Distributed Database Design using Evolutionary Algorithms

Umut Tosun

Abstract: The performance of a distributed database system depends particularly on the site-allocation of the fragments. Queries access different fragments among the sites, and an originating site exists for each query. A data allocation algorithm should distribute the fragments to minimize the transfer and settlement costs of executing the query plans. The primary cost for a data allocation algorithm is the cost of the data transmission across the network. The data allocation problem in a distributed database is NP-complete, and scalable evolutionary algorithms were developed to minimize the execution costs of the query plans. In this paper, quadratic assignment problem heuristics were designed and implemented for the data allocation problem. The proposed algorithms find near-optimal solutions for the data allocation problem. In addition to the fast ant colony, robust tabu search, and genetic algorithm solutions to this problem, we propose a fast and scalable hybrid genetic multi-start tabu search algorithm that outperforms the other well-known heuristics in terms of execution time and solution quality.

Index Terms: Ant colony optimization, distributed database design, hybrid algorithms, robust tabu search.

I. INTRODUCTION

Fragmentation and data allocation [1] are the two most critical problems when designing distributed databases. Before they are assigned to a site, the relations are mostly partitioned either horizontally or vertically. The replication of fragments is another issue to consider during a design. The memory capacity, communication channels, and processing power are some other design parameters. During the most-frequent queries, the data transferred between sites must be minimized, the locality of the related fragments must be maintained, and the data volume kept on the site must be smaller than the memory size.

A data allocation problem (DAP) is an optimization problem with certain constraints [2]. For instance, the disk I/O speed, parallel query execution, network load, and load balancing of the servers are design parameters that need to be handled. A DAP is an NP-complete problem regardless of these parameters. The most significant factor contributing to the response time of a query is the delay of a data transfer among sites. Therefore, most algorithms deal only with the delay of data to achieve acceptable execution times. A quadratic assignment problem (QAP) has some similarities with a DAP, and also keeps track of the resource locality.

The QAP was first presented by Koopmans and Beckman [3]. A set of $n$ facilities and $n$ locations is maintained, and the distances between locations are defined. A flow is defined for the amount of supplies to be transferred between pairs of facilities. The problem is assigning the facilities to different locations to minimize the flow between each pair multiplied by the distance between their locations. The QAP can express the dependence of the fragments and sites, where the fragments are considered to be facilities and the sites are considered to be locations. The flow between two facilities is the amount of data transferred between two sites, and the distance between two locations is the cost of sending a data item between the two sites.

The remainder of this paper is organized as follows: Section II describes an overview of both a DAP and a QAP. In Section III, the mathematical formulation behind the modeling of a DAP as a QAP is handled. Section IV presents the algorithms used. The experimental environment is then described in Section V. Finally, Section VI provides some concluding remarks regarding this research.

II. RELATED WORKS

A DAP can be solved using a static or dynamic allocation. Static algorithms exploit the defined prerequisites, whereas dynamic algorithms adapt to the modifications [4]. DAPs and QAPs have been widely studied by the database research community. For example, Ceri and Plagatti proposed a greedy algorithm for redundant and non-redundant data [5]. Ahmad and Karlapalem [6] introduced a query-driven strategy. Adl and Ranjooohi [7], transformed a QAP formulation into a DAP, the model of which handles the non-redundant allocation of data with certain capacity constraints. Distributed database management system (DDBMS) queries access several tables and fragments over a network. A query is initialized from a site, and the major portion of the plan execution cost is from the retrieval of fragments from different sites. Data allocation algorithms attempt to assign fragments to sites in such a manner that minimizes the total cost of the data transfer while executing user and/or application queries. DAP algorithms aim to find an optimal fragment-assignment solution while also taking into account the replications, update costs, and average query response times.

Different queries may share the same sub-tasks, and the same queries may be issued from different originating sites. A DDBMS design is a problem with a set of multiple objectives including the efficient usage of computer storage and processing resources, and a minimization of the query response times, while taking care not to violate the constraints regarding the site capacity. It is necessary to model the problem in a way that satisfies all of these criteria. Several algorithms for
data allocation and data fragmentation problems in distributed
databases have been proposed in the literature. Techniques based
on genetic algorithms (GAs) have been used by Frieder and
Siegelmann [8]. However, their formulations do not consider
the site capacities or replication of fragments and/or tables to
improve the query response times. Ahmad [6] proposed the
use of a GA, simulated annealing, and mean field annealing
solutions, where non-redundant data (i.e., replications) were
not considered. Adl [7] proposed an ant colony heuristic, and
modeled the DAP as a QAP; however, the update and replication
costs are not handled in this work.

III. SOLUTION FORMULATION WITH QUADRATIC
ASSIGNMENT OPTIMIZATION

The data allocation cost can be represented as the sum of
direct and indirect transaction-fragment dependencies [7]. A
transaction \( t \) and fragment \( f \) have a direct dependency if the data
from the container site of \( f \) are transmitted for every execution
of \( t \). There is an indirect dependency if the data need to be
transmitted to a site other than from where the transaction
originates. The data allocation cost is expressed as the sum of
costs \( Cst1 \) and \( Cst2 \), as described in (1).

\[
Cst(\Phi) = Cst1(\Phi) + Cst2(\Phi).
\] (1)

Here, \( Cst1 \) is the multiplication of two matrices, \( STFR \) and
\( UC \), where \( STFR \) is the site fragment dependency matrix and
\( UC \) is the unit communication matrix. \( UC \) holds the network
communication costs among the sites. \( \Phi \) is an \( m \) element vector
and \( \Phi_j \) represents the site where \( f_j \) is stored. Partial cost matrix
\( PCST_{1n \times m} \) is the cost of fragment \( f_j \) to be stored in site \( s_i \).
The unit partial cost matrix is represented in (2).

\[
pcst_{ij} = \sum_{q=1}^{n} uc_{iq} \times stfr_{aq},
\] (2)

The unit partial cost \( pcst_{ij} \), for each \( i \) and \( j \) is calculated,
and \( Cst1 \) is expressed through (3).

\[
Cst1(\Phi) = \sum_{j=1}^{m} pcst_{1\Phi_j, j}.
\] (3)

An inter-fragment dependency matrix (\( FRDEP \)) is the multipli-
cation of the matrices \( QFR_{l \times m \times m} \) and \( Q_{1 \times m \times m} \). The execution
frequencies of the transactions are represented by the matrix
\( QFR \) which is multiplied with matrix \( Q \) to obtain the
\( FRDEP \) matrix of the inter-fragment dependencies. The indirect
transaction-fragment dependency is shown through \( Q \).
The indirect transaction-fragment dependency cost \( Cst2 \) is a
form of \( QAP \) and is represented in (4).

\[
Cst2(\Phi) = \sum_{j_1=1}^{m} \sum_{j_2=1}^{m} frdep_{j_1, j_2} \times uc_{\Phi_1, j_2} \times uc_{\Phi_2, j_2}.
\] (4)

IV. PROPOSED ALGORITHMS FOR THE DATA
ALLOCATION PROBLEM

A. Genetic Algorithm

GAs exploit the selection, crossover, and mutation opera-
tions on an initial randomly chosen population. They create
new generations, and a fitness function exists to find the best
individuals within the population [9]. The termination condition
may be defined depending on the total execution time, number
of generations produced, or whether no improvements in the
average fitness value of the population have been found [10].

A partially mapped crossover (PMX) is used for the GA
because it is one of the best performing operators for a QAP
solution. The chromosome structure of the solution is shown
in Fig. 1. Facilities are placed in an array of locations. The
PMX copies a random segment from parent1 to the first child.
It looks for elements in that segment of parent2 that have not
been copied, starting from the initial crossover point. For each
of these elements, e.g., \( i \), PMX looks in the offspring to see what
element \( j \) has been copied in its place from parent1, and places
\( i \) into the position occupied by \( j \) in parent2 because we know that \( j \)
will not be put there. If the position in the offspring occupied by
\( j \) in parent2 has already been filled by \( k \), we put \( i \) in the position
occupied by \( k \) in parent2. The rest of the offspring can be filled
from parent2, and the second child is created in a similar manner

Algorithm 1 Standard ant system

1. Pheromone trail is initialized
2. while stopping criterion is not met do
3.     for each ant in the colony do
4.         Construct a new solution with the current pheromone
5.         trail
6.         Construct an evaluation of the partial solution
7.     end for
8.     Update pheromone trail
9. end while

B. Fast Ant System

Ant colony optimization (ACO) was first proposed by Dorigo
to solve hard combinatorial problems [12]. It exploits a model
based on the real-life cooperation of self-organizing ants. Taill-
lard [13] proposed the fast ant system (FANT) for solving a
QAP by incorporating both diversification and intensification.
This improves the best solution up to the current execution
time of the algorithm in a systematic way by clearing the memory and reducing the weight of the best solution if the process is stagnating. The ant process constructs new solutions by randomly choosing the location of the facilities with a certain probability. The solutions are then improved through a local search and sent to the queen process. While implementing the automatic intensification and diversification, the queen process requires a parameter for managing the traces. The algorithm for a standard ant system is shown in Algorithm 1.

### C. Robust Tabu Search

A robust tabu search (RTS) is a well-known optimization algorithm for producing high-quality solutions [14]. It is a variant of a simple TS algorithm. RTS starts with the steepest descent algorithm, and makes up and down movements toward the solution. A tabu list is kept to prevent backward movement for a defined number of moves. RTS tries to evade the local minimum values even when it finds a solution worse than the previous one. It uses adaptive memory, and several of its variants use intensification and diversification to obtain better solutions. RTS has an adaptive tabu list size, which it reduces in order to search near a local minimum, although it can also expand the list size to evade this minimum.

Several aspiration criteria are defined to create exceptions to the restrictions in RTS. RTS has short-term memory and does not maintain the statistics of highly frequent solutions in the way that long-term memory algorithms do. The algorithm generates new permutations by changing the previous allocations of two facilities, which is called a two-way exchange, thereby saving an important amount of execution time. The cost between the old and new permutations is stored in a matrix. Instead of calculating the cost for the whole permutation vector when calculating the cost for a new permutation, these costs are added to the total cost. Although backward movement is forbidden, certain moves are exempt from this rule when they satisfy an aspiration criterion. The tabu list keeps track of the forbidden moves. There is also a parameter called the “total number of failures,” which defines the number of unsuccessful iterations for terminating a search for a better solution. The RTS algorithm is similar to Algorithm 2.

### D. Hybrid Genetic Multi-Start Tabu Search Algorithm

The hybrid genetic multi-start tabu search algorithm (HG-MTS) is a hybrid of a GA and multi-start RTS. HG-MTS is a two-step algorithm consisting of a seed generation and TS diversification. Fig. 2 shows the stages of this particular algorithm. The TS diversification phase uses the diversification operator of a cooperative parallel tabu search [16]. After choosing a high-quality seed, multi-start TS conducts a stepwise procedure to determine the best diversification toward the solution.

---

**Algorithm 2** Robust tabu search algorithm [14]

**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, MaxPer, 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×j+i))/l = 0...n-l, j = 0...n-l

for (iteration = 1; iteration < MaxIter; iteration++) {

  for each Neighbor (i, j) {
    current1 = TABU_LIST[i][CurSol[j]];
    current2 = TABU_LIST[i][CurSol[j]];
    Authorized = (current1 < iteration && iteration < Aspiration);
    Aspired = (current1 < iteration && iteration < Aspiration);
    if (Aspired && Already_Aspired)
      TABU_LIST[i_retained][j_retained] =
      if (Aspired) Already_Aspired = true;
    }
    if (i, retained != infinite)
      if (iteration < BestCost)
        BestCost = CurCost;
      UPDATE_MOVE_COSTS(FLOW, DIST, CurSol, Delta, i, j, i_retained, j_retained);
  }
}

---

### V. EXPERIMENTAL SETUP AND TEST RESULTS

#### A. Experimental Environment

We tested the proposed algorithms using a number of different experiments. For each test, one of the parameters was varied whereas the others were fixed. The algorithms were tested using the same test data, which were generated based on the rules defined in subsection B. The experiments were performed using a 2.21 GHz AMD Athlon (TM) 64 × 2 dual processor with 2 GB of RAM and MS Windows 7 (TM) operating system. The implementation language used was C++. The test data were generated according to the experimental environment described by Adl and Ranjooohi [7]. The only difference is that we chose a unit cost in range of [0,1]. Our test data generator obtained the number of fragments m, number of sites n, and other parameters as inputs, and created a random DAP instance.

We chose the fragment size randomly from the range [c/10, 20 × c/10], where c is a number between 10 and 1000. The random choice of fragments is defined using a constraint because a fragment should be placed at a site with a capacity larger than the fragment size. We chose the site capacities in [1, 2 × m/n – 1]. The sum of the site capacities should be equal to the total fragment size m, where n is the total number of sites. We assumed that the number of sites n is equal to the
number of fragments \( m \). We selected the unit transmission costs as a random number within the range of \([0,1]\). We generated a random probability request per transaction \((RT)\) to allow each transaction to be requested at a site. Transaction fragment dependency is also represented using the probability access per fragment \((AF)\). The site fragment frequency matrix, \(FREQ\), was determined as the multiplication of probability \(RT\) and a random frequency of range \([1, 1000]\). A transaction fragment dependency matrix is generated as a multiplication of \(AF\) and a uniformly distributed random value in \([0, f_j]\), with \(f_j\) 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 a multiplication of the matrices \(QFR_{m \times m}\) and \(Q_{m \times m}\), where \(QFR\) takes into account the execution frequencies of the transactions and \(Q\) represents the indirect

### Table 1. Genetic algorithm performance on DAP-20 instance.

<table>
<thead>
<tr>
<th>Population</th>
<th>Generation</th>
<th>Cost</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>50</td>
<td>2.655762</td>
<td>4.214</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>2.674170</td>
<td>7.777</td>
</tr>
<tr>
<td>250</td>
<td>150</td>
<td>2.715429</td>
<td>14.878</td>
</tr>
<tr>
<td>250</td>
<td>200</td>
<td>2.673403</td>
<td>15.171</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>2.666135</td>
<td>18.124</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td>2.640902</td>
<td>15.451</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>2.660429</td>
<td>19.617</td>
</tr>
<tr>
<td>500</td>
<td>150</td>
<td>2.651025</td>
<td>35.112</td>
</tr>
<tr>
<td>500</td>
<td>200</td>
<td>2.649761</td>
<td>53.737</td>
</tr>
<tr>
<td>500</td>
<td>250</td>
<td>2.649985</td>
<td>76.297</td>
</tr>
<tr>
<td>750</td>
<td>50</td>
<td>2.653905</td>
<td>18.796</td>
</tr>
<tr>
<td>750</td>
<td>100</td>
<td>2.636911</td>
<td>49.153</td>
</tr>
<tr>
<td>750</td>
<td>150</td>
<td>2.646546</td>
<td>63.947</td>
</tr>
<tr>
<td>750</td>
<td>200</td>
<td>2.631491</td>
<td>117.159</td>
</tr>
<tr>
<td>750</td>
<td>250</td>
<td>2.648080</td>
<td>173.525</td>
</tr>
<tr>
<td>1,000</td>
<td>50</td>
<td>2.638806</td>
<td>27.666</td>
</tr>
<tr>
<td>1,000</td>
<td>100</td>
<td>2.603268</td>
<td>62.974</td>
</tr>
<tr>
<td>1,000</td>
<td>150</td>
<td>2.637986</td>
<td>84.13</td>
</tr>
<tr>
<td>1,000</td>
<td>200</td>
<td>2.629396</td>
<td>123.789</td>
</tr>
<tr>
<td>1,000</td>
<td>250</td>
<td>2.638621</td>
<td>148.553</td>
</tr>
<tr>
<td>1,250</td>
<td>50</td>
<td>2.639087</td>
<td>54.687</td>
</tr>
<tr>
<td>1,250</td>
<td>100</td>
<td>2.640518</td>
<td>167.222</td>
</tr>
<tr>
<td>1,250</td>
<td>150</td>
<td>2.648000</td>
<td>148.553</td>
</tr>
<tr>
<td>1,250</td>
<td>200</td>
<td>2.648000</td>
<td>148.553</td>
</tr>
<tr>
<td>1,250</td>
<td>250</td>
<td>2.629725</td>
<td>417.682</td>
</tr>
</tbody>
</table>

### Table 2. Genetic algorithm performance on DAP-50 instance.

<table>
<thead>
<tr>
<th>Population</th>
<th>Generation</th>
<th>Cost</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>50</td>
<td>55.129698</td>
<td>17.581</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>55.330358</td>
<td>33.727</td>
</tr>
<tr>
<td>250</td>
<td>150</td>
<td>55.235938</td>
<td>55.411</td>
</tr>
<tr>
<td>250</td>
<td>200</td>
<td>55.116918</td>
<td>46.914</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>55.229595</td>
<td>66.79</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td>54.946089</td>
<td>28.204</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>54.932897</td>
<td>79</td>
</tr>
<tr>
<td>500</td>
<td>150</td>
<td>54.833906</td>
<td>86.822</td>
</tr>
<tr>
<td>500</td>
<td>200</td>
<td>54.945268</td>
<td>123.973</td>
</tr>
<tr>
<td>500</td>
<td>250</td>
<td>54.908336</td>
<td>135.938</td>
</tr>
<tr>
<td>750</td>
<td>50</td>
<td>55.121281</td>
<td>41.009</td>
</tr>
<tr>
<td>750</td>
<td>100</td>
<td>54.889295</td>
<td>76.333</td>
</tr>
<tr>
<td>750</td>
<td>150</td>
<td>54.681993</td>
<td>143.941</td>
</tr>
<tr>
<td>750</td>
<td>200</td>
<td>54.650527</td>
<td>183.265</td>
</tr>
<tr>
<td>750</td>
<td>250</td>
<td>54.667462</td>
<td>224.648</td>
</tr>
<tr>
<td>1,000</td>
<td>50</td>
<td>55.002209</td>
<td>60.278</td>
</tr>
<tr>
<td>1,000</td>
<td>100</td>
<td>54.629883</td>
<td>154.437</td>
</tr>
<tr>
<td>1,000</td>
<td>150</td>
<td>54.627740</td>
<td>207.555</td>
</tr>
<tr>
<td>1,000</td>
<td>200</td>
<td>54.764319</td>
<td>471.978</td>
</tr>
<tr>
<td>1,000</td>
<td>250</td>
<td>54.995658</td>
<td>644.301</td>
</tr>
<tr>
<td>1,250</td>
<td>50</td>
<td>54.953858</td>
<td>90.265</td>
</tr>
<tr>
<td>1,250</td>
<td>100</td>
<td>54.783223</td>
<td>310.599</td>
</tr>
<tr>
<td>1,250</td>
<td>150</td>
<td>54.689684</td>
<td>359.574</td>
</tr>
<tr>
<td>1,250</td>
<td>200</td>
<td>54.651822</td>
<td>499.853</td>
</tr>
<tr>
<td>1,250</td>
<td>250</td>
<td>54.676825</td>
<td>460.665</td>
</tr>
</tbody>
</table>
transaction fragment dependency. We used almost the same parameters as Adl and Rankoohi [7] to better understand the performances of these algorithms in the literature.

### B. Experimental Results

We performed several tests using a 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. In addition, three configuration settings were selected as GA1, GA2, and GA3 after the experiments shown in Tables 1, 2, and 3. GA1 uses a population size of 1,000 and 200 generations. GA2 uses a population size of 1,250 and 200 generations. Finally, GA3 uses a population size of 1,250 and 150 generations. We determined experimentally that these are the best performing parameters. Furthermore, these parameters reflect a performance trade-off among the values because they were chosen in such a way as to minimize the execution time while showing near-optimal solutions.

We used FANT [13] with the parameter $R = 5$ for managing the traces and 20,000 iterations. In addition, we used the aspiration parameter a maximum of 200,000 failures and $(9 \times n)/10$ and $(11 \times n)/10$ for the lower and upper limits of the tabu list, respectively, where $n$ is the instance size. We used a population size of 250, and 50 generations, for the initial phase of HG-MTS. The diversification phase uses 1,000 for the maximum number of failures, and $(n \times n)/10$ and $(n \times n)/10$ for the lower and upper limits of the tabu list, respectively. These are the optimal parameters reported for both algorithms [13], [14]. After completing the experiments on instances ranging from a size of 5 to a size of 100, it was concluded that HG-MTS outperforms the other algorithms in terms of both time and cost measurements. Only RTS can achieve better results than HG-MTS for a few instances. However, HG-MTS executes more quickly than all of the other methods for all instances as shown in Tables 4, 5, Figs. 3 and 4.

### VI. CONCLUSIONS

In this paper, we introduced a new set of quadratic assignment optimization algorithms for designing a distributed database using non-redundant data. We used a well-known genetic algorithm, the fast ant system, and a robust tabu search for the solutions of the data allocation problem. Furthermore, we implemented a more efficient algorithm called HG-MTS by running a modified version of the robust tabu search after operating the genetic algorithm for a number of generations. The main contributions of this work are modeling the problem with using three prevailing algorithms, and the introduction of
the new tabu search based algorithm. In our experiments, the execution times and optimality of the different versions of the quadratic assignment problem algorithms were compared. HG-MTS was shown to outperform the genetic algorithm, fast ant system, and robust tabu search in terms of solution quality and execution times for almost all cases for the data allocation problem. It was observed that the robust tabu search and HG-MTS algorithms outperform the other algorithms particularly when the instance sizes increase. For the smaller instances, it is also obvious that these algorithms obtain the optimal or near-optimal solutions within shorter execution times. Currently, these algorithms consider only one fragment per site. In real life, there is more than one fragment to be considered for a site. Replication is another issue to be dealt with in detail. In the future, we plan to extend the proposed algorithm for the replication and management of multiple fragments for a site. Additionally, the originating sites for the queries and their aspects can be considered to enhance the impact of the proposed algorithms.

REFERENCES


Umut Tosun received his B.E. degree from Izmir Institute of Technology, Izmir, Turkey in 2004, his M.E. degree from Bilkent University, Ankara, Turkey in 2007 and his Ph.D. from METU, Ankara, Turkey in 2013, all in Computer Engineering. Between 2004 and 2013, he worked for Bilkent University, ASEL-SAN Defence, Turkish Telecom, and Siemens Enterprise Communications. He is currently an Assistant Professor in the Department of Computer Engineering at Baskent University. His main interests are parallel and distributed databases, computer networks, and information security.