stock, transport, store, order and delivery. The *SC*s are complex systems to be managed taking into account the members and processes, managerial decision coordination, indistinct material and knowledge flow. The disorder in the *SC* processes may arise with numerous side effects. The bullwhip effect is one of these side effects. It is the primary cause of production-demand mismatch. Companies waste plenty amount of money every year because of it. Fluctuations in layers of the *SC* define the bullwhip phenomena. The information deterioration in each layer of the *SC* may cause problems like lack of inventory, longer service time, raw material shortage, poor quality and missing deadlines [94, 34]. Information deterioration increases while moving up the *SC*. [112] created a game named Beer Game to show how an *SC* operates and how the bullwhip effect occurs. Customer demands usually change in a way difficult to foresee. Companies should predict the unstable customer demand to urgently manage resource positions. Generally predictions about the customer demands deflects even though they are statistically defined. Due to the deviations in the supply and demand balance, some of the companies buffer safety stocks. The need for safety stock increases more at higher levels of the *SC* when rising demands increase purchase orders. Each level of the *SC* observes more variation in a bottom-up traversal. There is also a risk of falling demand which causes excessive inventory at each level in case safety stock is used. Parallel algorithms used for *QAP* can be applied to reducing bullwhip effect in *SC*. There are many studies to reduce the bullwhip effect starting with [108]. He tried to determine the decrease in profitableness due to the bullwhip effect experimentally. The crucial influences of the bullwhip effect differs with respect to business ramifications. Even though business differs, decreasing the bullwhip effect increases profits and productivity considerably. [34] investigated the bullwhip effect on a two-level *SC*. The *SC* consisted of a single customer and a single manufacturer. It was showed that primary causes of bullwhip effect such as demand forecasting and procurement lead times can be reduced even though there is not a way to bypass the bullwhip effect. The results of this two level *SC* were further improved for multilevel *SC*s. Analyzing an *SC* with several levels, [24] claimed that the bullwhip effect may be reduced by computational methods. They emphasized the need for information sharing among the participants of the *SC* while achieving a commitment on future estimations. [117] states that the bullwhip effect reduces if a vendor managed inventory supply chain (*VMI-SC*) is used instead of a conventional *SC* serially linked. With the use of *VMI-SC* rationing game causing the bullwhip effect can be completely eliminated. [140] and [118] concentrated on stability and bullwhip effect on *SC* infrastructures. They claimed that estimations over a basic *SC* can be misleading because risks are present in real life.

There are many techniques proposed to reduce the Bullwhip Effect. Traditional approaches estimate *SC* features theoretically by calculus and stochastic analysis [94]. Mathematical programming is employed for *SC* problems [68] like sharing information. Discrete Event Simulation allow validating the *SC* behaviour [17]. Computational Intelligence (*CI*) algorithms like Genetic Algorithms [115], Fuzzy Inventory Controller [153] and Distributed Intelligence [50] are generally used to reduce the bullwhip effect. Genetic Algorithms (*GA*s) provide many advantages [71]. They do not use any mathematical constraints and they can be used for discrete, continuous and hybrid search space. Moreover, genetic operators make them effective for local and global search. Several researchers implemented *GA*s to reduce the bullwhip effect. [53] used *GA* for inventory control, [93] employs a *GA* for planning and scheduling, [29] modelled a two-stage *SC* consisting of a retailer and manufacturer with *GA*s. [87] showed that MIT Beer Game may be modelled with an *SC*. [115] tried to find efficient ordering strat-

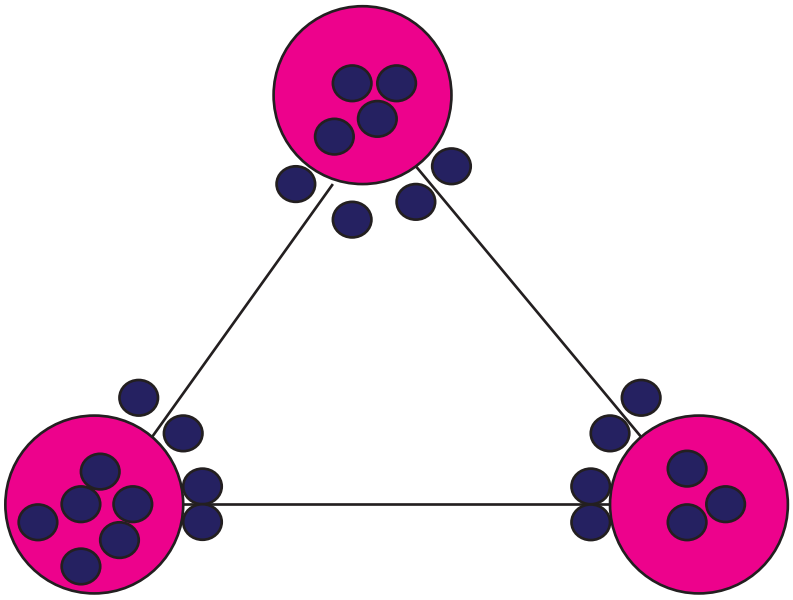

Figure 4.19: *B-IPGA*: Bullwhip Island Parallel Genetic Algorithm

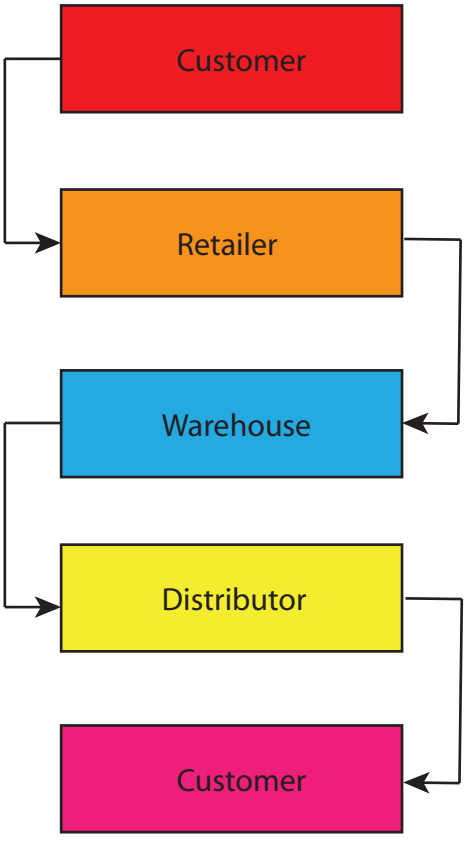

Figure 4.20: An Automotive Supply Chain Model

egy with *GA*s in probabilistic demand and supply conditions.

4.6.1 Modelling an Automotive *SC* as a Beer Game Model

MIT beer distribution game proposed by [112] is a generic template for *SC* modelling. The *SC* consists of Customer, Retailer, Warehouse, Distributor and Factory [115]. Decisions are amplified through *SC* to lower level participants. The beer concept is changed with automotive while preserving the same metrics. The retailer orders cars from warehouse whenever orders are high. This makes a domino effect and warehouse goes to distributor and distributor orders from the factory. The *SC* model considered is shown in Figure 4.19. The factory is considered to have limitless raw material. The *SC* has one period delay for each level. The system is deterministic and in an equilibrium. Orders are based on prevailing demand, expected demand, inventory, materials in production line, inventory request and inventory requests in production line.

$$tc = \sum_{i=1}^{n} \sum_{t=1}^{m} c_{i(t)} \tag{4.104}$$

The amplification among the layers are supposed to be reduced by participants. The cost of n members during m weeks is represented by Equation 4.104 [115]. $c_{i(t)}$ is represented by $inv_i(t) \times h_i + ufd_i(t) \times b_i$. $inv_i(t)$ is the inventory of participant i at week t while h_i is the inventory holding cost for participant i per unit week and b_i is the back order penalty cost of member i per week.

4.6.2 Parallel Genetic Algorithms for the Supply Chain Problem

Parallel computation tries to improve solution quality and time complexity of algorithms. GAs may divide the populations and compute the fitness values in parallel [147]. They emulate the process of evolution, survival and reproduction. The QAP-IPGA is an extension of the GA [147]. It generates non intersecting sub-populations and tries to speed up the search process. It calculates the fitness values of the populations in parallel nodes while changing the best individuals in initial steps. Initial populations are created by defining limits for respective nodes. The binary encoding is used because genes consist of bits. Crossover operator is used for the *QAP-IPGA* algorithm as it is better compared to first order and cycle crossovers [147]. It copies a random segment from first parent to the first child. Then, it looks for elements staying in second parent at corresponding part not copied from the prior crossover point. For each of the elements, it looks in the children to copy the corresponding element. For respective elements a, it looks in the child to see what element b is copied from first parent and places a in the place filled by b in second parent. If the place is already filled by c, set a to the position of c in second parent [59].

A new method called *B-IPGA* is presented to reduce the bullwhip effect. *B-IPGA* is a new algorithm using binary encoding as crossover and *IPGA* concept as model. Figure 4.20 shows how the *B-IPGA* algorithm operates. All the nodes transfer the best chromosomes to each other until a period of time. This limit is defined as %10 of the number of generations. Then, after the GA terminates, each node sends the best solution to the master node as in Figure 4.19. Master Node selects the best solution and announces as the best chromosome to reduce the bullwhip effect. The algorithm is shown in Algorithm 13. The Beer Game is used as an automotive SC model in this problem. The chromosome structure is the same with [115]. A bottom-up approach is used. Each layer of the SC requests immediately from the downstream participant. Thirty pieces of commodities are defined for each participant as in the

beer game. Each chromosome is represented with five bits. The *B-IPGA* is responsible for amplifying the demand to downstream layers.

Algorithm 13 *B-IPGA* Algorithm.

```
A) MASTER NODE:
broadcast IterationLimit to all slaves.
IPGAIteration = IterationLimit/10
while iteration < IPGAIteration do
   receive best results from all slaves.
   distribute set of best results to all slaves.
end while
receive best results from all slaves
print out best resulting chromosome and cost.
B) SLAVE NODE:
receive IterationLimit
generate populations
while  iteration < IterationLimit do
   if iteration < IPGAIteration then
      send best node to master
      receive best results of all nodes from master
   end if
end while
while population < populationLimit do
   (p1, p2) ← population
   CrossoverandMutation(p1, p2)
   add child to population
end while
truncate population
send best solution to master
```

CHAPTER 5

EXPERIMENTAL SETUP AND RESULTS

5.1 Integer Linear Programming Experiments for *MQO*

We performed our experiments on a computer with 2.53 GHz Intel Core 2 Duo CPU and 4 GB main memory. Python 2.7 is used for the A-Star, LP formulation and MOSEK [111] is used for the linear programming. The number of plans, tasks, and the execution time of the tasks used in the problem sets are randomly generated between the upper and lower limits of previously defined parameters [16]. A heuristic value *inputsize* is used to estimate the search space complexity of the MQO problem. The *inputsize* is defined by the multiplication of the number of queries q, the average number of tasks per plan t_{av}, and the average number of plans per query p_{av}.

In order to compare LP with A-Star algorithm, we first verified that both A-Star and LP algorithms produce the same results. Later, we compared the time spent by LP and A-Star algorithms to find an optimal solution on the same MQO problem instances. The values used in 4 different problem sets with varying number of queries, plans, and tasks are given in Table 5.1.

Table5.1: Parameters used in the experiments.

Set	*inputsize*	#queries	#plans per query	#tasks per plan
1	[625, 12,500]	**5 to 100**	[10,15]	[9,11]
2	[625, 3,375]	5	**[10,15] to [65,70]**	[9,11]
3	[625, 1,250]	5	[10,15]	**[9,11] to [19,21]**
4	[1,375, 5,500]	**25 to 100**	[4,7]	[9-11]

For each problem set, one of the parameters is increased gradually and the others are fixed to obtain MQO problem instances with increasing difficulties. In the first problem set t_{av} is fixed to 10, and p_{av} is fixed to 12.5 and the number of queries is varied between 5 and 100. In the second problem set, the number of queries is fixed to 5, t_{av} is fixed to 10, and p_{av} varies like [10,15]; [15,20] ,..., [65,70]. In the third problem set, the number of queries is fixed to 5, p_{av} is fixed to 12.5 and tasks per plan varies as [9,11]; [10,12] ,..., and [19,21]. The fourth problem set uses moderate size number of plans and tasks to show the capability of LP to provide optimal results without having an exponential optimization time. The results of the problem sets 1, 2, 3, and 4 are given in Figures 5.1, 5.2, 5.3, and 5.4 respectively.

It is clearly seen from the results given in Figures 5.1 and 5.2 that LP outperforms the A-Star algorithm

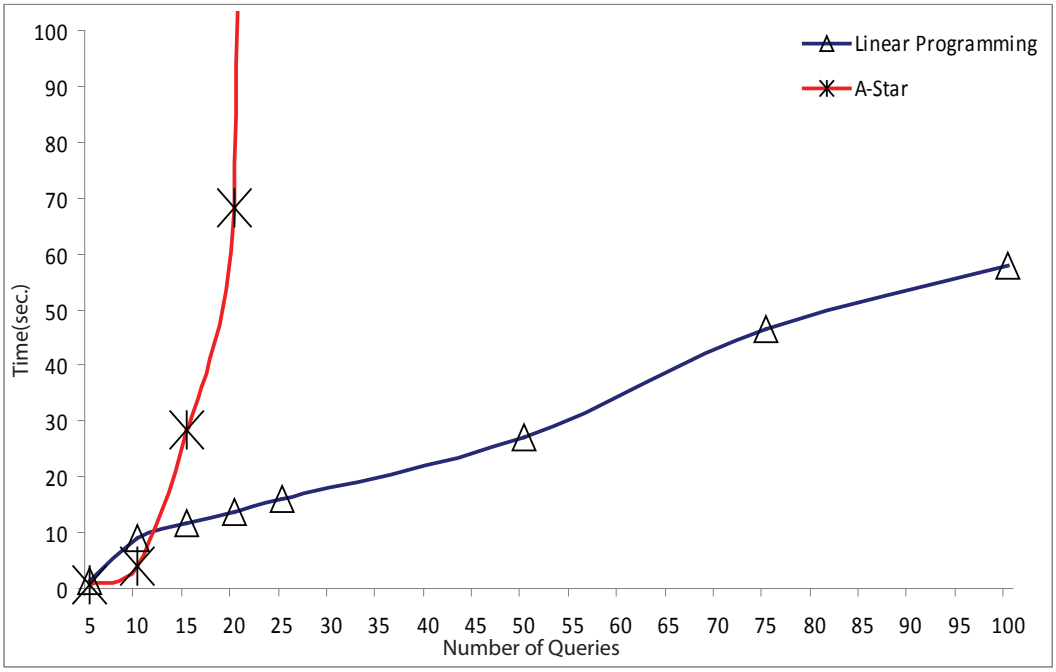

Figure 5.1: Performances of LP and A-Star algorithms for increasing number of queries where each query has [10-12] plans and each plan has [9-11] tasks (results of experiment 1).

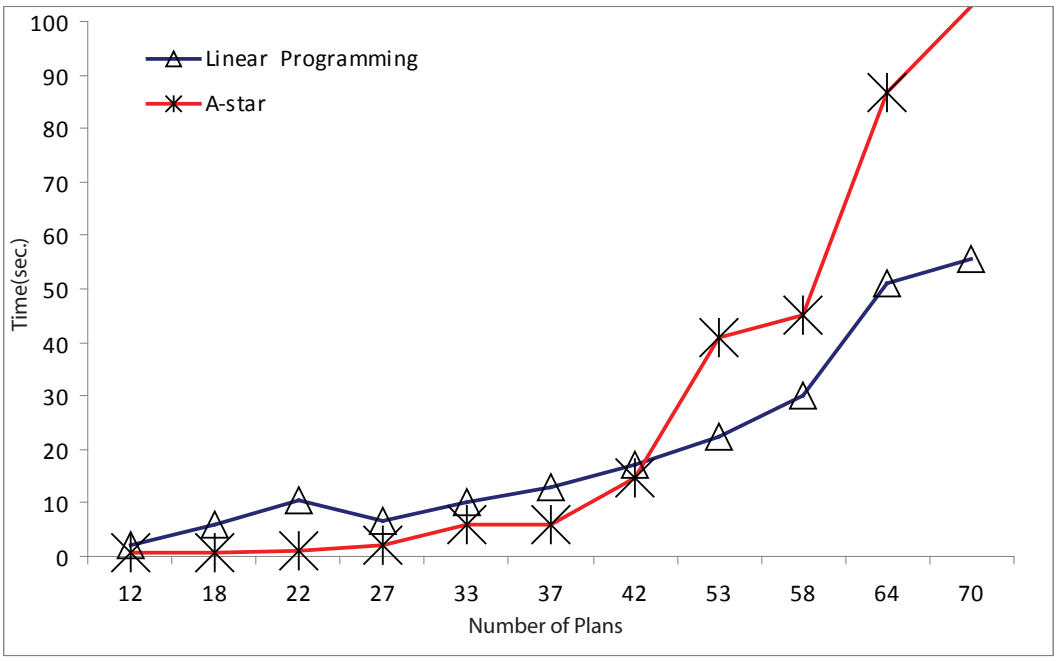

Figure 5.2: Performances of LP and A-Star algorithms for increasing number of plans where each query has [10-12] to [65,70] plans and each plan has [9-11] tasks (results of experiment 2).

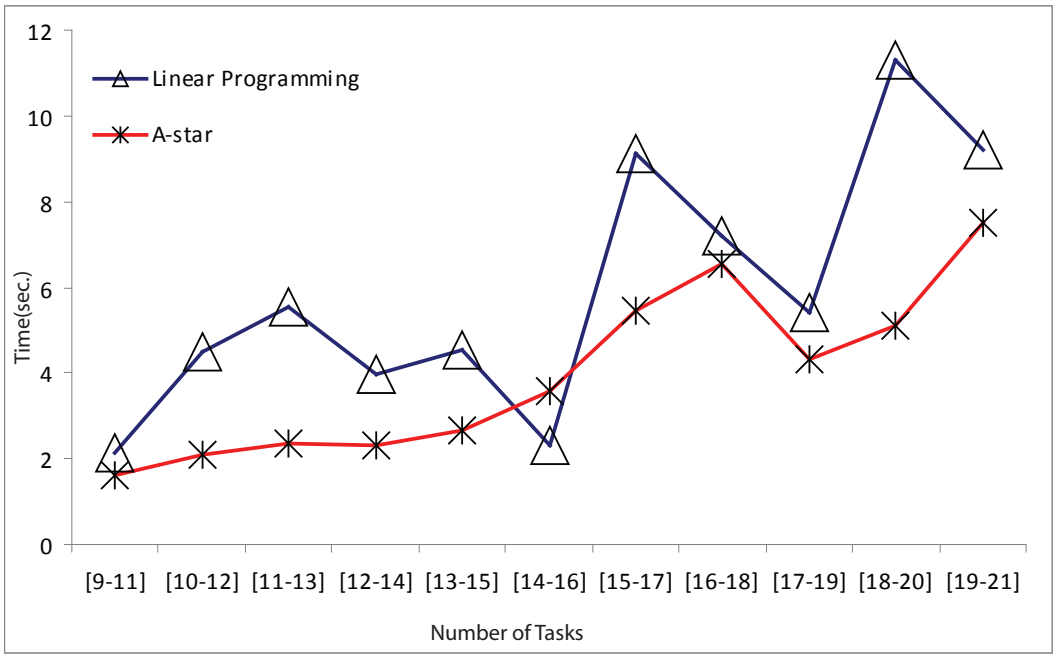

Figure 5.3: Performances of LP and A-Star algorithms for increasing task count where each query has [10-15] plans and each plan has [9-11] to [19-21] tasks (results of experiment 3).

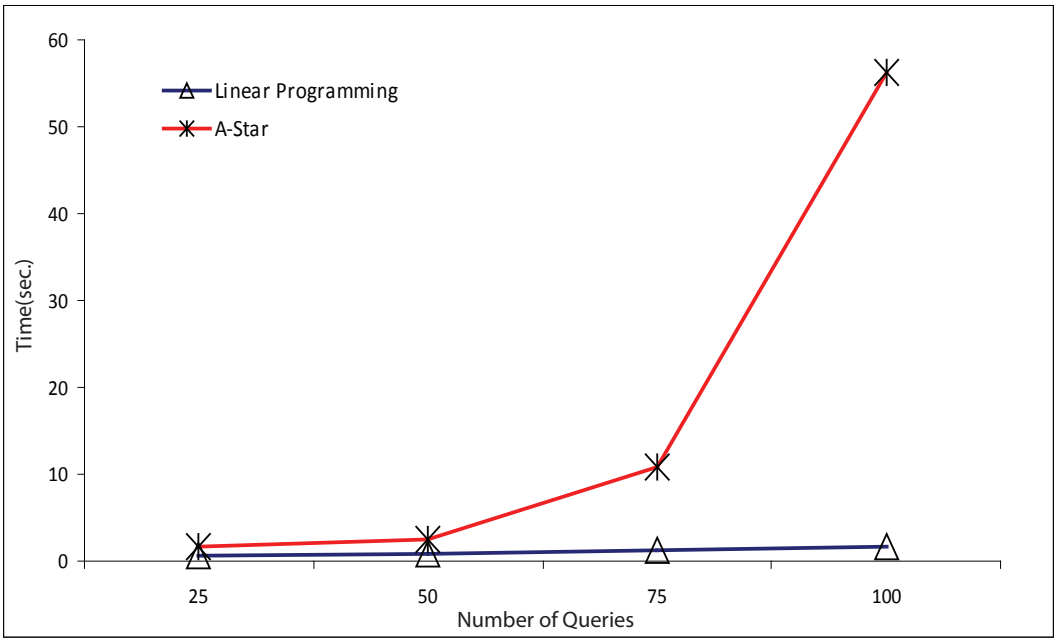

Figure 5.4: Performances of LP and A-Star algorithms up to 100 queries where each query has [4-7] plans and each plan has [9-11] tasks (results of experiment 4).

when the number of plans or queries is increased. However, due to the LP formulation of each task as a variable, the complexity of the LP formulation increases parallel with the number of tasks while this has less impact in the performance of A-Star algorithm (Figure 5.3). A-star can solve cannot solve MQO instances with more than 20 plans when number of alternative plans is more than 10 while LP performs remarkably well and can solve these MQO instances even when the number of queries reaches 100. Figure 5.4 gives the results of the problem set 4 where each query has [4-7] plans and each plan has [9-11] tasks where A-star is able to find the solution but its execution times increases exponentially while LP's running time increases linearly. A-star typically consumes more memory during these experiments. The optimization time of LP is much better than A-star algorithm when the number of queries is increased. By employing intelligent alternative plan generation techniques and keeping the number of tasks at minimum, it is guaranteed that the performance of LP will be much better than A-star and will have almost linear complexity in terms of the number of plans and queries.

5.2 Experimental Evaluation of the Island Parallel Genetic Algorithm

5.2.1 Experimental Environment

We tested our *QAP-IPGA* with data sets of the *QAPLIB*. The *QAPLIB* serves as a library of benchmark problem instances for the researchers working on the *QAP* [20]. Most of the proposed algorithms for solving the *QAP* are tested on these benchmark instances. There are more than 100 instances that are derived either from randomly generated problem instances or real life applications. Hospital layout (kra30), Manhattan distances of rectangular grids (Had12), and the backboard wiring (Ste36a) are some of the problem instances that can be found in *QAPLIB*. Literature surveys, source codes, and the *QAP* related research people are other resources for the problem instances in this library.

Our experiments are performed on a cluster computer. A cluster is a group of loosely coupled computers working together closely, so that in many aspects they can be thought as a single computer [12]. Clusters are typically used for "high availability" to achieve greater reliability or high performance computing to provide greater computational power than a single computer can provide. The key components of a cluster include, multiple stand alone computers, an operating system, a high performance interconnect, communication software, middleware, and applications.

Complete configuration of our *HPC* system is; 46 nodes, each with 2 *CPU*s giving 92 *CPU*s. Each *CPU* has 4 cores giving a total number of 368 cores. Each node has 16GB of RAM giving 736GB of total memory and 146GBs of disk storage per node, giving 6.5TB total disk space. In addition, there are 2 high performance disk units each with 3 TB giving 6 TB common storage area. For providing high-bandwidth communication among *HPC* nodes, two 24 port Gigabit Ethernet Switches, and one 24 port Infiniband switch are used.

Software installed on *HPC* is; Scientific Linux v4.5 64-bit operating system, Lustre v1.6.4.2 parallel file system, Torque v2.1.9 resource manager, Maui v3.2.6 job scheduler, and OpenMPI v1.2.4.

When developing *QAP-IPGA*, we used *C* + + programming language and *MPI* libraries [89]. *MPI* is a specification for Message Passing Libraries, designed to be a standard for the distributed memory message-passing parallel computing. *MPI* defines primarily the message-passing parallel programming model, in which data is moved from the address space of one processor to that of another processor through cooperative send and receive operations on each processor.

We ran each test 10 times. Different numbers of generations, populations, and CPUs are used in the

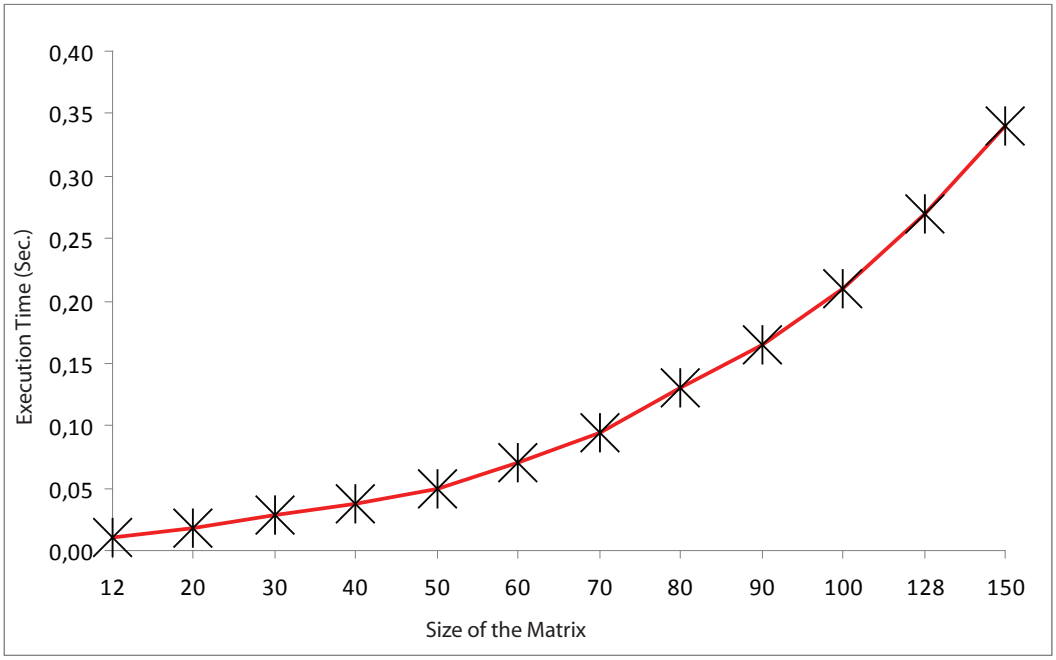

Figure 5.5: The cost of fitness evaluation for the individuals (Wall Clock Time).

tests (Table 5.2). The number of processors and the population size of *GA* are increased until the best solutions can be produced by the *QAP-IPGA*.

Table5.2: Parameters of different test types for *QAP-IPGA*.

Test Type	Generations	Population	Number of Processors
1	1,000	10,000	30
2	2,000	20,000	30
3	3,000	30,000	50
4	3,000	30,000	100
5	3,000	50,000	240

5.2.2 The effect of fitness evaluation for larger data sets

The fitness evaluation of the individuals requires some costly operations involving many multiplications & summations, and becomes more CPU intensive as the problem size grows. Figure 5.5 shows how the time required to evaluate fitness values increases. Population and generation sizes are fixed during these tests in order to discover the effect of solving larger problem instances (represented as square matrices). The cost of evaluating the fitness of a 150x150 matrix is 35 times more than a 12x12 matrix.

5.2.3 Experiments with the Parallel Exhaustive Algorithm

First we tested the efficiency of our *PEA*. By using *PEA* alone we were able to find optimal results for problem instances with up to 16 facilities and locations. Although parallel algorithms are scalable and provides linear speedup when given more computational power, the intractable nature of the *QAP* prevents us from exhaustively solving larger problem instances in practical times. Figure 5.6 shows the running times of *PEA* with the increasing number of *CPU*s for the instance of Char12a.

Table 5.3 shows the running times and the results of the *PEA* for instances with up to 16 locations using 200 processors. It took almost 3 days to find the exact result of the Had16 instance.

5.2.4 Choosing the Best Crossover Operator and the Best Selection Mechanism

The crossover operators and the selection mechanisms were evaluated with different problem instances containing up to 90 locations, and with test type 2 in Table 5.2 to decide the best performing parameters. After the tests it was observed that tournament selection is the fastest among the selection techniques since it does not require expensive sort operations.

Table5.3: The results of the Parallel Exhaustive Algorithm tests.

Instance	**n**	**Result**	**Exhaustive Execution(min)**
Chr12a	12	9,552	0.2
Had14	14	2,724	16.2
Chr15a	15	9,896	375
Had16	16	3,720	4,037 (3 days)

On the other hand, roulette wheel is the slowest one because it requires sorting and construction of roulette wheel from the sorted individuals. Truncation requires only sorting. However, the execution times of crossover operators and selection mechanisms are negligible. Because the execution times of *QAP-IPGA* do not vary much, deviation of the results from the best known solutions can be used to rank the algorithms. *PEA* gives almost linear speed-up as the number of processors increases (Figure 5.6).

The truncation selection is capable of providing better initial populations. This property of truncation selection makes it more favorable than the other selection mechanisms. From the results in Table 5.4 (*Truncation-Tr*, *Tournament-To*, *Roulette Wheel-RW*), we can see that *PMX* crossover operator is the best performing one when it is applied with the truncation selection. Subsequent tests with all the instances of the *QAPLIB* are performed using *PMX* crossover and truncation selection mechanism to evaluate the performance of the developed *QAP-IPGA* algorithm.

5.2.5 Parameter settings for the *QAP-IPGA*

The number of populations and generations, the size and the initialization of the population, the migration frequency and rate, reducing the communication overhead, and determining the employed parallel architecture and the communication topology are the main issues to be decided in designing the *QAP-IPGA*.

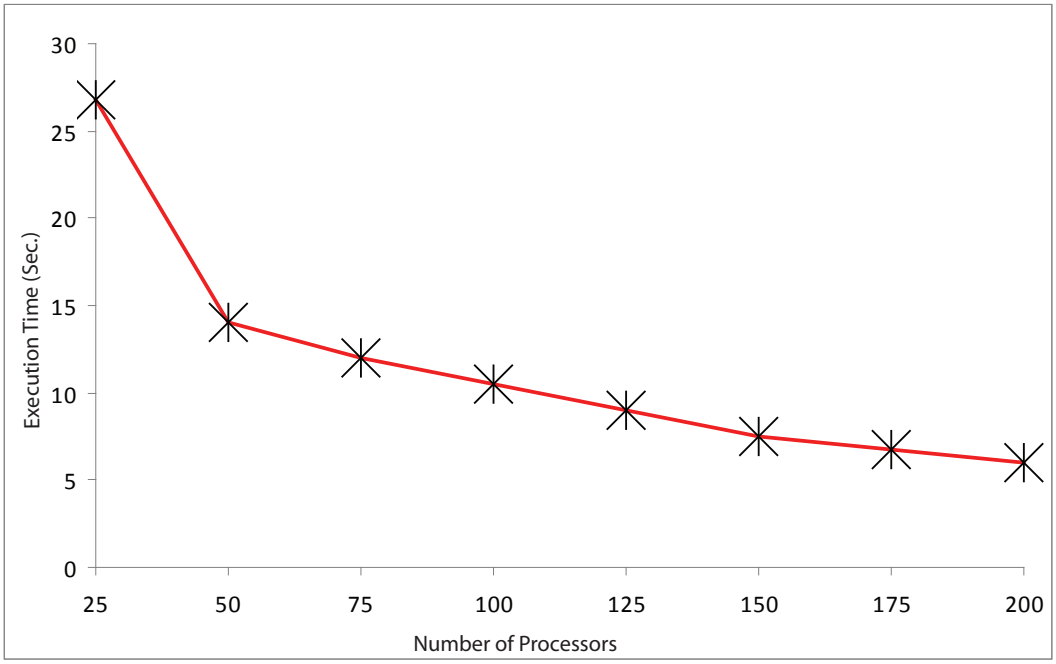

Figure 5.6: PEA's running time with increasing number of processors for the Chr12a instance.

Before studying different instances, we performed a few tests concerning these parameters with the *QAPLIB* instance Sko100a. The crossover operator was *PMX* and the selection mechanism was truncation during these tests. We ran every test set 10 times and took the average values of the results. The percentage value of the diagrams on Y-Axis shows how the quality of *QAP-IPGA* solutions improves by comparing with the best *QAPLIB* result given for the Sko100a instance.

Table5.4: The deviation percentages of crossover and selection mechanisms on different instances.

	Order1 (%)			***PMX (%)***			***Cycle (%)***		
Instance	***Tr***	***To***	***RW***	***Tr***	***To***	***RW***	***Tr***	***To***	***RW***
Chr12a	0	1.8	0	0	1.4	0	3.6	4.7	4.2
Chr15a	2.1	3.4	4.1	0	2	0	1.3	3.5	2.7
Chr18a	6.5	7.1	9.4	0	8.4	1.4	2	3.7	1.5
Chr20a	8	7	4.3	0	3	1.6	4	3	2.1
Chr25a	6.2	6.2	7.4	1	1.8	2.4	2.4	1.6	6.7
Nug30	3.9	4.2	5.4	0.4	1.2	2.6	5.7	6.9	6.4
Lipa50	4.7	7.2	9.5	2.1	1.2	2.1	7.3	9.9	8.2
Sko64	8.5	9.4	12.3	2.4	2.8	2.5	11.2	14.1	16.2
Lipa90	12.5	15.4	17.4	3	3.2	4	14.2	18.5	17.4

In this part of our tests, the results are given in Figure 5.7, the number of processors, generations, and the size of the population are increased all together to see how the *QAP-IPGA* performs. Configurations of the test parameters are summarized in Table 5.5. For obtaining the parameters for the next level of the tests, we doubled the size of the current level's parameters. With the increasing parameter values, the *QAP-IPGA* runs efficiently and reaches down to 1 % deviation of the best value of Sko100a instance

Table5.5: Types of tests with different configurations.

Configuration Number	Generations	Population	Number of Processors
1	10	250	2
2	20	500	4
3	50	1,000	8
4	100	2,000	16
5	200	5,000	32
6	400	10,000	64
7	800	20,000	128
8	1,600	40,000	256

with the most powerful configuration. It can be seen that *QAP-IPGA* still has potential for improving its performance by using more resources.

5.2.6 Execution time and the accuracy of *QAP-IPGA*

We can see the results of *QAP-IPGA* in Table 5.6 (*BKS*: Best known solution, *PDB*: Percentage of deviation from *BKS*, TT: Test configuration type). We have solved a comprehensive set of instances from the *QAPLIB* with problem sizes ranging from 12 up to 150 locations. (The best permutations are given in Appendix A.) The parameters of the reported test types used in this work can be seen in Table 5.2. When the gap between our solutions and the best solutions reported in the *QAPLIB* started increasing, we applied the next level of more powerful (and more time consuming) test configurations. In our experiments, we have preferred to use denser matrix problem instances from the library. The accuracy of the algorithm is calculated by using the value of *PDB* (The Percentage Deviation of the best result of *QAP-IPGA* after 10 runs). The smaller the average *PDB*, the more robust is the evaluated algorithm.

QAP-IPGA has a very robust performance. When the parameter settings and the execution time are reasonable *QAP-IPGA* can discover the best results for most of the data sets smaller than 30 locations. For ($30 < n < 150$) *QAP-IPGA* is capable of producing solutions with an average *PDB* gap of 1 %.

We devoted a great deal of effort to determine the best parameters for the *QAP-IPGA*. This was one of the hardest parts of this study. After performing extensive tests we obtained the values given in Table 5.2. The performance of *QAP-IPGA* can be further improved by increasing the population sizes and the number of processors, depending on the RAM and processor resources available for the *QAP-IPGA*.

We have selected the parameters of *QAP-IPGA* to obtain best possible results, while also considering the *HPC* architecture configuration we are using. From our experiments we conclude that the simplest way to improve the performance of an *IPGA* is to increase the number of allocated processors. For the connected *IPGA*s, this may increase the execution time of the algorithm, but by migrating only a small number of individuals in each generation the communication overhead can be minimized while still benefitting from migration.

Table5.6: The gap between *QAP-IPGA* solutions and best known solutions in *QAPLIB*. (Best of 10 runs reported.)

Instance	**BKS**	**PDB**	**Time (min)**	**Gap (%)**	**TT**
Chr12a	9,552	0	1.5	0	1
Chr12b	9,742	0	1.5	0	1
Chr12c	11,156	0	1.5	0	1
Rou12	235,528	0	1.5	0	1
Tai12a	224,416	0	1.5	0	1
Nug12	578	0	1.5	0	1
Nug14	1,014	0	9.9	0	2
Chr15a	9,896	0	10.1	0	2
Nug15	1,150	0	10.1	0	2
Rou15	354,210	0	10	0	2
Esc16a	68	0	11.7	0	2
Esc16h	996	0	11.7	0	2
Had16	3,720	0	9.9	0	2
Nug16a	1,610	0	9.8	0	2
Nug17	1,732	0.1	10.2	0	2
Had18	5,358	0	10.3	0	2
Nug18	1,930	0	10	0	2
Chr20a	2,192	0	11.4	0	2
Had20	6,922	0	10	0	2
Lipa20a	3,683	0	10	0	2
Nug20	2,570	0	10.2	0	2
Tai20a	703,482	0.1	32.2	0	3
Nug22	3,596	0.2	10.4	0	2
Nug24	3,488	0.3	10.3	0	2
Nug25	3,744	0.5	12.6	0	2
Bur26a	5,426,670	0.2	14.6	0	2
Bur26b	3,817,852	0.15	14.7	0	2
Nug27	5,234	1.1	28.6	0	3
Lipa30a	13,178	1.4	33.6	0	3
Lipa30b	151,426	1.1	33.9	0	3
Nug30	6,124	1.3	33.4	0	3
Tho30	149,936	1.1	34.2	0	4
Tai30a	1,818,146	2.1	34.1	0.62	4
Tai35a	2,422,002	2.1	33.3	0.82	4
Ste36a	9,526	1.2	34.7	0.8	4
Lipa40a	31,538	2.1	34.2	0.49	4
Tai40a	3,139,370	1.4	34.1	0.7	4
Sko42	15,812	2.2	33.3	0.53	4
Sko49	23,386	1.4	34.7	0.74	4
Lipa50a	62,093	1.2	35.5	0.84	4
Wil50	48,816	1.3	35.9	0.35	4
Lipa60a	107,218	2	44.1	0.72	4
Sko64	48,498	2.2	45.1	0.35	4
Lipa70a	169,755	1.7	47.1	0.9	4
Lipa80a	253,195	2.4	48.4	0.86	4
Lipa90a	360,630	2.2	51.1	0.78	4
Sko100a	152,002	1.6	52.4	0.29	4
Sko100c	147,862	1.7	53.3	0.38	4
Sko100e	149,150	2.44	52.7	0.43	4
Tho150	8,133,398	2.45	57.9	0.94	5
Tai150b	498,896,643	2.5	57.3	0.79	5

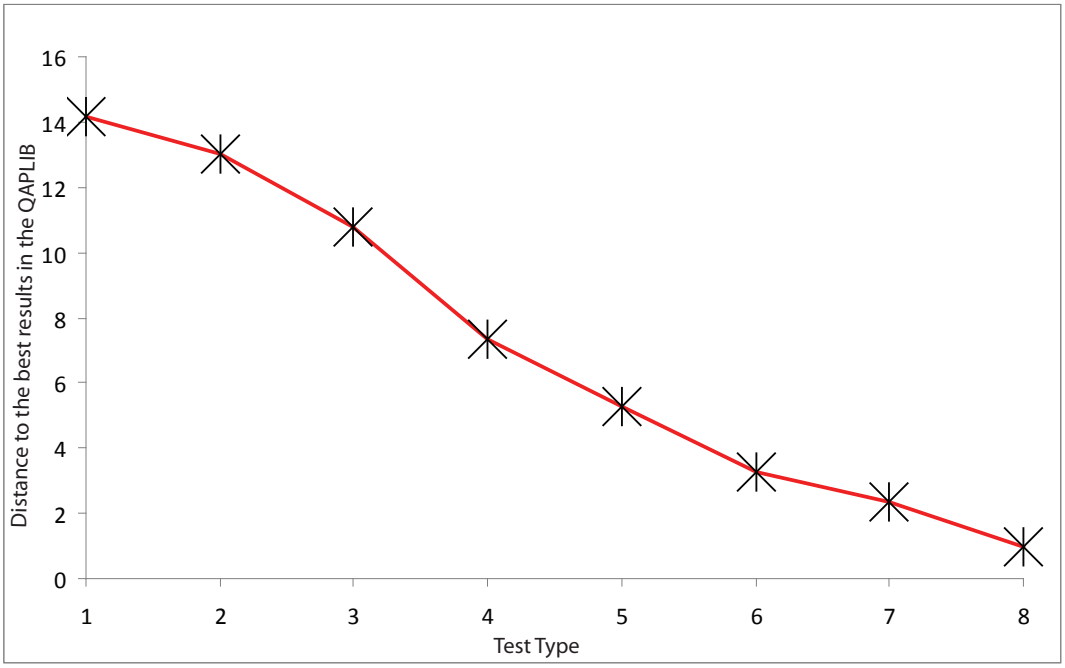

Figure 5.7: Change in solution quality of *QAP-IPGA* as number of generations, populations, processors increases (Table 5.5).

5.2.7 Performance comparison with other algorithms in the literature

The experimental results given in this work can be compared with the results of other approaches published in the literature by comparing their optimization times and the quality of solutions discovered by them. Algorithms such as dynamic programming and branch-and-bound can hardly solve problem instances of size 15-30 for which our algorithms have consistently found the same results within a few minutes in every repeated execution.

The optimization times of algorithms such as simulated annealing, genetic algorithms, and their hybrids can be kept small and stopped after a predetermined time period. The performance of such algorithms can be compared against our algorithm based on the quality of solutions they find for *QAPLIB* problem instances because all of these works report experimental results for *QAPLIB* instances. These algorithms can be easily used for solving large *QAP* instances with around 100 facilities. For such instances, exact results are not known and only the best results reported in the literature are given in *QAPLIB* without much information on optimization times and/or details of the parallel architecture used in the experiments.

One final comparison of algorithms can be made in terms of the scalability of parallel implementations with increasing numbers of processors. Our *IPGA* formulation allows us to increase the number of processors while keeping the amount of communication between processors under tight control. Our results show that almost linear speed-up is achieved for upto 50 processors while the speedup is half of that for 50 to 200 processors, and there is no loss in solution quality.

Table5.7: Parameter settings for DivTS.

Maximum Number of Failures	Allowable Failures		Tabu Tenure		Aspiration Value
	Lower Limit	Upper Limit	Lower Limit	Upper Limit	
50,000×n	5000×n	5000×n	(n/10) (variable)	(11×n)/10 (variable)	n×n×2

Table5.8: Parameter settings for RTS.

Maximum Number of Failures	Allowable Failures		Tabu Tenure		Aspiration Value
	Lower Limit	Upper Limit	Lower Limit	Upper Limit	
500,000-2,000,000	NA	NA	((9×n/10)) (static)	(11×n)/10 (static)	n×n×2

Table5.9: The results of the PHGETS tests with different instances of the QAPLIB. (Each data set was tested 10 times).

Instance	***Best Known Solution (BKS)***	***APD***	***#BKS***	***Wall Clock Time (min.)***	***Total CPU Time (min.)***	***Instance***	***Best Known Solution (BKS)***	***APD***	***#BKS***	***Wall Clock Time (min.)***	***Total CPU Time (min.)***
Chr12a	9552	0	10	0.15	1.47	Bur26a	5426.67	0	10	0.62	6.73
Chr12b	9742	0	10	0.15	1.55	Bur26b	3817852	0	10	0.84	8.38
Chr12c	11156	0	10	0.15	1.47	Nug27	5234	0	10	0.78	7.52
Rou12	235528	0	10	0.15	1.46	Lipa30a	13178	0	10	1.05	9.09
Tai12a	224416	0	10	0.15	1.46	Lipa30b	151426	0	10	0.84	8.93
Tai12b	39464925	0	10	0.15	1.46	Nug30	6124	0	10	0.88	9.01
Nug12	578	0	10	0.17	1.48	Tai30a	1818146	0	10	0.97	8.89
Nug14	1014	0	10	0.19	1.92	Tho30	149936	0	10	0.84	8.84
Chr15a	9896	0	10	0.23	2.23	Tai35a	2422002	0	10	1.28	12.34
Nug15	1150	0	10	0.22	2.22	Ste36a	9526	0	10	1.37	12.76
Rou15	354210	0	10	0.23	2.23	Lipa40a	31538	0	10	1.52	15.21
Esc16a	68	0	10	0.25	2.53	Tai40a	3139370	0	10	10.6	100.83
Esc16h	996	0	10	0.24	2.47	Sko42	15812	0	10	1.6	16.54
Had16	3720	0	10	0.25	2.49	Sko49	23386	0	10	4.03	40.18
Nug16a	1610	0	10	0.24	2.49	Lipa50a	62093	0	10	12.62	126.37
Nug17	1732	0	10	0.28	2.78	Wil50	48816	0	10	12.6	126.3
Had18	5358	0	10	0.3	3.06	Lipa60a	107218	0	10	19.53	195.06
Nug18	1930	0	10	0.31	3.07	Sko64	48498	0	10	23.1	230.78
Chr20a	2192	0	10	0.37	3.73	Lipa70a	169755	0	10	28.23	298.78
Had20	6922	0	10	0.38	3.73	Lipa80a	253195	0	10	39.3	394.41
Lipa20a	3683	0	10	0.38	3.71	Lipa90a	360630	0	10	40.51	407.48
Nug20	2570	0	10	0.37	3.73	Sko100a	152002	0	10	41.72	420.93
Tai20a	703482	0	10	0.37	3.73	Sko100c	147862	0	10	42.19	410.63
Nug22	3596	0	10	0.45	4.46	Sko100e	149150	0	10	42.47	423.05
Nug24	2488	0	10	0.51	5.23	Tho150	8133398	0.009	0	177.41	1767.99
Nug25	3744	0	10	0.62	6.49	Tai150b	498896643	0.026	0	177.44	1777.48

Table5.10: Comparison of the PHGETS with CPTS and sequential algorithms on the Skorin-Kapov instances.

Problem	*BKS*	*PHGETS*		*CPTS*		*JRG-DivTS*		*TL-ACO/GA/LS*		*D-GA/SD*		*D-GA/S-TS*		*D-GA/C-TS*		*D-GA/IC-TS*	
Skorin-Kapov instances		*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*
Sko42	15812	0	1.6	0	5.3	0	4	0	0.7	0.014	0.16	0.001	0.3	0	1.2		
Sko49	23386	0	4.03	0	11.4	0.008	9.6	0.056	7.6	0.107	0.28	0.062	0.5	0.009	2.1		
Sko56	34458	0	16.24	0	21	0.002	13.2	0.012	9.1	0.054	0.42	0.007	0.7	0.001	3.2	0	13.6
Sko64	48498	0	23.1	0	42.9	0	22	0.004	17.4	0.051	0.73	0.019	1.2	0	5.9	0	26.2
Sko72	66256	0	33.63	0	69.6	0.006	38	0.018	70.8	0.112	0.93	0.056	1.5	0.014	8.4	0	38.3
Sko81	90998	0	39.87	0	121.4	0.016	56.4	0.025	112.3	0.087	1.44	0.058	2.2	0.014	13.3	0.003	63.1
Sko90	115534	0	40.53	0	193.7	0.026	89.6	0.042	92.1	0.139	2.31	0.073	3.5	0.011	22.4	0.001	102.3
Sko100a	152002	0	41.72	0	304.8	0.027	129.2	0.021	171	0.114	3.42	0.07	5.1	0.018	33.6	0.002	177.6
Sko100b	153890	0	42.32	0	309.6	0.008	106.6	0.012	192.4	0.096	3.47	0.042	5.1	0.011	34.1	0	170.2
Sko100c	147862	0	42.19	0	316.1	0.006	126.7	0.005	220.6	0.075	3.22	0.045	4.7	0.003	33.8	0.001	158.4
Sko100d	149576	0	41.91	0	309.8	0.027	123.5	0.029	209.2	0.137	3.45	0.084	5.2	0.049	33.9	0	164.2
Sko100e	149150	0	42.47	0	309.1	0.009	108.8	0.002	208.1	0.071	3.31	0.028	4.7	0.002	30.7	0	169.6
Sko100f	149036	0	41.98	0.003	310.3	0.023	110.3	0.034	210.9	0.148	3.55	0.11	5.3	0.032	35.7	0.003	174.6
Average		0	27	0	178.8	0.012	72.1	0.02	117.1	0.093	2.1	0.05	3.1	0.013	19.9	0.001	114.4

Table5.11: Comparison of PHGETS with CPTS and sequential algorithms on the symmetric and asymmetric Taillard instances.

Problem	*BKS*	*PHGETS*		*CPTS*		*JRG-DivTS*		*M-ETS-1*	*M-ETS-2*	*M-ETS-3*		*TL-ACO/GA/LS*		*M-GA/TS*	*M-GA/TS-1*	
Taillard symmetric instances																
		APD	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*APD*	*APD*	*Time (min.)*	*APD*	*Time (min.)*	*APD*	*APD*	*Time (min.)*
Tai20a	70382	0	0.37	0	0.1	0	0.2	0	0	0	0			0.061	0	0
Tai25a	1167256	0	0.55	0	0.3	0	0.6	0.037	0	0.015	0.1			0.088	0	0.1
Tai30a	1818146	0	0.97	0	1.6	0	1.3	0.003	0.041	0	0.2	0.341	1.4	0.019	0	0.3
Tai35a	2422002	0	1.28	0	2.3	0	4.4	0	0	0	3.7	0.487	3.5	0.126	0	0.6
Tai40a	3139370	0	10.6	0.148	3.5	0.222	5.2	0.167	0.13	0.173	28.3	0.593	13.1	0.338	0.209	1.4
Tai50a	4938796	0	12.74	0.44	10.3	0.725	10.2	0.322	0.354	0.388	116.7	0.901	29.7	0.567	0.424	5
Tai60a	7205962	0	19.58	0.476	26.4	0.718	25.7	0.57	0.603	0.677	116.7	1.068	58.5	0.59	0.547	12
Tai80a	13499184	0.644	39.97	0.69	94.8	0.874	52.7	0.442	0.511	0.526	200	1.3	152.2	0.392	0.441	53.3
Tai100a	21052466	0.537	71.89	0.589	261.2	0.856	142.1	0.398	0.402	0.472	666.7	1.146	335.6	0.327	0.275	200
Average		0.131	17.55	0.26	44.5	0.377	26.9	0.215	0.227	0.25	125.8	0.834	84.9	0.279	0.211	30.3
Taillard asymmetric instances																
Tai20b	122455319	0	0.37	0	0.1	0	0.2	0	0	0	0			0	0	0
Tai25b	344355646	0	0.57	0	0.4	0	0.5	0	0	0	0			0.007	0	0
Tai30b	637117113	0	0.81	0	1.2	0	1.3	0	0	0	0.1	0	0.3	0	0	0
Tai35b	283315445	0	1.11	0	2.4	0	2.4	0	0.019	0	0.2	0	0.3	0.059	0	0.1
Tai40b	637250948	0	1.57	0	4.5	0	3.2	0	0	0	0.2	0	0.6	0	0	0.1
Tai50b	458821517	0	5.82	0	13.8	0	8.8	0	0.003	0	4.5	0	2.9	0.002	0	0.3
Tai60b	608215054	0	9.49	0	30.4	0	17.1	0	0.001	0.003	22.5	0	2.8	0	0	0.7
Tai80b	818415043	0	27.7	0	110.9	0.006	58.2	0.008	0.036	0.016	116.7	0	60.3	0.003	0	2.5
Tai100b	1185996137	0	42.5	0.001	241	0.056	118.9	0.072	0.123	0.034	333.3	0.01	698.9	0.014	0	7.3
Average		0	9.99	0	45	0.007	23.4	0.009	0.02	0.006	53.1	0.001	109.4	0.009	0	1.2
Overall		0.057	13.77	0.122	44.7	0.184	25.2	0.104	0.115	0.12	89.4	0.407	97.1	0.136	0.098	15.8

Table5.12: Comparison of the PHGETS results with other parallel algorithms.

Instance	***Best Solution***	***PHGETS***		***TB-MTS***	***TB-COSEARCH***	***CPTS***	
		APD	***Time***	***APD***	***APD***	***APD***	***Time (min.)***
tai25a	1167256	0	0.83	0.736	0.736	0	0.3
bur26d	3821225	0	0.89	0	0	0	0.4
nug30	6124	0	0.88	0	0	0	1.7
tai35b	283315445	0	1.28	0	0	0	2.4
ste36c	8239110	0	1.37	0	0	0	2.5
lipa50a	62093	0	12.62	0	0	0	11.2
sko64	48498	0	23.1	0.004	0.003	0	42.9
tai64c	1855928	0	12.94	0	0	0	20.6
tai100a	21052466	0.537	71.89	0.814	0.544	0.589	261.2
tai100b	1185996137	0	42.5	0.397	0.135	0.001	241
sko100a	152002	0	41.72	0.073	0.054	0	304.8
wil100	273038	0	41.97	0.035	0.009	0	316.6
tai150b	498896643	0.026	177.44	1.128	0.439	0.076	1549.4
tho150	8133398	0.009	177.41	0.012	0.065	0.013	1991.7
Average		0.041	43.35	0.229	0.142	0.049	339.05

5.3 Experimental Evaluation of *PHGETS*

5.3.1 Experimental Environment

We tested our *PHGETS* with a comprehensive data set from the *QAPLIB* which is as a library of benchmark problem instances for researchers dealing with the *QAP* [19]. Most of the published algorithms for solving the *QAP* are tested on these benchmark instances. There are more than 100 instances that are derived either from real life applications or randomly generated problem instances such as the hospital layout (kra30), Manhattan distances of rectangular grids (Had12), and the backboard wiring (Ste36a). Our experiments were performed on a cluster computer which has 46 nodes, each with 2 *CPU*s giving 92 *CPU*s. Each CPU has 4 cores giving a total of 368 cores. Each node has 16GB of RAM giving 736GB of total memory, and a total disk capacity of 6.5TB configured in a high performance RAID. High-bandwidth communication is available among nodes, using two 24 port *Gbps* ethernet switches, and one 24 port infiniband switch, providing very low latency messaging with a capacity of 8Gbps. *PHGETS* was developed in C++ language and the *MPI* libraries [89] were used.

To minimize measurement errors, each test was run 10 times and the averages were reported. The crossover strategy was selected as *PMX* with a generation limit of 100 and an initial population size of 5,000 was selected. These values have been previously defined as best performing values for the *QAP* [147]. *DivTS* [84] is a multi-start algorithm which uses diversified working solutions. It changes the search strategy by modifying the working solution with a diversified permutation. *PHGETS* uses a similar diversification strategy but it uses a different diversification operator. The *TS* diversification phase uses the best execution parameters of the *DivTS* as given in Table 5.7. The decision phase uses the best working parameter settings for *RTS* as given in Table 5.8. In keeping with the reported results of the CPTS; 10 processors were used in order to make a fair comparison.

In Table 5.7 and Table 5.8, maximum number of failures define the number of iterations for *TS* to run.

The allowable failure limit defines the number of iterations to diversify when there is no improving move found. Tabu tenure upper and lower limit sizes define the tabu list size. Finally, the aspiration criteria is used to make a decision about the acceptance of a tabu move.

5.3.2 Analysis of execution time and the accuracy of PHGETS

The performance of the *PHGETS* can be seen in Table 5.9. (The resulting permutations are given in Appendix A.) We solved a comprehensive set of instances from the *QAPLIB* with problem sizes ranging from 12 up to 150 locations. In our experiments, we chose to use dense matrix problem instances from the library. The accuracy of the algorithm was calculated by using the value of *APD* (The Average Percentage Deviation of the *PHGETS* after 10 runs). The smaller the average *APD*, the more robust the evaluated algorithm. The Total CPU Time is the sum of CPU times used by all processors whereas the Wall Clock Time is the CPU time used by only one processor during the execution of the *PHGETS* algorithm.

Our results show that *PHGETS* is a very robust algorithm. With the given parameter settings and in reasonable execution times, *PHGETS* can find the best results for most of the data sets smaller than 100 locations. For problem sizes ranging from 100 to 150 locations, *PHGETS* is capable of producing solutions with an *APD* gap of 0.05 %. We carefully tuned the parameters for the *PHGETS* algorithm, which takes a long time since the large problem instances require hours and days to solve. The performance of the *GA* can be further improved by increasing the population sizes and the number of processors. However, since our aim was also to determine minimal parameters for the *QAP* to achieve these competitive results, we only used 10 processors with sufficiently large populations.

5.3.3 Comparing *PHGETS* with other algorithms

We compared the results of *PHGETS* with the best performing recent sequential and parallel algorithms for the *QAP* in the literature. The sequential algorithms we used were: Ant Colony Optimization/*GA*/Local Search Hybrid (*TL-ACO/GA/LS*) [146], Multi-Start *TS* Algorithm *JRG-DivTS* [84], *GA* Operator *D-GA/IC-TS* [58], *GA* Hybrid with Concentric *TS* Operator *D-GA/C-TS*, *GA* Hybrid with a Strict Descent Operator *D-GA/SD*, *GA* Hybrid with a Simple *TS* Operator *D-GA/S-TS* [57], *TS* Variant (*M-ETS-1*, *M-ETS-2*, *M-ETS-3*) [110], and *GA* Hybrids with a *TS* (*M-GA/TS*, *M-GA/TS-I*) [109]. We compared the following parallel algorithms: Independent Parallel *TS* (*TB-MTS*) [144], Cooperative Parallel *TS* Hybrid with a *GA* (*TB-COSEARCH*) [144], and the Cooperative Parallel *TS* (*CPTS*) [83]. First we compared our results with the prevailing methods to show the need for parallelization for the problem as shown in Tables 5.10 and 5.11.

Later, we compared our algorithm with the best parallel algorithms given in the literature as shown in Table 5.12. *PHGETS* outperforms all of these methods in terms of solution quality and running times.

Table 5.12 contains a summary of the results of the comparison between our algorithm and the Cooperative Parallel *TS* (*CPTS*), Independent Parallel *TS* (*TB-MTS*), and a Cooperative *TS* hybrid with a *GA* (*TB-COSEARCH*). To the best of our knowledge, these algorithms are the only published parallel algorithms available for comparison. The other parallel algorithms from the literature either work on different problem domains or only report on single run results related to timing and solution quality which is usually the best case result for that algorithm. The results in Table 5.12 are obtained in different platforms. Therefore, the results are shown just to give an idea, but it is not so fair to compare the performance of these algorithms related to execution time since they are executed in different

platforms. Even though the platforms are different, we know that the solution quality of *PHGETS* outperforms the other algorithms.

Talbi and Bachelet presented their well-known parallel algorithms *TB-MTS* and *TB-COSEARCH* [144]. The Cooperative Parallel *TS* (*CPTS*) [83] outperforms *MTS* and *COSEARCH* on all instances except tai100a. *PHGETS* gives better results than all these algorithms and it increases the iterations in the final phase to provide better results for the TAI instances. Thus, as seen in the execution times shown in Table 5.12, Sko100a runs in less time than Tai100a. Since Talbi and Bachelet did not report the execution times, it is not possible to compare the results with *TB-MTS* and *TB-COSEARCH*. Furthermore, the *CPTS* results were reported for a different platform. However, we restricted our experiments with 10 processors as in *CPTS* and solved each instance 10 times to minimize the measurement errors. Talbi and Bachelet used 150 processors, whereas we restricted ourselves to the *CPTS* because it is the most successful parallel algorithm reported for the *QAP* in the literature.

5.4 Experimental Setup and Test Results for the Data Allocation Problem

We tested the proposed algorithms through a number of experiments. In each test, one parameter varies while fixing the others. The algorithms are tested by the same test data. Experiments are performed using a 2.21 GHz AMD Athlon (TM) 64x2 dual processor with 2GB RAM and MS Windows 7 (TM) operating system. Each processing node has 102 buffers (each buffer is one page), and page size is 10240 bytes, disk I/O time is 10ms (per page), available memory is sufficient to perform all join operations in main memory and each table is loaded into memory only once. The implementation language is C++. Test data is generated according to the experimental environment of Adl [1]. The only difference is that we choose the unit costs in range [0,1]. Our test data generator gets number of fragments *m*, number of sites *n* and other parameters as input and creates a random DAP instance. We choose the fragment size randomly from the range $[\frac{c}{10}, 20 \times \frac{c}{10}]$, where c is a number between 10 and 1000. We choose the site capacities in $[1, 2 \times \frac{m}{n}$ - 1]. The sum of the site capacities should be equal to total fragment size *m*, where n is the total number of sites. We assumed that the number of sites *n* is equal to number of fragments *m*. Each fragment size is chosen randomly. We selected the unit transmission costs as a random number in range [0,1]. We generate a random probability request per transaction(*RT*) for each transaction to be requested as a site. Transaction fragment dependency is also represented with a probability access per fragment(*AF*). The site fragment frequency matrix *FREQ* is determined as the multiplication of probability *RT* and a random frequency of range [1, 1000]. Transaction fragment dependency matrix is generated as the multiplication of *AF* and a uniformly distributed random value in [0, f_j] as f_j being the *j*'th fragment. Finally the site fragment dependency matrix *STFR* is equal to *FREQ* $\times$ *TRFR*. We define the inter-fragment dependency matrix *FRDEP* as multiplication of the matrices $QFR_{l\times m\times m}$ and $Q_{l\times m\times m}$ where *QFR* takes into account the execution frequencies of the transactions and *Q* represents the indirect transaction fragment dependency.

We performed several tests over genetic algorithm to set the appropriate parameters. We varied the population size and number of generations to find the optimal running time settings. We performed tests on three DAP instances of sizes 20, 50 and 100.

Besides, three configuration settings are selected as GA1, GA2 and GA3 after the experiments shown in Tables 5.13 and 5.14. GA1 uses population size 1,000 and number of generations 200. GA2 uses population size 1,250 and number of generations 200. Finally, GA3 uses population size 1,250 and number of generations 150. These are the best performing parameters in our experiments. We used the

Table5.13: Genetic Algorithm Performance on DAP-20 and DAP-50 instances.

Size	Population	Generations	Cost	Time(sec)	Instance	Population	Generations	Cost	Time(sec)
20	250	50	2,655,762	4.214	50	250	50	55,129,698	17.581
20	250	100	2,674,170	7.777	50	250	100	55,330,358	33.727
20	250	150	2,715,429	14.786	50	250	150	55,235,938	55.411
20	250	200	2,673,403	15.171	50	250	200	55,116,918	46.914
20	250	250	2,666,135	18.124	50	250	250	55,229,595	66.79
20	500	50	2,640,902	15.451	50	500	50	54946089	28.204
20	500	100	2,660,429	19.617	50	500	100	54932897	79
20	500	150	2,651,025	35.112	50	500	150	54833906	86.822
20	500	200	2,649,761	53.737	50	500	200	54945268	123.973
20	500	250	2,649,985	76.297	50	500	250	54908336	135.938
20	750	50	2,653,905	18.796	50	750	50	55121281	41.009
20	750	100	2,636,911	49.153	50	750	100	54889295	76.333
20	750	150	2,646,546	63.947	50	750	150	54681993	143.941
20	750	200	2,631,491	117.159	50	750	200	54650527	183.265
20	750	250	2,648,080	173.525	50	750	250	54667462	224.648
20	1000	50	2,638,806	27.666	50	1000	50	55002209	60.278
20	1000	100	2,630,268	62.974	50	1000	100	54629883	154.437
20	1000	150	2,637,986	84.13	50	1000	150	54627740	207.555
20	1000	200	2,629,396	123.789	50	1000	200	54764319	471.978
20	1000	250	2,638,621	253.896	50	1000	250	54995658	644.301
20	1250	50	2,639,087	54.687	50	1250	50	54953858	90.265
20	1250	100	2,640,549	83.482	50	1250	100	54783223	310.599
20	1250	150	2,640,518	167.222	50	1250	150	54689684	359.574
20	1250	200	2,648,000	148.553	50	1250	200	54651822	499.853
20	1250	250	2,629,725	417.682	50	1250	250	54676825	460.665

Table5.14: Genetic Algorithm Performance on DAP-100 instance.

Size	Population	Generations	Cost	Time(sec)
100	250	50	440,608,146	48.757
100	250	100	438,827,255	117.298
100	250	150	439,988,411	145.033
100	250	200	441,213,326	181.507
100	250	250	438,959,166	200.907
100	500	50	438,380,273	91.566
100	500	100	437,361,781	174.24
100	500	150	436,810,249	333.716
100	500	200	438,871,504	441.14
100	500	250	437,437,210	493.093
100	750	50	439,058,054	171.855
100	750	100	436,156,120	317.021
100	750	150	436,641,217	618.125
100	750	200	435,207,238	943.319
100	750	250	435,592,737	1015.88
100	1000	50	438,421,608	216.33
100	1000	100	436,734,155	487.567
100	1000	150	436,150,353	568.726
100	1000	200	436,194,801	1236.301
100	1000	250	435,735,532	1463.757
100	1250	50	438,726,232	334.48
100	1250	100	436,604,972	688.081
100	1250	150	434,451,806	808.024
100	1250	200	435,041,686	1079.677
100	1250	250	435,406,898	1455.124

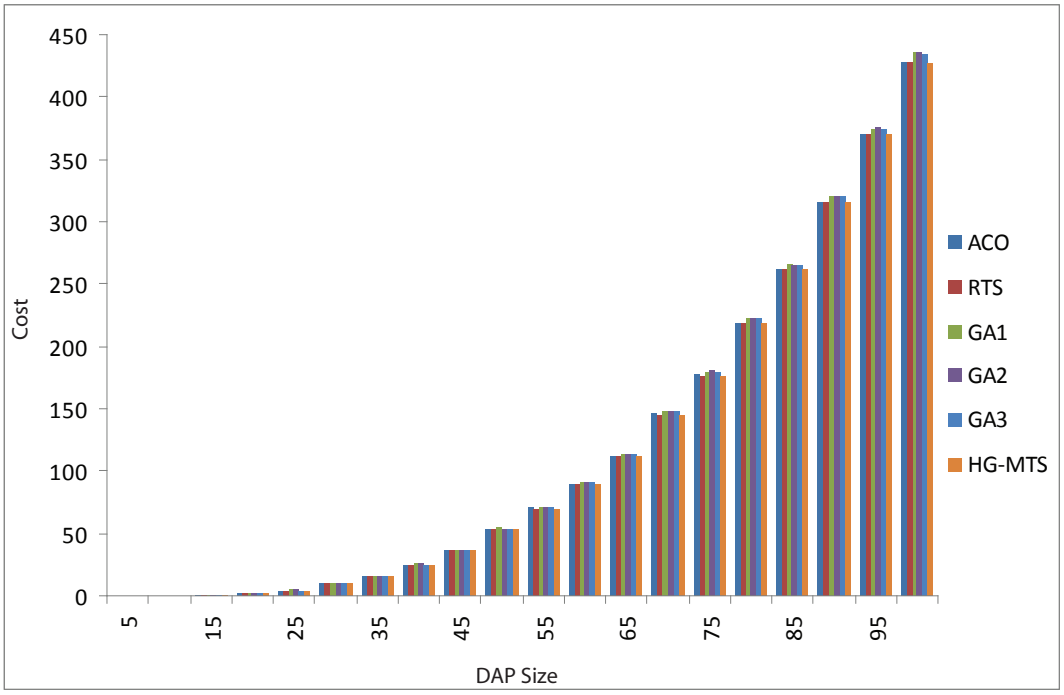

Figure 5.8: Quality comparison of data allocations discovered by all algorithms

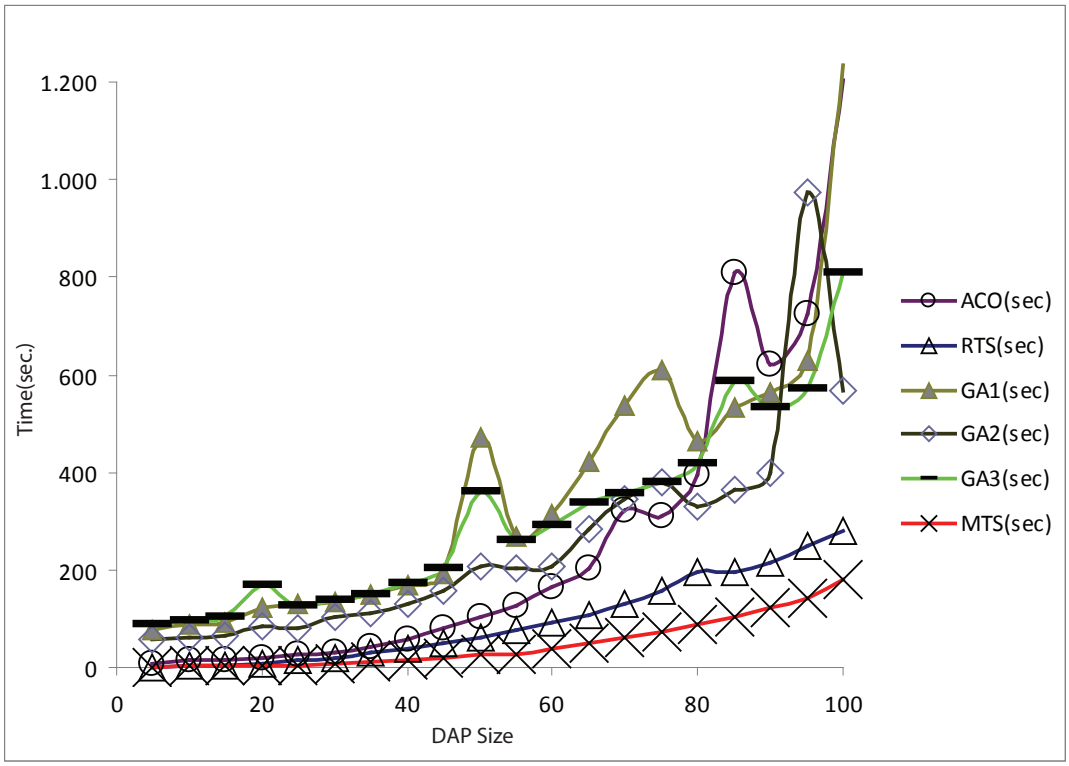

Figure 5.9: Time vs. Instance Size Comparisons of the algorithms.

Table5.15: Comparison of Algorithms on DAP Instances (Cost value is column$\times 10^6$).

DAP Size	Cost ACO	Cost RTS	Cost GA1	Cost GA2	Cost GA3	Cost HG-MTS	DAP Size	Time ACO(sec)	Time RTS(sec)	Time GA1(sec)	Time GA2(sec)	Time GA3(sec)	Time MTS(sec)
5	0.04	0.04	0.04	0.04	0.04	0.04	5	9.26	0.83	76.27	56.11	88.11	1.44
10	0.31	0.31	0.32	0.31	0.31	0.31	10	14.52	2.73	87.80	60.37	94.91	2.45
15	0.98	0.98	0.99	0.98	0.98	0.98	15	13.74	5.66	90.76	66.22	104.13	2.65
20	2.61	2.61	2.63	2.64	2.64	2.61	20	17.91	8.89	123.79	84.13	167.22	4.17
25	5.19	5.19	5.25	5.26	5.24	5.19	25	25.86	14.52	131.98	81.96	125.30	5.21
30	10.27	10.27	10.39	10.42	10.41	10.27	30	31.17	20.89	132.46	104.64	137.02	7.38
35	16.39	16.39	16.64	16.61	16.66	16.39	35	43.31	29.06	150.06	111.87	151.02	10.73
40	25.91	25.90	26.28	26.33	26.21	25.92	40	56.59	37.05	166.80	128.75	173.21	15.60
45	37.28	37.26	37.73	37.80	37.82	37.27	45	80.92	48.67	191.93	159.10	202.10	20.80
50	53.93	53.89	54.76	54.63	54.69	53.88	50	105.33	62.74	471.98	207.56	359.57	26.80
55	71.30	71.19	72.72	72.40	72.13	71.21	55	126.00	76.07	268.31	201.43	261.71	27.22
60	90.35	90.16	91.76	91.49	91.56	90.20	60	166.55	91.79	315.31	208.37	290.46	39.56
65	112.31	112.13	113.59	113.75	113.84	112.08	65	204.35	109.20	421.93	284.08	336.01	48.92
70	146.41	146.19	148.48	148.80	148.18	146.15	70	320.62	131.54	536.15	344.20	358.03	63.13
75	177.90	177.70	180.04	180.75	180.63	177.65	75	309.51	155.31	609.77	379.07	380.81	73.41
80	219.40	219.26	223.10	222.80	222.96	219.18	80	396.18	193.63	464.17	331.17	416.18	87.84
85	262.24	261.88	267.04	266.15	266.19	261.99	85	807.43	195.80	532.05	364.71	586.21	102.79
90	316.11	315.86	320.88	320.93	320.58	315.86	90	621.55	215.58	563.15	400.37	531.13	123.19
95	370.14	369.92	375.49	375.85	375.29	369.91	95	725.93	250.72	629.55	974.24	569.92	143.16
100	428.40	428.28	436.19	436.15	434.45	427.98	100	1.203.99	278.63	1.236.30	568.73	808.82	179.07

Fast Ant System [142] with parameter R=5 for managing traces and number of iterations as 20,000. In addition, we used the aspiration parameters 200,000 for the maximum number of failure, $\frac{9\times n}{10}$ for the lower limit of the tabu list and $\frac{11\times n}{10}$ for the upper limit of the tabu list where n is the instance size. We used a population size 250 and number of generations 50 for the initial phase of Hybrid Genetic Multi Start Tabu Search. The diversification phase uses 1,000 for the maximum number of failure, $\frac{n\times n}{10}$ for the lower limit of the tabu list and $\frac{11\times n}{10}$ for the upper limit of the tabu list. After completing the experiments on instances ranging from size 5 to 100, it is concluded that HG-MTS outperforms the other algorithms in terms of time and cost measurements. Only RTS can perform better results than HG-MTS on a few instances. However, HG-MTS executes faster than all other methods on all instances as seen in Table 5.15, Figure 5.8 and Figure 5.9.

5.5 Experiments with *B-IPGA*

This section discusses the test results on the sample SC model. The cost model proposed for the SC with *B-IPGA* aims to reduce the total cost accumulated by participants. Experiments show the behavior of the SC under varying market conditions. Purchase orders are amplified through downstream participants. A one to one match among the orders of neighbor members exists. In the original beer game, the players are customer, retailer, distributor, warehouse and factory. Each member orders at most 30 quantities [115]. As a result, each gene is represented with 5 bits when exploiting a genetic algorithm solution to reduce the bullwhip effect. In our model, we used a binary encoding consisting of 5 SC members and a gene structure of 5 bits. 30 processors are used with a population size of 100 and 50 generations. The truncation limit is 0.1 and mutation operator is performed with a probability of %1. The experiments are conducted for 100 weeks. Figure 5.10 and Figure 5.12 show the costs for 5 week periods over 100 weeks. As it is seen in Figure 5.12, the bullwhip effect reduces drastically when *B-IPGA* methodology is used. The total accumulated costs are also reduced as seen in Figure 5.12. However, customer demand is not fully satisfied and downstream members try to protect themselves in this model.

Our experiments are performed on a cluster computer. A cluster is a group of loosely coupled computers working together closely, so that in many aspects they can be thought as a single computer [12].

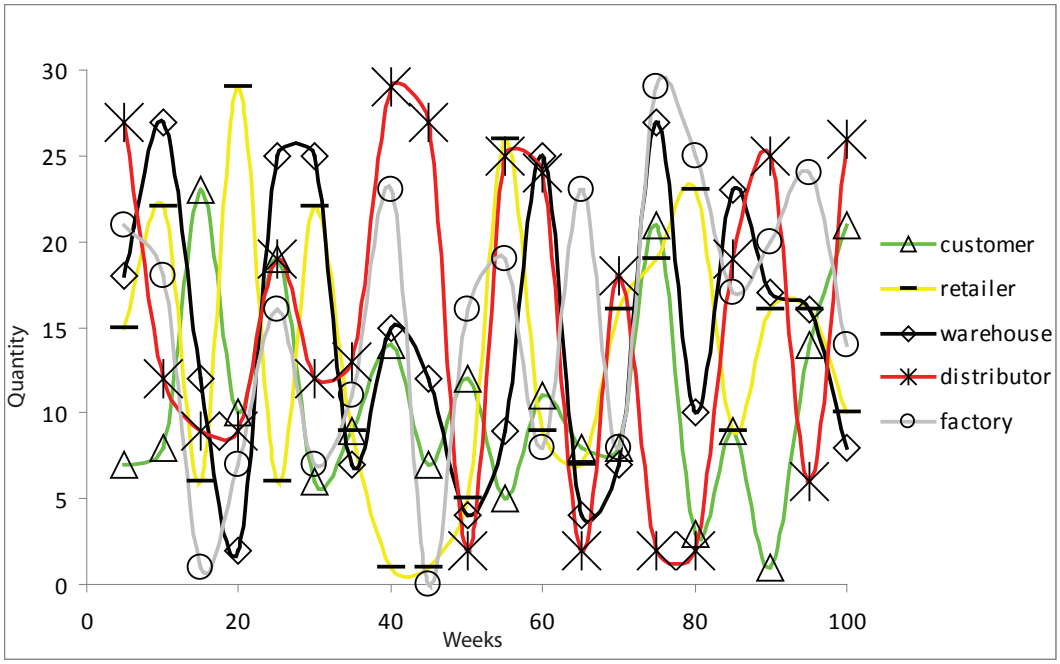

Figure 5.10: Normal Behavior of the Supply Chain

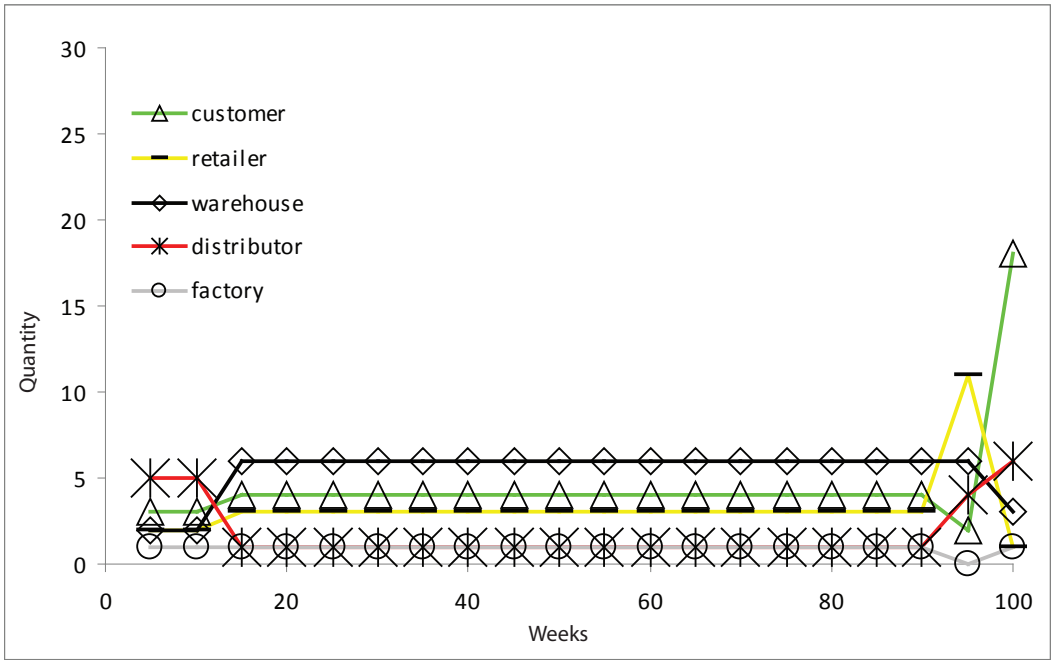

Figure 5.11: *B-IPGA* Behavior of the Supply Chain

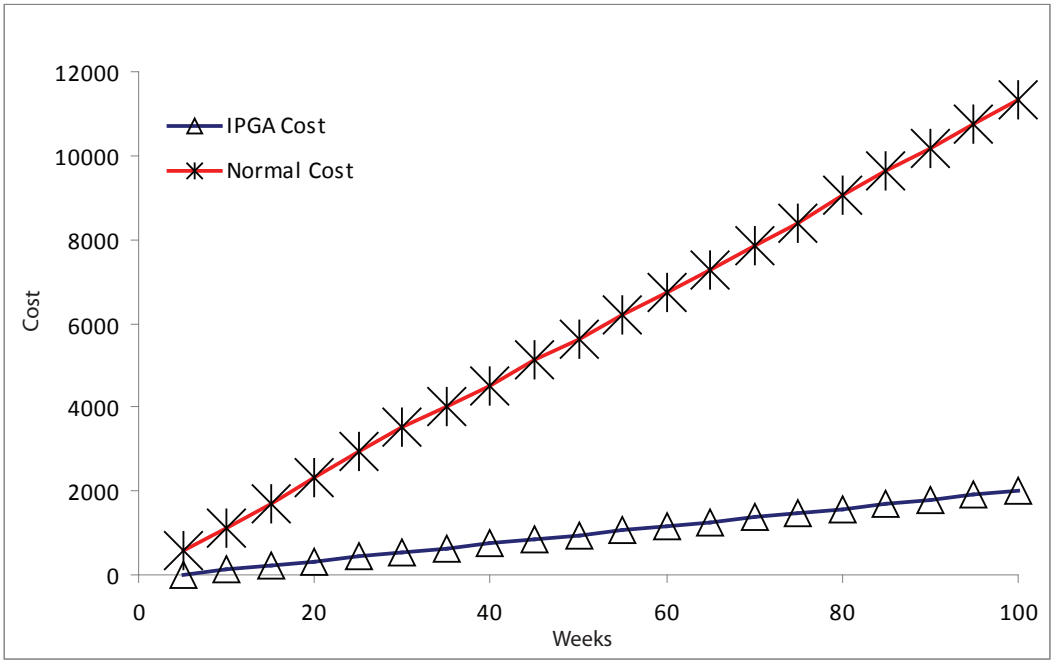

Figure 5.12: Normal Cost versus *B-IPGA* Cost

Clusters are typically used for ”high availability” to achieve greater reliability or high performance computing to provide greater computational power than a single computer can provide. The key components of a cluster include, multiple standalone computers, an operating system, a high performance interconnect, communication software, middleware, and applications. Complete configuration of our HPC system is; 46 nodes, each with 2 CPUs giving 92 CPUs. Each CPU has 4 cores giving a total number of 368 cores. Each node has 16GB of RAM giving 736GB of total memory, and 146GBs of disk storage per node, giving 6.5TB total disk space. In addition, there are 2 high performance disk units each with 3 TB giving 6TB common storage area. For providing high-bandwidth communication among HPC nodes, two 24 port Gigabit Ethernet Switches, and one 24 port infiniband switch are used. Software installed on HPC is; Scientific Linux v4.5 64-bit operating system, Lustre v1.6.4.2 parallel file system, Torque v2.1.9 resource manager, Maui v3.2.6 job scheduler, and OpenMPI v1.2.4.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we apply integer linear programming on Multiple Query Optimization (*MQO*) and Distributed Database Design problems. Further, we propose new parallel and heuristic methods for the Data Allocation Problem (*DAP*) and Quadratic Assignment Problem (*QAP*). We propose a complete model including replication, query frequencies, update costs etc. The earlier work in the literature lack most of these features because the problem is quite complex. We present two efficient algorithms, *QAP-IPGA* and *PHGETS* for solving *QAP* problem.

MQO is another NP-hard problem for solutions using A*, dynamic programming, and Genetic Algorithms exist, and A* based techniques are known to have best performance for this problem. We propose an ILP based formulation to solve large MQO instances efficiently. The experimental results show that integer linear programming is very effective in solving large *MQO* instances, and with increasing number of queries A*'s performance degrades quickly. Our ILP formulation proves to be very robust and it solves large *MQO* instances efficiently with a simple and straight forward model obtained easily from the traditional *MQO* formulation using tasks, plans and queries as linear constraints and a simple objective function containing only one term for each task in the *MQO* problem. We believe that further improvements can be obtained by solving the *MQO* with general purpose linear programming software that can be executed on parallel hardware.

Our thesis shows that the proposed genetic algorithm based QAP-IPGA and PHGETS algorithms can perform very well even on *QAP* a very difficult optimization problem. We have also concluded that traditional exhaustive search techniques can provide linear speedup when they are parallelized; however, this does not provide much improvement in the size of optimally solvable problem instances since the complexity of intractable problems such as Quadratic Assignment Problem is exponential. On the other hand, genetic algorithms with their well known ability to provide satisfactory results for intractable problems can increase solution qualities when they are given adequate computational power with parallel machines. Genetic algorithms do not guarantee finding the optimal solutions; however, in shorter periods of time they will provide more satisfying results than exhaustive algorithms. Our results show that Island Parallel Genetic Algorithms are promising search tools.

In this thesis, we present a new robust three stage Parallel Hybrid Genetic Tabu Search Algorithm (*PHGETS*), for the solution of large *QAP* instances. Our experiments show that Genetic Algorithms can provide good (optimal or near-optimal for small and medium size instances) solutions for the *QAP*. We implemented and experimentally evaluated *PHGETS* on a parallel machine and showed that it can find good solutions in minutes.

The *QAP* formulation for the distributed database design problem was also evaluated on our experimental database environment and it was experimentally shown to give solutions that are better than

the existing techniques in the literature. We used *RTS*, Genetic Algorithm and *FANT* algorithms to solve the Data Allocation Problem as a *QAP*. We also present a new data allocation algorithm called *HG-MTS* which is based on the genetic algorithm and Multi Start Robust Tabu Search. In our experiments, the execution times and optimality of the other well known *QAP* algorithms are compared. *HG-MTS* outperforms Genetic Algorithm, *FANT* and *RTS* in terms of solution quality and execution times in almost all cases for the Data Allocation Problem. Currently, the proposed model considers only one fragment at a site. This limitation will be relaxed in the future. We also plan to extend the proposed algorithm to handle fragment replication on a site. Also, load balancing and parallel execution of tasks to minimize the response time can be considered in the future work. Including network installation/operations costs and average response time as constraints will also improve the power of our model.

REFERENCES

[1] R.K. Adl and S.M.T.R. Rankoohi. A new ant colony optimization based algorithm for data allocation problem in distributed databases. *Knowledge and Information Systems*, 20(3):349-372, 2009.

[2] I. Ahmad, K. Karlapalem. Evolutionary Algorithms for Allocating Data in Distributed Database Systems. *Distributed and Parallel Databases*, 11(1):5-32, 2002.

[3] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27(10):917-934, 2000.

[4] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Diaz, I. Dorta, J. Gabarro, C. Leon, G. Luque, J. Petit, C. Rodriguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project. *Parallel Computing*, 32 (5-6):415-440, 2006.

[5] G.S. Almasi. *Highly parallel computing.* 2nd Edition, Benjamin-Cummings publishers, 1994.

[6] Ş. Alpay and N. Yuzugullu. Dynamic job shop scheduling for missed due date performance. *International Journal of Production Research*. 47 (15):4047-4062, 2009.

[7] Ş. Alpay. GRASP with path relinking for a multiple objective sequencing problem for a mixed-model assembly line. *International Journal of Production Research*, 47(21):6001-6017, 2009.

[8] K. Anstreicher, N. Brixius, J. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563-588, 2002.

[9] K. Anstreicher and N. W. Brixius. Solving quadratic assignment problems using convex quadratic programming relaxations. *Optimization Methods and Software*, 16(1-4):49-68, 2001.

[10] M. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preu, M. Schoenauer. A framework for distributed evolutionary algorithms. *LNCS* 24(39):665-675, 2002.

[11] M. M. Astrahan, W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. A&Jones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, AND V. Watson. System R: A relational approach to database management. *ACM Transactions on Database Systems*, 1(2):97-137, 1976.

[12] A. Bader and R. Pennington. Cluster computing: applications. *The International Journal of High Performance Computing*, 15(2):181-185, 2001.

[13] S.M. Bartolomei-Suarez and P.J. Egbelu. Quadratic assignment problem QAP with adaptable material handling devices. *International Journal of Production Research*, 38(4): 855-873, 2000.

[14] R. Battiti, G. Tecchiolli. Parallel biased search for combinatorial optimization: genetic algorithms and tabu. *Microprocessors and Microsystems*, 16(7):351-367, 1992.

[15] R. Battiti and G. Tecchiolli. The reactive tabu search, *ORSA Journal on Computing* , 6(2):126-140, 1994.

[16] M. A. Bayir, I. H. Toroslu, A. Cosar. Genetic Algorithm for the Multiple-Query Optimization Problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(1): 147-153, 2007.

[17] B. Beamon and V. Chen. Performance analysis of conjoined supply chains. *International Journal of Production Research*, 39(14): 3195-3218, 2001.

[18] D. A. Bell. Difficult data placement problems. *Computer Journal*, 27(4):315-320, 1984.

[19] R. E. Burkard and E. Cela. Linear assignment problems and extensions. In Pardalos, P. and Du, D.-Z. (eds), Handbook of Combinatorial Optimization, *Kluwer Academic Publishers*, Supplement vol. A, pp. 75-149, 1999.

[20] R.E. Burkard, S.E. Karisch, F. Rendl. QAPLIB: a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391-403, 1997.

[21] S. Cahon, N. Melab, and E. -G. Talbi. ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357-380, 2004.

[22] J. P. Cohoon and W. D. Paris, "Genetic placement," *IEEE Transactions on Computer-Aided Design*, 6(6):956-964, 1987.

[23] C. Calzon-Bousono, "The hopfield neural network applied to the quadratic assignment problem," *Neural Computing and Applications*, 3(2):64-72, 1995.

[24] C. Carlsson and R. Fuller. Reducing the bullwhip effect by means of intelligent, soft computing methods. Hawaii International Conference on System Sciences, 3:3027, 2001.

[25] E. Cantu-Paz. *Efficient and accurate parallel genetic algorithms*. Kluwer Academic Publishers, 2000.

[26] P. Carraresi and F. Malucelli. A new lower bound for the quadratic assignment problem. *Operations Research*, 40(1):22-27, 1992.

[27] S. Ceri, G. Pelagatti. *Distributed databases principles and systems*. McGraw-Hill, 1984.

[28] S. Ceri, S. B. Navathe, G. Wiederhold. Distribution Design of Logical Database Schemas. I*EEE Transactions on Software Engineering*, 9(4): 487-504, 1983.

[29] F.T.S. Chan and S.H. Chung. A multi-criterion genetic algorithm for order distribution in a demand driven supply chain. *International Journal of Computer Integrated Manufacturing*, 17 (4):339-351, 2004.

[30] J. Chakrapani and J. Skorin-Kapov. A connectionist approach to the quadratic assignment problem. *Computers & Operations Reasearch*, 19(3-4):287-295, 1992.

[31] U.S. Chakravarthy, A. Rosenthal. Anatomy of a modular multiple query optimizer, in: Proceedings of the VLDB, pp. 230-239, 1988.

[32] S. Chakravarthy. Divide and conquer: A basis for augmenting a conventional query optimizer with multiple query-processing capabilities, in Proc. 7th Int. Conf. Data Eng., Kobe, Japan, Apr. 8(12):482-490, 1991.

[33] U.S. Chakravarthy and J. Minker. Multiple query processing in deductive database using query graphs. In Proc. of the VLDB Conf., pp. 384-391, 1986.

[34] F. Chen, Z. Drezner, K. R. Jennifer, and D. Simchi-Levi. Quantifying the Bullwhip Effect in a Simple Supply Chain: The Impact of Forecasting, Lead-Times and Information. Management Science, 46(3):436-443, 2000.

[35] D. Chen, R. G. Batson, Y. Dang. *Applied Integer Programming: Modeling and Solution*. John Wiley & Sons, 2010.

[36] F. Chen, Z. Drezner, K. R. Jennifer, and D. Simchi-Levi. The Bullwhip Effect: Managerial Insights on the Impact of Forecasting and Information on Variability in a Supply Chain. In Tayur Sridhar (Editor), Quantitative Models for Supply Chain Management. Kluwer Academic Publishers, 1998.

[37] S. Chopra and P. Meindl. *Supply Chain Management: Strategy, Planning and Operation.* 2nd Edition Prentice Hall New Jersey, 2004.

[38] V. Ciriani, N. Pisanti and A. Bernasconi. Room allocation: a polynomial subcase of the quadratic assignment problem. *Discrete Applied Mathematics*, 144(3):263-269, 2004.

[39] J. Clausen and M. Perregaard. Solving large quadratic assignment problems in parallel. *Computational Optimization and Applications*, 8(2):111-127, 1997.

[40] D.T. Connolly. An improved annealing scheme for the QAP. *Europan Journal of Operational Research*, 46(1):93-100, 1990.

[41] A.L. Corcoran, J. Hale. A genetic algorithm for fragment allocation in a distributed database system, In: SAC '94: proceedings of the 1994 ACM symposium on applied computing, Phoenix, pp. 247-250, 1994.

[42] J-F. Cordeau, M. Gaudioso, G. Laporte, and L. Moccia. The service allocation problem at the Gioia Tauro Maritime Terminal. *European Journal of Operational Research*, 176(2):1167-1184, 2007.

[43] D.W. Cornell and P.S. Yu. Site assignment for relations and join operations in the distributed transaction processing environment, in Proceedings of the IEEE International Conference on Data Engineering, 1988.

[44] D. W. Cornell, P. S. Yu: An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases. *IEEE Trans. Software Eng.* 16(2):248-258, 1990.

[45] A. Cosar, J. Srivastava, S. Shekhar. On the multiple pattern multiple object (MPMO) match problem, in: International Conference on Management of Data, India, 1991.

[46] A. Cosar. Design and experimental evaluation of a multiple query optimizer, Ph.D. dissertation, 1996.

[47] A. Cosar, E. P. Lim, and J. Srivastava, Multiple query optimization with depth-first branch-and-bound and dynamic query ordering, in Proc. CIKM 93, pp. 433-438, 1993.

[48] T.G. Crainic and M. Toulouse. *Parallel strategies for metaheuristics*. Handbook of Metaheuristics. Kluwer Academic Publishers, 2003.

[49] N.N. Dalvi, S.K. Sanghai, P. Roy, S. Sudarshan. Pipelining in multi-query optimization, in: Proceedings of the Twentieth PODS, Santa Barbara, CA, pp. 59-70, 2001.

[50] D. De La Fuente and J. Lozano. Application of distributed intelligence to reduce the bullwhip effect. *International Journal of Production Research*, 45(8):1815-1833, 2007.

[51] E. W. Dijkstra. A Note on Two Problems in Connection with Graphs. Numeriche Mathematik, 1:269-271, 1959.

[52] E.W. Dijkstra. *A discipline of programming*. Prentice-Hall, 1976.

[53] S. Disney, M. Naim, and D. Towill. Genetic algorithm optimisation of a class of inventory control systems. International Journal of Production Economics, 68(3):259-278, 2000.

[54] T. Dokeroglu, U. Tosun, A. Cosar. Parallel Optimization with Mutation Operator for the Quadratic Assignment Problem. *In Proceedings of WIVACE 2012, Italian Workshop on Artificial Life and Evolutionary Computation,* Parma/Italy, 2012.

[55] T. Dokeroglu, A. Cosar. Dynamic Programming with Ant Colony Optimization Metaheuristic for The Optimization of Distributed Database Queries," in Proc. of the 26th Int. Sym. On Computer and Information Sciences (ISCIS), London, UK, September, 2012.

[56] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions Systems, Man and Cybernetics - Part B*, 26(1):2-9, 1996.

[57] Z. Drezner. A new genetic algorithm for the quadratic assignment problem. *Informs Journal on Computing*, 15(3):320-330, 2003.

[58] Z. Drezner. The extended concentric tabu for the quadratic assignment problem. *European Journal of Operational Research*, 160(2):416-422, 2005.

[59] A.E. Eiben and J. E. Smith. *Introduction to evolutionary computing*, Springer, 2003.

[60] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 1999.

[61] E. Felten, S. Karlin, and S.W. Otto. The travelling salesman problem on a hypercubic, MIMD computer. *Proceedings of the International Conference on Parallel Processing*, pp. 6-10, 1985.

[62] N. Fescioglu-Unver and M.M. Kokar. Tabu search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering*, 60(2):310-319, 2011.

[63] C.N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 51(3):243-267, 1994.

[64] S. Finkelstein. Common expression analysis in database applications. In Proc. of the ACMSIGMOD Int '1 Conf. on the Management of Data, pp. 235-245, 1982.

[65] J. Forrester. Industrial dynamics: A major breakthrough for decision makers. Harvard Business Review, 36(4):3766, 1958.

[66] O. Frieder, H.T. Siegelmann. Multiprocessor document allocation: A genetic algorithm approach. IEEE Transactions on Knowledge and Data Engineering 9(4):640-642, 1997.

[67] L.M. Gambardella, E.D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167-176, 1999.

[68] R. Gaonkar and N. Viswanadham. Collaboration and Information sharing in global contract manufacturing networks. *IEEE/ASME Transactions on Mechatronics*, 6(4):366-376, 2001.

[69] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-Completeness*. Freeman, 1979.

[70] B.Gavish and H. Pirkul. Computer and database location in distributed computer systems. *IEEE Transactions on Computers*, C-35(7):583-590, 1986.

[71] M. Gen and R. Cheng. Genetic algorithms and engineering design. John Wiley & Sons, 1997.

[72] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society of Industrial and Applied Mathematics*, 10(2):305-313, 1962.

[73] F. Glover. A template for scatter search and path relinking. In: J.-K. Hao, et al. (Eds.), Artificial Evolution, In: LNCS 1363. *Springer-Verlag*, pp. 3-54, 1998.

[74] D. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[75] J. Grant and J. Minker. On optimizing the evaluation of a set of expressions. Int'1 J. Comput. Inform. Sci., 11, March 1982.

[76] X. Gu, W. Lin. Practically realizable efficient data allocation and replication strategies for distributed databases with buffer constraint. *IEEE Trans Parallel and Distributed Systems*, 17(9):1001-1013, 2006.

[77] A. Haghani and M. Chen. Optimizing gate assignments at airport terminals. *Transportation Research Part A: Policy and Practice*, 32(6):437-454, 1998.

[78] P. Hahn and J. Krarup, A hospital facility layout problem finally solved, *Journal of Intelligent Manufacturing*, 12(5-6):487-496, 2001.

[79] J.H. Holland. *Adaptation in natural and artificial systems*. The MIT Press, 1975.

[80] K. L. Hoffman, M. Padberg. Solving Airline Crew Scheduling Problems by Branch-and-Cut. *Management Science*, 39(1):657-682, 1993.

[81] T. James, C. Rego, and F. Glover. Multi-start tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(3):579-596, 2009.

[82] T. James, C. Rego, and F. Glover. Sequential and parallel pathrelinking algorithms for the quadratic assignment problem. *IEEE Intelligent Systems*, 20(4):58-65, 2005.

[83] T. James, C. Rego, and F. Glover. A cooperative parallel tabu search algorithm for the QAP. *European Journal of Operational Research*, 195(3):810-826, 2009.

[84] T. James, C. Rego, and F. Glover. Multi-start tabu search and diversification strategies for the quadratic assignment problem. *Working Paper*, Virginia Tech, 2006.

[85] P. Kalnis and D. Papadias. Multi-query optimization for on-line analytical processing. *Information Systems*, 28(5):457-473, 2003.

[86] J. P. Kelly, M. Laguna, and F. Glover. A study of diversification strategies for the quadratic assignment problem. *Computers and Operations Research*, 21(8):885-893, 1994.

[87] S. Kimbrough, D.J. Wu and F. Zhong. Computers play the beer game: can artificial agents manage supply chains? *Decision Support Systems*, 33(3):323-333, 2002.

[88] T.C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):53-76, 1957.

[89] V. Kumar. *Introduction to parallel computing.* Addison-Wesley, 2002.

[90] M. Laguna, R. Marti, and V. Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operations Research*, 26(12):1217-1230, 1999.

[91] L.J. Laning, M.S. Leonard. File allocation in a distributed computer communication network. *IEEE Transactions on Computers*, 32(3):232-244, 1983.

[92] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586-599, 1963.

[93] H. Lee, 2002. Unleashing the power of intelligence. *ECR Journal*, 2(1):61-73.

[94] H. L. Lee, V. Padmanabhan and S. Whang. Information Distortion in a Supply Chain: The Bullwhip Effect. *Management Science*, 43(4):546-558, 1997.

[95] Z. Lee, S. Su, C. Lee. A heuristic genetic algorithm for solving resource allocation problems," *Knowledge and Information Systems*, 5(4):503-511, 2003.

[96] D. S. Levi and P. Kaminsky. *Designing and Managing the Supply Chain: Concepts, Strategies and Case Studies*. Mc Graw-Hill Education, 2002.

[97] Y. Li, P. M. Pardalos, and M. G. C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem," Quadratic assignment and related problems. P. M. Pardalos and H. Wolkowicz, eds., *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 16(1):237-261, 1994.

[98] M. H. Lim, Y. Yuan and S. Omatu. Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. *Computational Optimization and Applications*, 15(3):249-268, 2000.

[99] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23(4):658-670, 2007.

[100] X. Lin, M. E. Orlowska. An Integer Linear Programming Approach to Data Allocation with the Minimum Total Communication Cost in Distributed Database Systems. *Information Sciences*, 85(1-3): 1-10, 1995.

[101] J. D. C. Little, K. G. Murty, D. W. Sweene, C. Karel. An Algorithm for the Traveling Salesman Problem. *Operations Research*, 11(3), 972-989, 1963.

[102] E. Loiola, N. Maia de Abreu, P. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657-690, 2007.

[103] G. Luque and E. Alba. *Parallel genetic algorithms, theory and applications*, Springer, 2011.

[104] A.S. Mamaghani, M. Mahi, M.R. Meybodi and M.M. Moghaddam. A Novel Evolutionary Algorithm for Solving Static Data Allocation Problem in Distributed Database Systems," Second International Conference on Network Applications, Protocols and Services. Reviews Booklet, Brussels, 2010.

[105] V.K. Manupati, S. Deo, N. Cheikhrouhou, and M.K. Tiwari. Optimal process plan selection in networked based manufacturing using game-theoretic approach. *International Journal of Production Research*, 50(18):5239-5258, 2012.

[106] A. Marzetta and A. Brungger. A dynamic-programming bound for the quadratic assignment problem. *Lecture Notes in Computer Science*, 1627:339-348, 1999.

[107] T. Mautor and C. Roucairol. A new exact algorithm for the solution of quadratic assignment problems. *Discrete Applied Mathematics*, 55(3):281-293, 1994.

[108] R. Metters. Quantifying the bullwhip effect in supply chains. *Journal of Operations Management*, 15(2):89-100, 1997.

[109] A. Misevicius. Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem. *Knowledge-Based Systems*, 16(5-6):261-268, 2003.

[110] A. Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30(1):95-111, 2005.

[111] www.mosek.com

[112] E. Mosekilde, E. Larsen and J. Sterman. Coping with Complexity: Deterministic Chaos in Human Decision Making Behaviour. Edited by: Casti, John L. and Karlqvist, Anders. Beyond Belief: Randomness, Prediction and Explanation in Science, 1991.

[113] A. Nagurney, *Supply Chain Network Economics: Dynamics of Prices, Flows, and Profits*. Edward Elgar Publishing, 2006.

[114] M. Nystrom, *Solving certain large instances of the quadratic assignment problem: Steinberg's examples*. Department of Computer Science, California Institute of Technology, 1999.

[115] T. O'Donnell et al. Minimising the bullwhip effect in a supply chain using genetic algorithms. *International Journal of Production Research*, 44(8):1523-1543, 2006.

[116] J. Olivella, A. Corominas and R. Pastor. Task assignment considering cross-training goals and due dates. *International Journal of Production Research*, 51(3):952-962, 2012.

[117] Y. Ouyang and X. Li. Vendor-Managed Inventory and Bullwhip Reduction in a Two-Level Supply Chain. *International Journal of Operations & Production Management*, 23(6):625-651, 2003.

[118] Y. Ouyang and X. Li. The bullwhip effect in supply chain networks. *European Journal of Operational Research*, 201(3):799-810, 2010.

[119] M. T. Ozsu, P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2011.

[120] M. W. Padberg, Ed. Combinatorial Optimization. Mathematical Programming Study, 1980.

[121] S. Papadomanolakis and A. Ailamaki. An integer linear programming approach to database design. In Workshop on Self-Managing Database Systems, 2007.

[122] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The Quadratic Assignment Problem: A Survey and Recent Developments. In Quadratic Assignment and Related Problems, P. M. Pardalos and H. Wolkowicz, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16:1-42, 1994.

[123] D. W. Pentico. Assignment problems: a golden anniversary survey. *European Journal of Operational Research*, 176(4):774-793, 2007.

[124] G.F. Pfister. *In search of clusters.* 2nd Edition, Prentice Hall, 1998.

[125] N.J. Radcliffe and P.D. Surry. The reproductive plan language RPL2: Motivation, architecture and applications. *Genetic Algorithms in Optimisation, Simulation and Modeling. IOS Press*, 1999.

[126] S. Ram and R.E. Marsten. A model for database allocation incorporating a concurrency control mechanism. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):389-395, 1991.

[127] D. F. Rossin, M.C. Springer and B.D. Klein. New complexity measures for the facility layout problem: an empirical study using traditional and neural network analysis. *Computers & Industrial Engineering*, 36(3):585-602, 1999.

[128] P. Roy, S. Seshadri, S. Sudarshan, S. Bhobe, Efficient and extensible algorithms for multi-query optimization. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 249-260, 2000.

[129] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Association for the Computing Machinery*, 23(3):555-565, 1976.

[130] T. Sellis. Multiple query optimization. *ACM Transactions on Database Systems*, 13(1):23-52, 1988.

[131] T. Sellis. Global query optimization. In Proc. of the ACM-SIGMOD Int '1 Conf. on the Management of Data, 1986.

[132] E. Sevinc and A. Cosar. An evolutionary genetic algorithm for optimization of distributed database queries. *The Computer Journal*, 54(5):717-725, 2011.

[133] K. Shim, T. Sellis, D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. Data and Knowledge Engineering, 12(2):197-222, 1994.

[134] S. P. Singh and R. K. Sharma. Two-level modified simulated annealing based approach for solving facility layout problem. *International Journal of Production Research*, 46(13):3563-3582, 2008.

[135] J. Skorin-Kapov. Extensions of a tabu search adaptation to the quadratic assignment problem. *Computers & Operations Research*, 21(8):855-865, 1994.

[136] L. Steinberg. The backboard wiring problem: a placement algorithm. *SIAM Review*, 3(1):37-50, 1961.

[137] J. Stender. *Parallel genetic algorithms: theory and applications*. IOS Press, 1993.

[138] P. Stock and G. Zülch. Reactive manufacturing control using the ant colony approach. *International Journal of Production Research*, 50(21):6150-6161, 2012.

[139] T. Stutzle and M. Dorigo. *ACO algorithms for the quadratic assignment problem. New Ideas for Optimization.* McGraw-Hill, 1999.

[140] E. Sucky. The bullwhip effect in supply chains an overestimated problem. *International Journal of Production Economics*, 118(1):311-322, 2009.

[141] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443-455, 1991.

[142] E. D. Taillard, L.-M. Gambardella, M. Gendreau, and J.Y. Potvin. Adaptive Memory Programming: A Unifed View of Meta-Heuristics, EURO XVI Conference Tutorial and Research, 1998.

[143] E-G. Talbi, Z. Hafidi, and J-M. Geib. Parallel adaptive tabu search for large optimization problems. *MIC*, pp. 137-142, 1997.

[144] E-G. Talbi and V. Bachelet. COSEARCH: A parallel cooperative metaheuristic. *Journal of Mathematical Modeling and Algorithms*, 5(1):5-22, 2006.

[145] D. M. Tate and A. E. Smith. A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, 22(1):73-83, 1995.

[146] L. Tseng and S. Liang. A hybrid metaheuristic for the quadratic assignment problem. *Computational Optimization and Applications*, 34(1):85-113, 2005.

[147] U. Tosun, T. Dokeroglu, and A. Cosar. A New Robust Island Parallel Genetic Algorithm for the Quadratic Assignment Problem. *International Journal of Production Research*, 2012.

[148] S. Tsutsui and N. Fujimoto. Solving quadratic assignment problems by genetic algorithms with GPU computation: a case study. *GECCO*, pp. 2523-2530, 2009.

[149] Z. Weihang, C. James and M. Alberto. SIMD tabu search for the quadratic assignment problem with graphics hardware acceleration. *International Journal of Production Research* 48(4):1035-1047, 2010.

[150] W. C. Wesley, I. T. Ieong. A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems. *IEEE Transactions on Software Engineering*, 19(8): 804-812 (1993)

[151] M. R. Wilhelm. Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, 19(1):107-119, 1987.

[152] E. Wong and K. Youssefi. Decomposition: A strategy for query processing. *ACM Transactions on Database Systems*, 1(3):223-241, 1976.

[153] G. Xiong and P. Helo. An application of cost-effective fuzzy inventory controller to counteract demand fluctuation caused by bullwhip effect. *International Journal of Production Research*, 44(24):5261-5277, 2006.

APPENDIX A

THE BEST SOLUTIONS FOUND BY *QAP-IPGA*

Tai30a
8-10-22-15-30-19-11-27-9-21-3-28-6-13-20-12-16-17-7-14-5-18-29-24-1-2-4-25-23-26
Tai35a
24-28-10-32-5-7-13-12-25-8-1-4-20-16-18-15-35-34-27-6-22-14-19-21-2-26-30-3-33-31-9-23-11-29-17
Ste36a
36-16-1-15-14-29-30-31-33-17-18-10-7-20-28-19-32-34-2-8-4-13-12-11-23-21-22-35-3-9-5-6-27-26-25-24
Lipa40a
7-6-14-27-19-37-21-36-16-1-40-24-30-2-5-28-22-34-20-35-32-26-29-3-4-11-12-10-8-38-9-17-31-18-33-15-25-23-39-13
Tai40a
22-38-16-31-26-35-32-1-12-28-6-34-4-30-25-13-2-5-37-10-8-40-21-29-23-20-9-19-17-18-24-39-7-15-11-33-14-3-27-36
Sko42
41-17-7-12-20-31-13-11-34-42-5-19-10-27-25-32-18-26-14-35-2-29-40-15-9-4-39-30-21-38-1-37-33-28-22-8-6-3-24-16-36-23
Sko49
46-28-20-18-14-13-47-40-11-25-29-41-42-36-49-48-43-26-35-7-1-12-2-6-34-10-9-15-3-31-38-33-45-37-24-22-23-27-30-5-32-19-44-39-8-16-21-17-4
Lipa50a
1-35-42-12-49-34-2-38-37-6-32-11-16-43-14-19-33-17-25-29-13-9-27-23-3-47-45-46-41-44-31-8-24-18-50-21-22-4-36-26-28-5-7-10-20-39-48-15-40-30
Wil50
26-25-40-3-31-38-44-5-35-22-9-13-50-32-41-30-16-10-14-1-12-2-15-24-23-49-45-21-29-48-27-18-11-47-39-37-6-4-28-46-20-7-8-19-43-36-33-34-42-17
Lipa60a
53-24-54-10-50-11-44-32-52-46-18-6-16-58-27-23-19-41-55-14-1-15-43-30-20-31-13-60-9-28-57-5-35-7-8-17-51-42-40-39-4-26-22-25-49-12-48-2-56-34-47-45-36-3-38- 59-29-33-21-37
Sko64
36-60-46-29-52-14-9-18-12-48-5-20-45-1-56-35-27-40-23-33-22-37-58-50-61-30-39-16-24-44-17-26-3-43-7-55-64-21-25-42-34-28-6-62-4-59-13-63-41-11-32-53-38-19-49-10-15-54-57-51-47-8-2-31
Lipa70a
52-1-68-55-50-33-66-42-13-43-70-23-62-36-11-26-64-41-56-63-32-17-12-67-30-45-2-8-31-4-22-51-39-53-16-28-40-24-58-21-7-34-5-38-10-49-20-61-3-65-60-19-57-15-37-25-48-46-59-54-69-6-14-47-35-9-18-44-29-27
Lipa80a

51-28-17-46-3-7-34-59-73-4-76-24-5-62-15-29-10-1-25-55-67-58-30-6-14-31-2-21-36-16-80-37-72-9-65-68-19-69-53-20-38-75-35-44-49-77-52-79-50-27-56-12-61-47-54-60-33-64-39-70-23-22-78-66-18-48-43-45-11-32-13-8-57-40-63-71-74-42-41-26

Lipa90a

15-67-22-73-45-37-80-56-28-71-66-49-7-41-25-8-29-62-74-33-72-87-34-42-13-4-11-9-88-52-53-78-27-21-14-59-19-39-2-90-38-5-77-24-85-63-57-10-65-32-58-40-76-70-18-48-1-30-46-3-64-16-43-51-60-89-55-12-47-31-23-6-44-36-26-81-84-75-79-54-20-35-82-86-61-17-69-50-68-83

Sko100a

15-6-28-88-86-8-66-90-87-58-33-77-43-22-60-59-69-47-100-70-1-25-49-32-19-68-40-51-38-45-11-13-94-16-52-98-67-72-50-21-83-54-2-34-96-42-26-46-41-10-37-95-14-31-85-81-61-64-48-23-71-74-75-89-30-56-73-82-36-7-39-79-63-57-4-84-12-99-53-76-65-55-5-80-9-24-62-92-17-29-78-91-18-97-27-93-20-44-35-3

Sko100c

63-1-66-20-95-45-79-13-85-87-65-21-9-10-97-40-41-43-55-78-28-7-68-48-15-25-92-3-59-12-8-99-77-98-88-33-84-56-49-18-67-57-58-93-30-35-16-19-22-71-90-86-81-50-2-17-83-42-54-38-89-23-51-11-91-100-14-34-36-52-72-82-44-75-61-29-74-80-26-70-27-39-53-73-31-5-96-47-6-46-62-94-4-69-24-32-60-76-37-64

Sko100e

38-86-67-73-80-81-1-46-63-41-16-66-44-24-8-36-77-18-32-65-59-71-48-20-7-68-78-29-88-100-45-51-10-50-75-28-87-72-33-53-52-85-99-27-89-56-19-11-91-79-17-37-60-94-55-97-2-64-84-69-30-34-47-90-82-14-92-25-70-57-31-26-58-83-12-4-76-96-95-21-22-3-93-23-49-42-54-43-98-39-74-6-9-5-13-40-35-62-15-61

Tho150

9-72-77-105-110-148-21-75-124-37-68-116-11-15-73-65-136-3-34-115-111-30-10-24-25-4-123-81-60-64-17-12-45-40-46-140-92-42-114-100-131-133-59-22-107-142-121-43-54-104-61-63-16-146-79-108-99-129-82-18-127-135-27-80-138-44-66-58-8-150-103-101-1-52-93-119-143-144-117-87-56-132-90-122-49-55-134-39-2-128-7-83-94-88-141-145-47-84-78-19-48-76-71-120-96-38-86-14-70-50-32-28-89-130-95-35-126-51-53-109-85-106-113-57-97-13-67-118-41-91-139-112-147-69-20-102-149-29-125-5-6-31-98-26-74-36-33-23-62-137

Tai150b

4-56-30-113-82-109-47-103-32-149-115-72-61-125-99-83-51-147-119-65-123-135-38-64-80-20-6-27-25-146-144-60-58-29-105-31-132-19-110-104-93-69-142-44-22-112-108-85-107-49-128-46-124-35-136-13-26-79-138-102-41-23-117-89-50-2-11-87-120-9-131-57-7-12-118-95-141-48-75-81-134-66-45-63-127-86-97-84-77-96-122-90-73-37-54-130-67-100-76-55-116-126-88-68-78-129-114-94-28-143-121-34-14-106-8-98-150-139-36-33-43-10-5-91-62-1-52-92-53-39-40-16-70-21-101-3-18-15-133-148-111-17-145-74-71-42-24-140-59-137

APPENDIX B

THE BEST SOLUTIONS FOUND BY *PHGETS*.

Nug30 ∗
20-14-3-27-18-15-4-30-16-11-22-23-25-19-7-8-1-17-28-29-9-10-24-26-21-2-13-6-12-5
Tai30a
19-18-4-24-30-25-5-7-1-22-28-20-11-13-9-16-8-10-17-21-12-29-2-15-3-14-26-27-23-6
Tho30 ∗
9-10-25-30-28-2-27-1-29-19-12-6-13-26-8-17-4-24-5-3-20-18-15-22-21-23-16-14-7-11
Tai35a ∗
19-9-28-12-7-33-13-26-5-2-31-16-24-27-22-15-3-30-29-11-6-25-21-23-34-20-18-4-10-1-14-8-32-35-17
Ste36a ∗
36-5-6-12-11-27-26-25-24-9-4-1-13-20-14-23-21-22-2-8-10-7-28-19-32-34-33-17-18-3-15-16-29-30-31-35
Lipa40a
7-6-14-27-19-37-21-2-16-1-40-24-30-23-5-28-22-8-20-35-32-26-29-3-4-11-36-10-13-38-9-17-31-18-33-15-25-34-39-12
Tai40a
11-18-28-1-5-13-29-10-20-3-25-26-24-32-9-14-12-19-21-36-38-35-37-7-6-40-31-30-27-8-2-16-34-17-39-15-33-22-4-23
Sko42
13-27-20-12-17-7-41-31-10-19-5-42-34-25-2-35-14-26-32-18-11-28-30-33-9-15-40-29-22-39-4-37-21-38-8-23-36-16-24-1-3-6
Sko49
47-13-14-24-1-19-4-36-7-41-9-15-32-17-26-42-35-10-45-37-21-25-20-43-6-27-33-5-28-18-29-34-38-30-16-46-11-48-2-23-8-39-49-40-12-31-3-22-44
Lipa50a
28-32-37-39-49-23-19-44-33-7-14-30-15-5-36-6-17-26-48-25-40-3-45-27-18-31-29-16-9-12-1-8-4-2-50-21-43-35-24-38-34-46-42-13-20-22-41-47-10-11
Wil50
46-28-42-50-17-4-34-36-18-7-22-10-21-6-11-37-39-8-13-26-35-29-30-16-41-45-49-43-2-20-14-5-32-44-3-24-38-23-12-25-1-48-40-31-15-33-47-19-9-27
Lipa60a
25-48-17-4-50-13-16-41-1-37-22-27-46-34-38-12-9-3-8-33-47-54-31-43-40-10-35-23-29-57-2-6-51-56-49-21-30-36-15-39-59-18-52-28-26-44-14-60-32-5-20-11-55-45-53 24-42-7-58-19
Sko64
31-57-53-10-38-54-32-15-2-8-59-49-19-47-11-41-26-17-13-63-62-6-28-34-42-25-21-64-4-51-43-3-33-44-7-16-55-39-30-12-50-37-1-23-24-5-40-27-35-58-45-20-22-48-46 61-9-56-14-52-29-18-60-36
Lipa80a

23-72-79-13-66-45-8-71-59-78-40-37-39-46-6-69-57-74-17-67-1-2-29-50-22-80-14-70-41-10-62-27-7-63-56-47-12-20-51-30-54-34-77-44-75-18-28-36-76-49-21-61-5-25-32-3-43-19-64-35-53-48-11-33-26-52-73-68-9-16-58-24-15-65-55-60-31-42-38-4

Lipa90a

22-11-41-12-21-39-37-36-57-35-44-74-10-3-9-13-59-78-70-32-38-25-60-72-71-24-42-86-77-49-7-27-80-87-2-81-15-79-26-90-16-40-1-56-67-6-47-83-66-18-30-68-14-82-46-20-63-58-88-62-61-84-31-55-69-54-89-29-52-53-17-64-85-33-28-8-45-50-19-73-4-34-75-76-5-51-43-23-65-48

Sko100a ∗

3-35-21-29-23-48-44-7-76-17-45-50-12-72-61-38-92-36-53-41-58-70-66-100-26-47-99-46-64-90-10-24-40-67-85-30-73-51-69-20-42-59-68-8-93-56-81-82-84-87-60-9-86-98-96-89-19-49-2-28-88-62-16-52-4-22-32-34-57-15-27-80-54-94-75-31-43-14-77-6-39-5-13-55-74-95-25-63-79-91-65-83-11-33-71-97-1-78-37-18

Sko100e

61-39-22-21-19-100-33-53-41-63-62-26-34-70-79-91-85-51-59-45-74-23-98-55-27-72-28-50-71-86-47-76-14-69-92-99-10-44-32-1-49-54-12-82-60-56-37-48-78-88-9-42-43-83-11-89-68-66-77-67-40-93-96-64-94-97-20-36-80-81-4-58-25-95-2-29-87-18-73-46-6-3-31-57-90-75-8-7-17-38-13-35-30-84-5-15-52-24-16-65

Wil100 ∗

57-84-74-63-81-14-46-1-9-15-62-24-22-73-31-37-71-29-10-28-35-86-58-3-27-13-39-98-43-100-91-11-93-96-54-45-76-79-56-64-66-8-42-5-4-80-26-90-78-95-60-40-7-75-47-41-69-83-77-88-82-59-38-19-44-61-52-23-65-32-21-17-12-67-2-16-99-72-68-30-85-70-89-97-34-18-53-49-55-87-94-92-20-51-36-6-25-33-48-50

Tai100a

89-3-60-31-1-42-36-95-32-59-41-55-90-52-13-82-81-11-80-43-96-50-2-19-40-26-99-25-87-21-7-5-39-65-29-93-46-48-100-9-37-98-38-91-34-69-68-18-16-67-77-10-63-76-4-24-51-84-54-15-73-66-94-71-22-75-74-53-79-45-6-58-20-47-49-78-72-83-56-33-85-44-70-35-30-27-92-62-23-14-64-12-28-61-57-17-86-8-97-88

Tai100b

85-37-27-81-65-70-50-45-82-48-66-26-94-29-72-95-1-86-99-30-32-53-79-5-19-15-89-80-31-33-68-11-6-77-61-28-43-20-57-41-24-3-78-76-98-54-21-51-87-58-100-39-84-60-64-38-12-36-17-35-44-14-91-92-49-93-75-8-13-69-9-73-90-7-10-74-2-18-71-52-96-34-67-46-25-63-62-42-47-40-97-59-4-83-55-22-56-23-16-88

Tho150

21-105-65-75-115-86-27-50-33-77-26-99-24-15-137-112-119-110-111-30-91-45-3-140-144-37-25-116-81-73-47-72-148-63-61-79-80-134-101-108-48-123-4-11-82-127-124-44-46-16-66-78-150-76-131-55-59-68-64-18-60-97-138-42-100-104-135-103-129-114-71-133-22-92-96-17-94-54-58-146-90-12-56-132-40-52-39-87-93-107-32-88-10-49-141-143-28-1-23-126-19-51-2-109-128-34-117-136-84-89-70-8-122-41-130-95-69-120-53-142-98-29-43-67-118-121-145-113-13-74-139-147-35-38-20-9-149-125-102-14-31-83-106-57-5-36-85-7-6-62

Tai150b ∗

63-84-115-31-65-147-51-123-105-38-30-83-27-58-82-99-19-25-144-111-48-104-76-125-93-47-113-61-109-103-116-69-132-54-146-68-110-37-128-64-130-88-124-92-78-73-41-114-74-13-22-142-138-121-119-46-126-75-42-18-53-137-89-133-131-87-16-39-21-24-120-71-2-9-91-4-97-90-149-26-60-66-122-100-56-95-127-136-20-96-77-57-112-29-139-134-35-141-72-49-32-148-102-80-106-14-33-34-129-36-40-12-98-117-150-7-8-28-143-94-81-107-108-101-55-70-5-23-67-79-118-52-10-43-15-1-3-62-44-85-6-135-50-45-59-140-11-86-17-145

APPENDIX C

INTEGER LINEAR PROGRAMMING WITH *MQO*

TableC.1: E→Equals, G→Greater than or Equal, L→ Less than or Equal

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	obj
x_1	1			-5							
x_2	1				-6						
x_3		1				-4					
x_4		1					-3				
x_5		1						-7			
x_6			1						-4		
x_7			1							-1	
x_8				3	3			3	3		3
x_9					3	3		3			3
x_{10}				1				1	1		1
x_{11}				1		1				1	1
x_{12}							3				3
	E	E	E	G	G	G	G	G	G	G	N
	1	1	1	0	0	0	0	0	0	0	

APPENDIX D

DISTRIBUTED DATABASE DESIGN WITH INTEGER LINEAR PROGRAMMING

TableD.1: Update Costs in our model.

$x_1 \rightarrow 10\text{MB} \times (0.5\text{sec.} / 1\text{MB}) = 5\text{sec.}$
$x_2 \rightarrow 10\text{MB} \times (0.5\text{sec.} / 1\text{MB}) = 5\text{sec.}$
$x_3 \rightarrow 10\text{MB} \times (0.5\text{sec.} / 1\text{MB}) = 5\text{sec.}$
$x_4 \rightarrow 10\text{MB} \times (0.5\text{sec.} / 1\text{MB}) = 5\text{sec.}$
$x_5 \rightarrow 8\text{MB} \times (0.2\text{sec.} / 1\text{MB}) = 1.6\text{sec.}$
$x_6 \rightarrow 8\text{MB} \times (0.2\text{sec.} / 1\text{MB}) = 1.6\text{sec.}$
$x_7 \rightarrow 8\text{MB} \times (0.2\text{sec.} / 1\text{MB}) = 1.6\text{sec.}$
$x_8 \rightarrow 8\text{MB} \times (0.2\text{sec.} / 1\text{MB}) = 1.6\text{sec.}$
$x_9 \rightarrow 6\text{MB} \times (0.1\text{sec.} / 1\text{MB}) = 0.6\text{sec.}$
$x_{10} \rightarrow 6\text{MB} \times (0.1\text{sec.} / 1\text{MB}) = 0.6\text{sec.}$
$x_{11} \rightarrow 6\text{MB} \times (0.1\text{sec.} / 1\text{MB}) = 0.6\text{sec.}$
$x_{12} \rightarrow 6\text{MB} \times (0.1\text{sec.} / 1\text{MB}) = 0.6\text{sec.}$
$x_{13} \rightarrow 5\text{MB} \times (0.05\text{sec.} / 1\text{MB}) = 0.25\text{sec.}$
$x_{14} \rightarrow 5\text{MB} \times (0.05\text{sec.} / 1\text{MB}) = 0.25\text{sec.}$
$x_{15} \rightarrow 5\text{MB} \times (0.05\text{sec.} / 1\text{MB}) = 0.25\text{sec.}$
$x_{16} \rightarrow 5\text{MB} \times (0.05\text{sec.} / 1\text{MB}) = 0.25\text{sec.}$

TableD.2: Cost representations for shared part of Query 1 which originates from sites S_0 and S_3

T_0	t'
S_0	S_0→(100+5)×(10MB ×0.5× (0 Sec./1MB))=0Sec.
S_0	S_1→(100+5)×(10MB ×0.5× (5 Sec./1MB))=2625Sec.
S_0	S_2→(100+5)×(10MB ×0.5×(8 Sec./1MB))=4200Sec.
S_0	S_3→(100+5)×(10MB ×0.5×(17 Sec./1MB))=8925Sec.
S_1	S_0→(100+5)×(10MB ×0.5×(5 Sec./1MB))=2625Sec.
S_1	S_1→(100+5)×(10MB ×0.5×(0 Sec./1MB))=0Sec.
S_1	S_2→(100+5)×(10MB ×0.5×(13 Sec./1MB))=6825Sec.
S_1	S_3→(100+5)×(10MB ×0.5×(12 Sec./1MB))=6300Sec.
S_2	S_0→(100+5)×(10MB ×0.5×(8 Sec./1MB))=4200Sec.
S_2	S_1→(100+5)×(10MB ×0.5×(13 Sec./1MB))=6825Sec.
S_2	S_2→(100+5)×(10MB ×0.5×(0 Sec./1MB))=0Sec.
S_2	S_3→(100+5)×(10MB ×0.5×(14 Sec./1MB))=7350Sec.
S_3	S_0→(100+5)×(10MB ×0.5×(17 Sec./1MB))=8925Sec.
S_3	S_1→(100+5)×(10MB ×0.5×(12 Sec./1MB))=6300Sec.
S_3	S_2→(100+5)×(10MB ×0.5×(14 Sec./1MB))=7350Sec.
S_3	S_3→(100+5)×(10MB ×0.5×(0 Sec./1MB))=0Sec.

TableD.3: Cost of second part of Query 1 representations which originates from site S_0.

t'	T_1	t_0
S_0	S_0	S_0→100×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=0sec.
S_0	S_1	S_0→100×[(10MB ×0.5× (5sec. / 1MB))+(10MB ×0.5×8×1×0.3×(5sec./1MB))]=8500sec.
S_0	S_2	S_0→100×[(10MB ×0.5× (8sec. / 1MB))+(10MB ×0.5×8×1×0.3×(5sec./1MB))]=13600sec.
S_0	S_3	S_0→100×[(10MB ×0.5× (17sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=28900sec.
S_1	S_0	S_0→100×[(10MB ×0.5× (5sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=2500sec.
S_1	S_1	S_0→100×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(5sec./1MB))]=6000sec.
S_1	S_2	S_0→100×[(10MB ×0.5× (13sec. / 1MB))+(10MB ×0.5×8×1×0.3×(8sec./1MB))]=16100sec.
S_1	S_3	S_0→100×[(10MB ×0.5× (12sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=26400sec.
S_2	S_0	S_0→100×[(10MB ×0.5× (8sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=4000sec.
S_2	S_1	S_0→100×[(10MB ×0.5× (13sec. / 1MB))+(10MB ×0.5×8×1×0.3×(5sec./1MB))]=12500sec.
S_2	S_2	S_0→100×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×8sec./1MB))]=9600sec.
S_2	S_3	S_0→100×[(10MB ×0.5× (14sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=27400sec.
S_3	S_0	S_0→100×[(10MB ×0.5× (17sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=8500sec.
S_3	S_1	S_0→100×[(10MB ×0.5× (12sec. / 1MB))+(10MB ×0.5×8×1×0.3×(5sec./1MB))]=12000sec.
S_3	S_2	S_0→100×[(10MB ×0.5× (14sec. / 1MB))+(10MB ×0.5×8×1×0.3×(8sec./1MB))]=16600sec.
S_3	S_3	S_0→100×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=20400sec.

TableD.4: Cost of second part of Query 1 representations which originates from site S_3.

t'	T_1	t_0
S_0	S_0	S_3→5×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=1020sec.
S_0	S_1	S_3→5×[(10MB ×0.5× (5sec. / 1MB))+(10MB ×0.5×8×1×0.3×(12sec./1MB))]=845sec.
S_0	S_2	S_3→5×[(10MB ×0.5× (8sec. / 1MB))+(10MB ×0.5×8×1×0.3×(14sec./1MB))]=1040sec.
S_0	S_3	S_3→5×[(10MB ×0.5× (17sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=425sec.
S_1	S_0	S_3→5×[(10MB ×0.5× (5sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=1145sec.
S_1	S_1	S_3→5×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(12sec./1MB))]=720sec.
S_1	S_2	S_3→5×[(10MB ×0.5× (13sec. / 1MB))+(10MB ×0.5×8×1×0.3×(14sec./1MB))]=1165sec.
S_1	S_3	S_3→5×[(10MB ×0.5× (12sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=300sec.
S_2	S_0	S_3→5×[(10MB ×0.5× (8sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=1220sec.
S_2	S_1	S_3→5×[(10MB ×0.5× (13sec. / 1MB))+(10MB ×0.5×8×1×0.3×(12sec./1MB))]=1045sec.
S_2	S_2	S_3→5×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(14sec./1MB))]=840sec.
S_2	S_3	S_3→5×[(10MB ×0.5× (14sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=350sec.
S_3	S_0	S_3→5×[(10MB ×0.5× (17sec. / 1MB))+(10MB ×0.5×8×1×0.3×(17sec./1MB))]=1445sec.
S_3	S_1	S_3→5×[(10MB ×0.5× (12sec. / 1MB))+(10MB ×0.5×8×1×0.3×(12sec./1MB))]=1020sec.
S_3	S_2	S_3→5×[(10MB ×0.5× (14sec. / 1MB))+(10MB ×0.5×8×1×0.3×(14sec./1MB))]=1190sec.
S_3	S_3	S_3→5×[(10MB ×0.5× (0sec. / 1MB))+(10MB ×0.5×8×1×0.3×(0sec./1MB))]=0 sec.

TableD.5: Cost of first part of the Query 2 representations which originate from S_2.

T_2	t"
S_0	S_0 →20×(6MB ×0.4× (0sec./1MB))=0sec.
S_0	S_1 →20×(6MB ×0.4× (5sec./1MB))=240sec.
S_0	S_2 →20×(6MB ×0.4× (8sec./1MB))=384sec.
S_0	S_3 →20×(6MB ×0.4× (17sec./1MB))=816sec.
S_1	S_0 →20×(6MB ×0.4× (5sec./1MB))=240sec.
S_1	S_1 →20×(6MB ×0.4× (0sec./1MB))=0sec.
S_1	S_2 →20×(6MB ×0.4× (13sec./1MB))=624sec.
S_1	S_3 →20×(6MB ×0.4× (12sec./1MB))=576sec.
S_2	S_0 →20×(6MB ×0.4× (8sec./1MB))=384sec.
S_2	S_1 →20×(6MB ×0.4× (13sec./1MB))=624sec.
S_2	S_2 →20×(6MB ×0.4× (0sec./1MB))=0sec.
S_2	S_3 →20×(6MB ×0.4× (14sec./1MB))=672sec.
S_3	S_0 →20×(6MB ×0.4× (17sec./1MB))=816sec.
S_3	S_1 →20×(6MB ×0.4× (12sec./1MB))=576sec.
S_3	S_2 →20×(6MB ×0.4× (14sec./1MB))=672sec.
S_3	S_3 →20×(6MB ×0.4× (0sec./1MB))=0sec.

TableD.6: Cost of second part of the Query 2 representations which originate from S_2.

t”	T_3	t_1
S_0	S_0	S_2 →20×[(6MB ×0.4× (0sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(8sec./1MB))]=153.6sec.
S_0	S_1	S_2 →20×[(6MB ×0.4× (5sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(13sec./1MB))]=489.6sec.
S_0	S_2	S_2 →20×[(6MB ×0.4× (8sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(0sec./1MB))]=384sec.
S_0	S_3	S_2 →20×[(6MB ×0.4× (17sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(14sec./1MB))]=1084.8sec.
S_1	S_0	S_2 →20×[(6MB ×0.4× (5sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(8sec./1MB))]=393.6sec.
S_1	S_1	S_2 →20×[(6MB ×0.4× (0sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(13sec./1MB))]=249.6sec.
S_1	S_2	S_2 →20×[(6MB ×0.4× (13sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(0sec./1MB))]=624sec.
S_1	S_3	S_2 →20×[(6MB ×0.4× (12sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(14sec./1MB))]=844.8sec.
S_2	S_0	S_2 →20×[(6MB ×0.4× (8sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(8sec./1MB))]=537.6sec.
S_2	S_1	S_2 →20×[(6MB ×0.4× (13sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(13sec./1MB))]=873.6sec.
S_2	S_2	S_2 →20×[(6MB ×0.4× (0sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(0sec./1MB))]=0sec.
S_2	S_3	S_2 →20×[(6MB ×0.4× (14sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(14sec./1MB))]=940.8sec.
S_3	S_0	S_2 →20×[(6MB ×0.4× (17sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(8sec./1MB))]=969.6sec.
S_3	S_1	S_2 →20×[(6MB ×0.4× (12sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(13sec./1MB))]=825.6sec.
S_3	S_2	S_2 →20×[(6MB ×0.4× (14sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(0sec./1MB))]=672sec.
S_3	S_3	S_2 →20×[(6MB ×0.4× (0sec. / 1MB))+(6MB ×0.4×5×0.8×0.1×(14sec./1MB))]=268.8sec.

TableD.7: E→Equals, G→Greater than or Equal, L→ Less than or Equal

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	c_{26}	c_{27}	c_{28}	c_{29}	c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	c_{36}	c_{37}	c_{38}	c_{39}	c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	obj
x_1	-1																																1				10				1				1
x_2		-1																															1					10			1				1
x_3			-1																														1						10		1				1
x_4				-1																													1							10	1				1
x_5									-1								-1																	1			8					1			0.4
x_6										-1								-1																1				8				1			0.4
x_7											-1								-1															1					8			1			0.4
x_8												-1								-1														1						8		1			0.4
x_9																					-1														1		6						1		3
x_{10}																						-1													1			6					1		3
x_{11}																							-1												1				6				1		3
x_{12}																								-1											1					6			1		3
x_{13}																													-1							1	5							1	0.5
x_{14}																														-1						1		5						1	0.5
x_{15}																															-1					1			5					1	0.5
x_{16}																																-1				1				5				1	0.5
x_{17}	1				-1								-1																																0
x_{18}	1					-1								-1																															2625
x_{19}	1						-1								-1																														4200
x_{20}	1							-1								-1																													8925
x_{21}		1			-1								-1																																2625
x_{22}		1				-1								-1																															0
x_{23}		1					-1								-1																														6825
x_{24}		1						-1								-1																													6300
x_{25}			1		-1								-1																																4200
x_{26}			1			-1								-1																															6825
x_{27}			1				-1								-1																														0
x_{28}			1					-1								-1																													7350
x_{29}				1	-1								-1																																8925
x_{30}				1		-1								-1																															6300
x_{31}				1			-1								-1																														7350
x_{32}				1				-1								-1																													0
x_{33}					1				1																																				0
x_{34}					1					1																																			8500
x_{35}					1						1																																		13600
x_{36}					1							1																																	28900
x_{37}						1			1																																				2500
x_{38}						1				1																																			6000
x_{39}						1					1																																		16100
x_{40}						1						1																																	26400
x_{41}							1		1																																				4000
x_{42}							1			1																																			12500
x_{43}							1				1																																		9600
x_{44}							1					1																																	27400
x_{45}								1	1																																				8500
x_{46}								1		1																																			12000
x_{47}								1			1																																		16600
x_{48}								1				1																																	20400
x_{49}													1				1																												1020
x_{50}													1					1																											845
x_{51}													1						1																										1040
x_{52}													1							1																									425
x_{53}														1			1																												1145
x_{54}														1				1																											720
x_{55}														1					1																										1165
x_{56}														1						1																									300
x_{57}															1		1																												1220
x_{58}															1			1																											1045
x_{59}															1				1																										840
x_{60}															1					1																									350
x_{61}																1	1																												1445
x_{62}																1		1																											1020
x_{63}																1			1																										1190
x_{64}																1				1																									0
x_{65}																					1				-1																				0
x_{66}																					1					-1																			240
x_{67}																					1						-1																		384
x_{68}																					1							-1																	816
x_{69}																						1			-1																				240
x_{70}																						1				-1																			0
x_{71}																						1					-1																		624
x_{72}																						1						-1																	576
x_{73}																							1		-1																				384
x_{74}																							1			-1																			624
x_{75}																							1				-1																		0
x_{76}																							1					-1																	672
x_{77}																								1	-1																				816
x_{78}																								1		-1																			576
x_{79}																								1			-1																		672
x_{80}																								1				-1																	0
x_{81}																									1				1																153.6
x_{82}																									1					1															489.6
x_{83}																									1						1														384
x_{84}																									1							1													1084.8
x_{85}																										1			1																393.6
x_{86}																										1				1															249.6
x_{87}																										1					1														624
x_{88}																										1						1													844.8
x_{89}																											1		1																537.6
x_{90}																											1			1															873.6
x_{91}																											1				1														0
x_{92}																											1					1													940.8
x_{93}																												1	1																969.6
x_{94}																												1		1															825.6
x_{95}																												1			1														672
x_{96}																												1				1													268.8
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	L	L	L	L	L	L	L	L	G	G	G	G	N
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	14	8	6	5	1	1	1	1	

APPENDIX E

DDB DESIGN WITH *QAP*

TableE.1: Cost Calculations

Q1:S0	Q1:S3	Q2:S2
0123	0123	0123
24147sec	1212.05sec	3281.48sec
0132	0132	0132
26067sec	1308.05 sec	3665.48sec
0213	0213	0213
22948.2sec	1152.85 sec	3029.0sec
0231	0231	0231
22948.2sec	1152.85 sec	3029.0sec
0312	0312	0312
26071.8sec	1303.25 sec	3666.44sec
0321	0321	0321
24151.8sec	1207.25 sec	3282.44sec
1032	1032	1032
26065sec	1310.05 sec	3660.68sec
1023	1023	1023
24145sec	1214.05 sec	3276.68sec
1230	1230	1230
29448.2sec	1477.85 sec	3653sec
1203	1203	1203
25448.2sec	1277.85 sec	3269sec
1320	1320	1320
30651.8sec	1532.25 sec	3906.44sec
1302	1302	1302
28571.8sec	1428.25 sec	3906.44sec
2013	2013	2013
22945.0sec	1154.05 sec	3036.68sec
2031	2031	2031
22945.0	1154.05	3036.68sec
2103	2103	2103
25447.0	1277.05	3281.48sec
2130	2130	2130
29447.0	1477.05	3665.48sec
2301	2301	2301
25451.8	1272.25	3282.44sec
2310	2310	2310
29451.8	1472.25	3666.44sec
3021	3021	3021
24145.0	1214.05	3276.68sec
3012	3012	3012
26065	1310.05	3660.68sec
3120	3120	3120
30647.0	1537.05	3905.48sec
3102	3102	3102
28567	1433.05	3905.48sec
3210	3210	3210
29448.2	1477.85	3653sec
3201	3201	3201
25448.2	1277.85	3269sec

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Tosun, Umut
Nationality: Turkish (TC)
Date and Place of Birth: 05.04.1982, Antalya
Marital Status: Single
Phone: +90 312 292 5044
e-Mail umut_tosun2003@yahoo.com

EDUCATION

Degree	Institution	Year of Graduation
MS	Bilkent University-Computer Engineering	2007
BS	Iztech-Computer Engineering	2004
High School	Antalya High School	2000

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2012-present	Siemens Enterprise Communications	Software Engineer
2011-2012	Argela Information Technologies	System Engineer
2008-2011	Aselsan	Software Engineer
2007-2008	Cybersoft Information Technologies	Software Engineer

PUBLICATIONS

U. Tosun, T. Dokeroglu, A. Cosar, ”A Robust Island Parallel Genetic Algorithm for the Quadratic Assignment Problem”, International Journal of Production Research, 2012.

International Conference Publications

U. Tosun, T. Dokeroglu, A. Cosar, PHGETS:Parallel Hybrid Genetic-Tabu Search Algorithm for the Quadratic Assignment Problem, Applied Mathematics and Computation Journal (submitted)

U. Tosun, T. Dokeroglu, A. Cosar, Heuristic Algorithms for Fragment Allocation in a Distributed Database System 27th International Symposium on Computer and Information Sciences,October 3-5,

2012, Paris, France.

U. Tosun, T. Dokeroglu, and A. Cosar, 2013, A New Parallel Genetic Algorithm for Reducing the Bullwhip Effect in an Automotive Supply Chain, International Federation of Automatic Control (IFAC), Saint Petersburg, Russia.

T. Dokeroglu, U. Tosun, A. Cosar, Parallel Optimization with Mutation Operator for the Quadratic Assignment Problem Proceedings of WIVACE 2012, Italian Workshop on Artificial Life and Evolutionary Computation, Parma, Italy.

T. Dokeroglu, U. Tosun, and A. Cosar, 2013, Evaluating the Performance of Recombination Operators with Island Parallel Genetic Algorithms, International Federation of Automatic Control (IFAC), Saint Petersburg, Russia.

T.Dokeroglu, U. Tosun, A. Cosar, Particle Swarm Intelligence as a Novel Heuristic for the Optimization of Distributed Database Queries, The 6th International Conference on Application of Information and Communication Technologies AICT 2012 Georgia, Tbilisi, 17-19 October 2012.

National Conference Publications

T. Dokeroglu, M.S. Cinar, U. Tosun, A. Cosar, Dağıtık Veritabanı Zincir Bağlı Birleştirme Sorgularının Paralel Genetik Algoritmalar ile Eniyilenmesi 3. Ulusal Yüksek Başarımlı Hesaplama Konferansı 2012, Bilkent Üniversitesi, Ankara, Türkiye.