Knowl Inf Syst (2009) 20:349–373
DOI 10.1007/s10115-008-0182-y

REGULAR PAPER

A new ant colony optimization based algorithm for data allocation problem in distributed databases

Rosa Karimi Adl · Seyed Mohammad Taghi Rouhani Rankoohi

Received: 27 November 2007 / Revised: 29 July 2008 / Accepted: 27 September 2008 /
Published online: 23 January 2009

Abstract The Performance and the efficiency of a distributed database system depend highly on the way data are allocated to the sites. The NP-completeness of the data allocation problem and the large size of its real occurrence, call for employing a fast and scalable heuristic algorithm. In this paper, we address the data allocation problem in terms of minimizing two different types of data transmission across the network, i.e., data transmissions due to site-fragment dependencies and those caused by inter-fragment dependencies. We propose a new heuristic algorithm which is based on the ant colony optimization meta-heuristic, with regards to the applied strategies for query optimization and integrity enforcement. The goal is to design an efficient data allocation scheme to minimize the total transaction response time under memory capacity constraints of the sites. Experimental tests indicate that our algorithm is capable of producing near- optimal solutions within a reasonable time. The results also reveal the flexibility and scalability of the proposed algorithm.

Keywords Distributed database system · Non-replicated data allocation · Site-fragment dependency · Inter-fragment dependency · Ant colony optimization

1 Introduction

Distributed database applications have been the subject of increasing attention over the last few decades. Distributed database systems are not only more compatible with the decentralized nature of organizations and their growing volume of required data, but they also help reduce costs (communication and equipment), increase efficiency by providing a higher degree of parallelism, and improve reliability as well as accessibility [6]. Consequently, in addition to utilization of a multitude of computer systems, the necessity of increasing

R. Karimi Adl (✉) · S. M. T. Rouhani Rankoohi
Electrical and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran
e-mail: Rosa_adl@std.sbu.ac.ir; rositta_bonita@yahoo.com

S. M. T. Rouhani Rankoohi
e-mail: Rohani@sbu.ac.ir

parallelism and reliability have added to the complexities of data maintenance and management in a distributed database system. One of the fundamental and yet complicated problems in this area is the database design,[1] where in addition to the classic design stage for the global schema (using the same old techniques as in centralized databases) two new stages of fragmentation and allocation design should also be considered.

In order to come up with the appropriate allocation units at the distribution design stage, global relations are either decomposed into horizontal and vertical fragments, or a combination of both. The placement of these fragments in sites as well as their possible replication will be decided subsequently at the allocation stage. Although fragmentation and allocation are two interconnected tasks, they are usually performed separately to make it possible to "deal with the complexity of the problem" [26]. Without considering their allocation, one group of algorithms provides relatively suitable units of allocation by fragmenting the base relations. The other group of algorithms takes these fragments as inputs and work out the proper placement for each (allocation scheme).

Furthermore, not only the environmental limitations are considered by good allocation scheme (e.g., memory capacity, processing power of each site, capacity of communication channels, etc.), but it also places the fragments in sites in order to minimize the total response time of all transactions. In order to achieve this goal, the fragments are required to be allocated in such a way that firstly the volume of transferred data between sites would be low, secondly logically related fragments would be located in nearby sites, and finally the volume of maintained data in every site would be less than its memory capacity (environmental limitations permitted) [3]. In addition to these rules, other determining conditions such as the amount of provided parallelism should also be considered. Moreover, in order to choose an appropriate query optimization strategy some other environmental factors including the different processing power and disk drives' speed of each site, workload on servers and the network traffic [30] should be taken into account. Therefore, the data allocation problem can be categorized within resource allocation problems which are defined as "optimization problems with constraints" [18]. It is also possible to show that even without considering many other criteria, the allocation problem, is NP-hard and requires heuristic methods to be solved [3]. Therefore, in order to achieve a relatively suitable solution to the allocation problem within an acceptable computation time, the majority of the proposed algorithms (see Sect. 2), including the algorithm presented in this article, are only concerned with the network data transfer delay. This delay is the most significant contributing factor to transaction response time among the other factors which include the processing time of a local request, the delay of data transfer between local storage hierarchy, etc.

In practice, there are two main causes for data transmission between sites:

1. Execution of a transaction in a site which lacks the required data for the transaction.
2. Execution of transactions in one site which depending on the fragments it contains necessitates some data items to be transferred between at least two other sites. These transactions define some logical and semantic relationship between fragments.

There are some similarities between the data allocation problem (DAP), the file allocation problem (FAP), and the quadratic assignment problem (QAP): simply by considering only the first cause of data transmissions, the problem becomes similar to the well known FAP. However, the FAP differs from the DAP in many ways, the most important of which is the logical and semantic relationship among fragments (the second cause of data transmission) [6,25]. The logical and semantic relationships among fragments require the related fragments

[1] By the term 'distributed database' we mean relational distributed database.

to be located at nearby. This point has lead us to find similarities between the DAP and the QAP.

The QAP is a well-known combinatorial optimization problem first formulated by Koopmans and Beckman [16]. The problem can be defined as follows: Consider a set of n facilities and a set of n locations. The distance between each pair of locations is predefined. For each pair of facilities there is a specified flow in terms of the amount of supplies needed to be transported between the two facilities. The problem is to assign the facilities to different locations in order to minimize the sum of flows between each two facilities multiplied by the distance between their corresponding locations.

The dependence of fragments on each other can be expressed as a QAP, in which every fragment is equivalent to a facility and every site is equivalent to a location. The flow between two facilities represents the amount of data transmitted between two sites which contain the two corresponding fragments and the distance between two locations is considered as the cost of sending a data item (for example a block of data or a frame of data) between two sites. However, in most cases DAP is more complicated than QAP, because of the fact that in QAP the number of facilities and locations are equal, whereas in DAP there are usually more fragments than sites. Besides, unlike QAP, there are certain relationships between sites and fragments (the first reason of data transmission) in DAP, which should also be considered.

In this paper there is a solution for the allocation problem presented for the first time by combining some new heuristics with a number of ant colony optimization (ACO) algorithms proposed for the QAP.

The rest of this paper is organized as follows: in Sect. 2, some proposed algorithms for DAP are studied briefly. In Sect. 3, we defined some of the new concepts that are widely used in the rest of the paper. Section 4 refers to the detailed explanation of the DAP and the cost evaluation of an allocation scheme. Next, in Sect. 5, an introduction to ACO algorithms is provided. In Sect. 6, the QAP (which is very similar to the DAP) is defined briefly and the applied ACO algorithms are discussed. Sections 7 and 8 are devoted to our proposed ACO-DAP algorithm for DAP, which is explained in detail. In Sect. 9, an alternate algorithm known as simulated evolutionary (SE) algorithm is introduced with some modifications so that the proposed ACO-DAP algorithm could be compared with the revised version. Section 10 explains the experiment environment and describes the way test data are generated. The achieved results are further studied in Sect. 11. Finally, Sect. 12 provides a summary and conclusions.

2 Related work

As mentioned in advance, the DAP in distributed databases (DAP) is a more complicated case of the FAP. The FAP has been extensively studied in the previous literature [3,7,10,17]. The proposed algorithms gradually considered some of the requirements of the DAP which do not exist in the FAP. For example, various authors have considered the DAP as a form of the replica placement problem with different types of underlying networks such as content delivery networks and networks with read-one-write-all policy [5,8]. Mei et al. in [21] put emphasis on security considerations in their proposed data allocation process. Ram and Marsten [27] examined the problem from the concurrency mechanism point of view.

Existing algorithms for the DAP could define an allocation scheme either statically or dynamically. In a static data allocation algorithm, the data allocation scheme is designed based on the predefined execution pattern of transactions in the target environment. In contrast, dynamic data allocation algorithms [4,14,34] react to the changes in the execution

pattern of transactions. We will go through those static algorithms with goals similar to those of our suggested algorithm.

In 1982, Navathe et al. [24] proposed a data distribution algorithm in which data fragmentation and data allocation were determined simultaneously. However, most of the data allocation algorithms were assumed to have predefined data fragments and tried to allocate these fragments to proper sites (with or without replication). Ceri and Plagatti [6] proposed a greedy algorithm for both replicated and non-replicated data allocation design problem, however since the proposed algorithm was not successful in considering the logical and semantic relationship between fragments, the achieved results were not satisfactory.

In 1984, Bell [2] showed that the DAP is NP-Hard. Later, Freider and Sieglemann [13] proved the NP-Completeness of multi processor document allocation problem (MDAP) by reducing it to a well known NP-Hard problem: the QAP.

Consequently, most of the algorithms introduced afterward, tried to use heuristic methods to solve DAP. Although Sarathy et al. [29] and Menon [21] tried to present optimal approaches (based on mathematical programming) to the DAP, their algorithms suffered either from the high order time complexity or the complexity of the formulation itself.

In 1994, Corcoran and Hale [9] proposed a solution for DAP based on genetic algorithms (GA), in which the logical and semantic relation between fragments were not taken into account. Later, Frieder and Siegelmann [13] applied a different GA to the DAP which did consider the logical and semantic relationship between fragments, though in its simplest form using binary logic (i.e., the degree of dependencies were not mentioned) and without taking into account the important concept of dependencies between sites and fragments.

In 2002, Ahmad et al. [1] proposed another GA, a SE algorithm and an algorithm based on Mean Field Annealing [26] and showed that the SE algorithm provides better solutions in a relatively short time. Although the SE algorithm considers the logical and semantic relation between fragments, it apparently assumes all fragments to have the same size, which is often not the case in real-world DAPs.

In short, most of the existing static data allocation algorithms proposed will fall into one of these categories: greedy algorithms (e.g., [6,24]), optimal approaches (e.g., [21,29]), evolutionary and GAs (e.g., [1,9,13]), and mean field annealing (e.g., [1]). Some of the other algorithms cannot be placed in any particular category. In this paper, a static algorithm based on the ACO metaheuristic is presented for the first time for the DAP and hence a new category of ACO algorithms is introduced. Our proposed algorithm is further compared with the SE algorithm [1], which was previously compared in [1] with the genetic, mean field annealing as well as greedy algorithms and has been reported to find better solutions in a shorter period of time.

3 Preliminaries

Before going deep into our proposed solution for the DAP, it is necessary to define some of the concepts used in this work:

Definition 1 *direct transaction-fragment dependency* The dependency between transaction t_k and fragment f_j is said to be direct, if for each execution of the transaction t_k there should be some data transmitted from site containing the fragment f_j to the site executing the transaction.

Example 1.1 When transaction t_k is executed in site s_i, and requests a select operation on two fragments f_{j_1} and f_{j_2} saved in sites s_{i_1} and s_{i_2}, respectively, it is necessary for some data items to be transmitted from s_{i_1} and s_{i_2} to the site s_i.

Definition 2 *indirect transaction-fragment dependency* The dependency between transaction t_k and fragment f_j is said to be indirect, if for each execution of the transaction t_k there should be some data transmitted from the site containing the fragment f_j to the site storing one of the other fragments (not the originating site of the transaction).

This type of dependency is usually defined according to the strategies used for query optimization, integrity constraint checking, security support, etc. The query optimization problem itself has been the subject of several researches. With the ever growing complexity of queries in database systems, the proposed query optimization methods have become more essential and intricate. One good illustration is suggested in [35], which is based on identifying and analyzing similar subqueries.

Example 2.1 Assuming transaction t_k is executed in site s_i, and requests a join operation on fragments f_{j_1} and f_{j_2}; a query optimization strategy is to send joinable data from the site containing smaller fragment (e.g., f_{j_1}) to the site containing the larger one (e.g., f_{j_2}) and then send the result to the site s_i. Thus, for each execution of transaction t_k (in any site), it is necessary to send some data items from the site containing f_{j_1} to the site having f_{j_2}.

Example 2.2 When transaction t_k inserts some tuples into fragment f_j, a vertical fragment of a global relation R, consistency issues urge the primary key or the tuple identifier (TID) of the inserted tuples to be sent to the other site containing the vertical fragments of R.

Example 2.3 When transaction t_k requests an update operation on attribute values involved in a calculated attribute (existing in another fragment) the updated data should be transmitted from the site containing the updated data to the site containing the calculated attribute.

Definition 3 *site-fragment dependency* The site s_i is said to be dependent on fragment f_j, if it executes at least one transaction which directly depends on the fragment f_j. The more frequently the transactionis executed at the site or the more data is required by it, the more dependent the site would be on the fragment.

Example 3.1 Suppose the scenario defined in Example 1, the execution of transaction t_k in site s_i, causes the site s_i dependent on fragments f_{j_1} and f_{j_2}.

Definition 4 *Inter-fragment dependency* The fragment f_{j_1} is said to be dependent on the fragment f_{j_2}, if there exists at least one transaction such as t_k which indirectly depends on fragment f_{j_1} and its execution necessitates data transmission from the site containing this fragment to the one containing fragment f_{j_2}.[2]

Example 4.1 Suppose the scenarios defined in Examples 2.1 and 2.2 for every execution of transaction t_k (in every site), it is necessary for some data items to be transmitted from the site containing f_{j_1} to the site storing f_{j_2}. In this way the fragment f_{j_1} is said to be dependent on the fragment f_{j_2}.

It can be seen easily that one of the factors making fragments dependent on each other is the enforcement of integrity constraints [15]. However, as far as we know, this matter has not been studied in the previous literature.

[2] The original idea of the dependencies between fragments was introduced in [1]. In that reference however, binary operations (such as join, union, etc.) were introduced as the only reason for this type of dependency, whereas we believe that other operations, specially the enforcement of integrity constraints, can also cause this kind of dependency.

4 Problem definition

4.1 Data allocation in distributed database systems

A distributed database consists of more than one site, each maintaining a portion of the global database. Different transactions with different execution frequencies are submitted to different sites, which may cause data transfer between the sites in the network (due to site-fragment and/or inter-fragment dependencies). The problem is to minimize the response time of each transaction considering the storage capacity constraint of each site. Since most of the delays in a distributed database system are related to the required time (cost) for data transmission, we have only considered this cost in determining the overall response times of transactions.

4.2 Pertinent parameters

Table 1 summarizes the key notations used in this paper.

Parameters associated with the sites:

Suppose the distributed database system is composed of n sites. The ith site ($1 \leq i \leq n$) is denoted as s_i and its storage capacity (expressed in bytes) as SiteCap_i. The sites are connected to each other in a network with predefined topology. The distance between each two sites is estimated according to this topology and the average cost of sending a unit data item from one site to another can be calculated. After defining the cost of unit data transmission between each two sites, a $UC_{n\times n}$ matrix can be defined in which $uc_{i_1 i_2}$ shows the cost of sending a unit data item from site s_{i_1} to the site s_{i_2}.

Parameters associated with the fragments:

If the global relations are decomposed into m fragments the jth fragment ($1 \leq j \leq m$) is denoted as f_j and its size (in average[3]) as fragSize_j.

Parameters associated with the transactions:

The determining factor in placement of fragments in sites, are the access and the execution patterns of transactions. Since the number of transactions in a typical environment may be large, only 20% of the most active transactions (that do 80% of data accesses in the system) may be taken into account [25]. If the number of such transactions in the environment is l the kth transaction ($1 \leq k \leq l$) is denoted as t_k. The frequencies of transaction executions in sites are shown with matrix $\text{FREQ}_{n\times l}$ in which freq_{ik} ($1 \leq i \leq n$, $1 \leq k \leq l$) is the execution frequency of transaction t_k at site s_i.

Parameters associated with the transaction dependencies on the fragments:

The direct transaction-fragment dependency (explained in Sect. 3) is shown with matrix $\text{TRFR}_{l\times m}$, in which trfr_{kj} indicates the volume of data items of fragment f_j that must be sent from site containing f_j to the site executing transaction t_k, for each execution of t_k.

[3] The size of the fragments may vary due to insert, update or delete operations during the life cycle of the corresponding distributed database system. Thus the volume is considered in average.

Table 1 Description of notations

Symbol	Description
n	The number of sites
s_i	The ith site
SiteCap_i	The storage capacity of site s_i
$UC_{n\times n}$	The matrix denoting the cost of unit data transmission between each two sites
$uc_{i_1 i_2}$	The cost of sending a unit data item from site s_{i_1} to the site s_{i_2}
m	The number of fragments
f_j	The jth fragment
fragSize_j	The size of fragment f_j
l	The number of considered transactions
t_k	The kth transaction
$\text{FREQ}_{n\times l}$	The matrix denoting the execution frequency of each transaction in each site
freq_{ik}	The execution frequency of transaction t_k in site s_i
$\text{TRFR}_{l\times m}$	The matrix denoting the direct transaction-fragment dependency
trfr_{kj}	The volume of data items of fragment f_j that must be sent from site containing f_j to the site executing transaction t_k, for each execution of t_k
$Q_{l\times m\times m}$	The matrix denoting the indirect transaction-fragment dependency
$q_{kj_1 j_2}$	The volume of data items that must be sent from site containing fragment f_{j_1} to the site storing f_{j_2}, for each execution of transaction t_k
Ψ	The m-element vector which denotes an allocation scheme
ψ_j	The site to which fragment f_j is assigned in the allocation scheme Ψ
$\text{COST}(\Psi)$	The cost of data transmission in an allocation scheme Ψ
$\text{COST1}(\Psi)$	The cost of data transmission in an allocation scheme Ψresulting from direct transaction-fragment dependencies
$\text{COST2}(\Psi)$	The cost of data transmission in an allocation scheme Ψ resulting from indirect transaction-fragment dependencies
$\text{STFR}_{n\times m}$	The matrix denoting the site-fragment dependency
stfr_{ij}	The volume of data items from fragment f_j which are accessed by site s_i in unit time (according to the site-fragment dependency)
$\text{PARTIALCOST1}_{nxm}$	The matrix denoting the $\text{COST1}(\Psi)$ incurred by allocating each fragment to each site
partialcost1_{ij}	The cost incurred by f_j allocated to site s_i as a result of direct transaction-fragment dependency
$\text{QFR}_{l\times m\times m}$	The matrix denoting the indirect transaction-fragment dependency taking the execution frequencies of the transactions into account
$\text{qfr}_{kj_1 j_2}$	The volume of data needed to be sent from site storing fragment f_{j_1} to the site having fragment f_{j_2} in unit time taking into account the transaction frequency of t_k.
$\text{FRDEP}_{m\times m}$	The matrix denoting the inter-fragment dependency
$\text{frdep}_{j_1 j_2}$	The volume of data items needed to be sent from site having fragment f_{j_1} to the site having fragment f_{j_2} in unit time due to the indirect transaction-fragment dependency

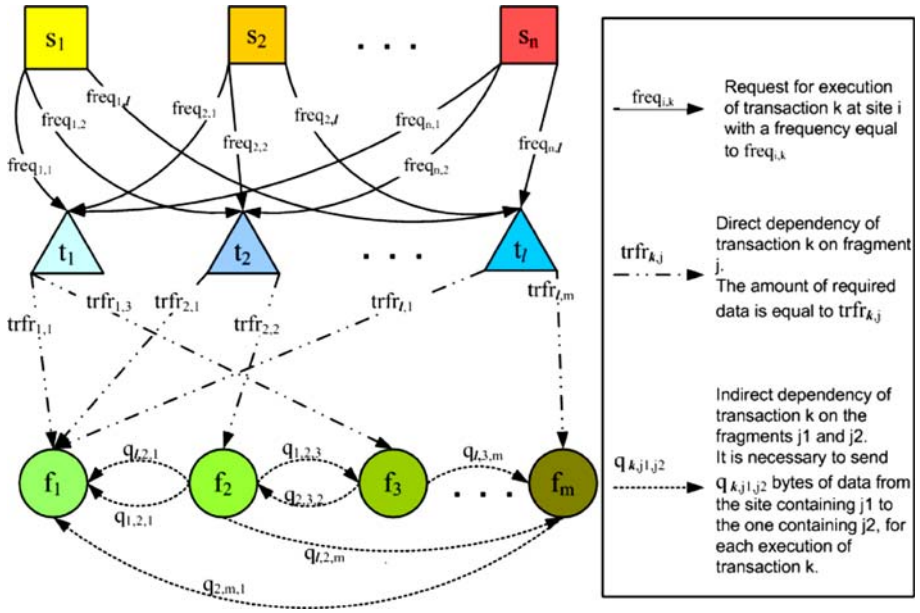


Fig. 1 The dependencies of transaction on fragment along with the dependencies of sites on transaction. Having these two types of dependencies, the dependency of sites to fragments can be inferred

One the other hand, the indirect transaction-fragment dependency is shown by a three dimension matrix $Q_{l\times m\times m}$ in which $q_{kj_1j_2}$ indicates the volume of data that must be sent from site containing fragment f_{j_1} to the site storing f_{j_2}, for each execution of transaction t_k.

Regarding these two kinds of dependency between transactions and fragments, and the dependencies of sites on the transactions, a proper allocation can be defined. In Fig. 1 the dependency between sites, transactions and fragments is shown.

4.3 Cost and constraints evaluation

As stated earlier, a proper allocation is one which minimizes the costs while considering the environmental constraints. Here, storage capacity is the only environmental constraint considered and because the most important cost and time delay is that of data transmission over the network, only the network data transmission cost is considered.

Assume an allocation scheme shown by the m-element vector Ψ in which ψ_j specifies the site to which f_j is allocated. Variable x_{ij} $(1 \le i \le n, 1 \le j \le m)$ could be defined as follows:

$$x_{ij} = \begin{cases} 1 & \text{if } \psi_j = s_i \\ 0 & \text{Otherwise} \end{cases}$$

Now the storage capacity constraint can be expressed as follows:

$$\sum_{j=1}^{m} \text{fragSize}_j \times x_{ij} \le \text{siteCap}_i \quad i = 1, \ldots, n$$

The cost of data transmission in an allocation scheme Ψ includes COST1 and COST2:

$$\text{COST}(\Psi) = \text{COST1}(\Psi) + \text{COST2}(\Psi)$$

COST1 illustrates the cost resulting from direct transaction-fragment dependencies, whereas COST2 is the cost of data transmission between sites due to indirect transaction-fragment dependencies.

To calculate COST1 the amount of site-fragment dependencies must be calculated. This amount is expressed by matrix $\text{STFR}_{n\times m}$ in which stfr_{ij} indicates the volume of data items from fragment f_j that are accessed by site s_i in unit time. This matrix is defined as:

$$\text{STFR}_{n\times m} = \text{FREQ}_{n\times l} \times \text{TRFR}_{l\times m}$$

Or:

$$\text{stfr}_{ij} = \sum_{k=1}^{l} \text{freq}_{ik} \times \text{trfr}_{kj}$$

From this matrix, $\text{PARTIALCOST1}_{n\times m}$ can be calculated in which partialcost1_{ij} is the cost of storing fragment f_j in site s_i incurred by direct transaction-fragment dependencies. This matrix can be calculated as follows:

$$\text{PARTIALCOST1}_{n\times m} = UC_{n\times n} \times \text{STFR}_{n\times m}$$

In other words:

$$\text{partialcost1}_{ij} = \sum_{q=1}^{n} uc_{iq} \times \text{stfr}_{qj}$$

Having matrix $\text{PARTIALCOST1}_{n\times m}$, the cost COST1 for an allocation scheme Ψ can be calculated from:

$$\text{COST1}(\Psi) = \sum_{j=1}^{m} \text{partialcost1}_{\psi_j j}$$

To calculate COST2 it is necessary to define inter-fragment dependencies. First, according to the matrix $Q_{l\times m\times m}$, the matrix $\text{QFR}_{l\times m\times m}$ is built, in which $\text{qfr}_{kj_1j_2}$ shows the volume of desired data to be sent from the site storing fragment f_{j_1} to the site containing fragment f_{j_2} in unit time taking the transaction frequency of t_k into account. The elements of this matrix are calculated as:

$$\text{qfr}_{kj_1j_2} = q_{kj_1j_2} \times \sum_{r=1}^{n} \text{freq}_{kr}$$

Using matrix $\text{QFR}_{l\times m\times m}$, matrix $\text{FRDEP}_{m\times m}$ can be determined. In this matrix, element $\text{frdep}_{j_1j_2}$ Shows the volume of data sent from site having fragment f_{j_1} to the site having fragment f_{j_2} in unit time due to the indirect transaction-fragment dependencies (according to all transactions and their execution frequencies). The elements of this matrix are calculated as:

$$\text{frdep}_{j_1j_2} = \sum_{k=1}^{l} \text{qfr}_{kj_1j_2}$$

Now, according to matrix $\text{FRDEP}_{m\times m}$, COST2 of an allocation scheme, Ψ can be found:

$$\text{COST2}\,(\Psi) = \sum_{j_1=1}^{m}\sum_{j_2=1}^{m} \text{frdep}_{j_1 j_2} \times uc_{\psi_{j_1}\psi_{j_2}}$$

As explained in Sect. 4, our proposed algorithm has some steps and in each step, the desirability of the achieved allocation scheme is calculated based on its COST using the formulae mentioned above.

5 General concepts of ACO algorithms

Ant Colony Algorithms are used for solving many complicated problems such as routing [11], assignment [19,20,31], scheduling [23], etc. These algorithms adopt the behavior of ants in the real world. Ant colonies are naturally simple distributed but organized systems, and thus appropriate to be mimicked in order to explore many distributed control and optimization problems.

A group of ant algorithms that are based on ACO meta-heuristic are known as ACO algorithms [12]. These algorithms are used to solve discrete optimization problems and here we applied an algorithm of this type to the DAP.

6 The quadratic assignment problem (QAP)

The QAP is an NP-Complete problem [28] to which ACO algorithms are applied with considerable success [32]. This problem can be defined formally as follows:

Assume n facilities with $B_{n\times n} = \left[b_{ij}\right]$ defined as the flow between facilities i, j and n locations with $A_{n\times n} = \left[a_{ij}\right]$ standing for the distance between locations i, j; the objective is to find an allocation scheme Ψ in which each facility is assigned to exactly one location so that the objective function $f(\Psi)$ is minimized. The objective function is defined with the formula below:

$$f(\Psi) = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} a_{\psi_i \psi_j}$$

Here ψ_i gives the location of facility i in the current allocation scheme Ψ.

Amongst proposed heuristic approaches, there have been some ACO algorithms suggested for the QAP, such as AS [20], $\mu\mu$ AS [31] and ANTS [19].

One can observe easily that the problem of minimizing COST2 (stated in Sect. 4.3) is very similar to the QAP. Thus we have adopted some concepts from QAP in solving ACO algorithms. More precisely, we applied a heuristic similar to the one proposed in AS algorithm and some other methods used in the $\mu\mu$AS algorithm such as "lower pheromone trail limit" and randomized the order of assignments. We have also applied a 2-opt local search procedure which is used in most of the proposed ACO algorithms for the QAP.

7 The proposed ACO algorithm for DAP

In the proposed ACO algorithm for the DAP, as the ACO meta-heuristic suggests, each pair of $\left(s_i, f_j\right)$ (couplings of sites and fragments) with $1 \le i \le n$ and $1 \le j \le m$ is associated

with a pheromone trail τ_{ij} and a heuristic value η_{ij}. The τ_{ij} is updated by each ant after each iteration and shows the probability (desirability) of fragment f_j being assigned to the site s_i in a near optimal (or optimal) assignment. On the other hand, η_{ij}, which is constant through all iterations, is the heuristic desirability of assigning fragment f_j to the site s_i. This heuristic desirability is computed based on the PARTIALCOST1$_{n\times m}$matrix and a coupling matrix which will be discussed later in this section.

The generic ACO algorithm for the DAP is described below (this algorithm is designed based on ACO metaheuristic):

Ant colony optimization algortithm for DAP

```
 1. Initialize all pheromone trails with τ0
 2. Calculate the heuristic desirability matrix η
 3. WHILE no_of_iterations < MAXIMUM_ITERATIONS
 4.     WHILE no_of_ants < ANTS_POPULATION
 5.       Sort the fragments randomly
 6.       Create a feasible allocation scheme probabilistically
 7.       Improve the assignment with local search
 8.       Calculate the cost of improved assignment
 9.       no_of_ants = no_of_ants + 1
10.     END WHILE
11.     Evaporate pheromone trails
12.     Update pheromone trails
13.     Enforce the minimum pheromone trail to be τ0
14.     no_of_iterations = no_of_iterations + 1
15. END WHILE
16. Output the best solution found so far
```

8 Description of our proposed ACO-DAP algorithm

8.1 Initialization

At the first step, we assign an equal amount of pheromone trail τ_0 to each pair of (s_i, f_j). Afterwards, the heuristic desirability should be calculated in order to be used in probabilistic assignment of fragments to the sites. The heuristic desirability of assigning fragment f_j to site s_i, η_{ij} is calculated as follows:

$$\eta_{ij} = \eta 1_{ij}^{\delta} \times \eta 2_{ij}^{\omega}$$

Where $\eta 1$ is the heuristic desirability corresponding to COST1. Inversely, the term $\eta 2$ refers to the heuristic desirability which is calculated according to COST2. Parameters δ and ω give weights to these two costs. For example, in an environment in which COST2 is much heavier than COST1, the designer can simply adjust ω value to be more than δ. The value $\eta 1_{\mathrm{ij}}$ is computed as:

$$\eta 1_{ij} = \frac{\max_{1\le p\le n}\left(\text{partialcost1}_{pj}\right)}{\text{partialcost1}_{ij}}$$

The computation of $\eta 2_{ij}$ is a generalization of the heuristic proposed in AS_QAP [20]. We define four vectors d_1, d_2, f_1 and f_2 which are calculated with the following formulas:

$$d_{1_i} = \sum_{p=1}^{n} uc_{ip} \quad i = 1, \ldots, n$$

$$d_{2_i} = \sum_{p=1}^{n} uc_{pi} \quad i = 1, \ldots, n$$

$$f_{1_j} = \sum_{p=1}^{m} \text{frdep}_{jp} \quad j = 1, \ldots, m$$

$$f_{2_j} = \sum_{p=1}^{m} \text{frdep}_{pj} \quad j = 1, \ldots, m$$

The lower d_{1_i} is, the more central is considered the site s_i, when the data is to be transmitted from site s_i to the other sites. Inversely, the lower d_{2_i} is, the more central the site s_i is considered, when the data is to be transmitted from the other sites to the site s_i. On the other hand, the higher f_{1_j} is, the more data should be sent from the site containing fragment f_j to the sites containing other fragments and the higher f_{2_j} is, the more data should be emitted from other sites to the site containing the fragment f_j.

In the next step, two coupling matrixes $E1$ and $E2$ are calculated as:

$$E1 = d_1^T \times f_1$$
$$E2 = d_2^T \times f_2$$

Where $e_{1_{ij}} = d_{1_i} \times f_{1j}$ and $e_{2_{ij}} = d_{2_i} \times f_{2j}$. Then we make the main coupling matrix E by simply adding $E1$ and $E2$:

$$E = E1 + E2$$

In fact, if the cost of transmitting a unit data item between each pair of sites is independent of the direction of transmission (i.e., $\forall i, j\ 1 \le i \le n, 1 \le j \le m, uc_{ij} = uc_{ji}$) we would simply define $FRDP'$ matrix as follows:

$$\forall j_1, j_2\ 1 \le j_1 \le m, 1 \le j_2 \le m$$
$$\text{frdep}'_{j_1 j_2} = \text{frdep}'_{j_2 j_1} = \text{frdep}_{j_1 j_2} + \text{frdep}_{j_2 j_1}$$

And compute vectors d and f with the following formula:

$$d_i = \sum_{p=1}^{n} uc_{ip}$$

$$f_j = \sum_{p=1}^{m} \text{frdep}'_{jp}$$

Then the coupling matrix $E = d^T \times f$ is calculated.

Now that the coupling matrix E is made (calculated either by the first formulas or by the second), the heuristic desirability $\eta 2_{ij}$ is defined as:

$$\eta 2_{ij} = \frac{1}{e_{ij}}$$

8.2 Ants' solutions construction

Similar to what is suggested in AS-QAP, a solution is constructed as follows:

For each ant in each iteration, fragments are sorted in a random order. At each step, the ant (ant_k) assigns the next yet unassigned fragment f_j to a site s_i which have still enough memory to store f_j. The assignment of fragment f_j to a site s_i is done with a probability given by (this formulation is based on what AS-QAP suggests):

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\gamma_1}.[\eta_{ij}]^{\gamma_2}}{\sum_{l \in N_j^k} [\tau_{lj}(t)]^{\gamma_1}.[\eta_{lj}]^{\gamma_2}}, & \text{if } i \in N_j^k \\ 0, & \text{otherwise} \end{cases}$$

Where τ_{ij} (t) is the pheromone trail associated with the couple (s_i, f_j) at iteration t.γ_1 and γ_2 are parameters determining the relative importance of pheromone trails and the heuristic information. The term N_j^k shows the sites still having enough storage capacity to save fragment f_j. In fact, the closer a fragment f_j is to the end of the fragment list, the fewer sites N_j^k does contain.

8.3 Local search

After constructing a solution, each ant tries to improve its solution by applying a local search. This local search has two phases: Exchange and Change.

8.3.1 Exchange phase

In the Exchange phase an ant follows these steps:

1. Set the_best_exchange_benefit = 0
2. If the terminating condition is not satisfied yet, chose next fragment f_j (or the first fragment if it is the first time being at step 2) from a randomly sorted fragment list.
3. Examine the benefit of exchanging the location of fragment f_j with the location of other fragments.[4] Find the fragment f^* which provides the biggest positive benefit if exchanged with f_j.
4. If such a fragment f^* exists and the benefit of exchanging the location of f_j with the location of f^* is more than the_best_exchange_benefit (the benefit of previously accomplished exchange), go to step 5, Otherwise go to step 2.
5. Exchange the location of fragment f_j with f^* and set the_best_exchange_benefit to the benefit of exchanging the location of f_j with the location of f^*. Go to step2.

The terminating condition is satisfied when either all of the fragments are chosen from the list or the number of accomplished exchanges equals to a parameter EX_Max. This parameter enables the designers to have trade-off between algorithm computation time and the optimality of the solution. It ranges between 0, where no exchanging local search is applied, and m, where all the fragments are chosen from the list.

Using the variable 'the_best_exchange_benefit' adds some random behavior to the local search method and reduces the possibility of frequent reaching a local optimum.

The benefit of exchanging the location of two fragments f_{j_1} and f_{j_2} in an allocation scheme Ψ is computed as follows:

$$\text{EXbenefit}(\Psi, j_1, j_2) = \text{EXbenefit1}(\Psi, j_1, j_2) + \text{EXbenefit2}(\Psi, j_1, j_2)$$

[4] Only those exchanges which do not threat the site capacity constraint are examined.

Where EXbenefit1(Ψ, j_1, j_2) stands for the benefit achieved according to COST1 and is calculated as follows:

$$\begin{aligned}\text{EXbenefit1}(\Psi, j_1, j_2) = {} & \text{partialcost1}_{\psi_{j_1}\, j_1} + \text{partialcost1}_{\psi_{j_2}\, j_2} \\ & - \text{partialcost1}_{\psi_{j_2}\, j_1} - \text{partialcost1}_{\psi_{j_1}\, j_2}\end{aligned}$$

On the other hand, the EXbenefit2(Ψ, j_1, j_2) is the benefit gained with respect to COST2. This benefit is computed using the following equation [33]:

$$\begin{aligned}&\text{EXbenefit2}(\Psi, j_1, j_2) \\ &= \text{frdep}_{j_1 j_2}\left(uc_{\psi_{j_1}\psi_{j_2}} - uc_{\psi_{j_2}\psi_{j_1}}\right) + \text{frdep}_{j_2 j_1}\left(uc_{\psi_{j_2}\psi_{j_1}} - uc_{\psi_{j_1}\psi_{j_2}}\right) \\ &\quad + \sum_{\substack{p=1 \\ p \neq j_1, j_2}}^{m} \left(\begin{array}{l} \text{frdep}_{pj_1}\left(uc_{\psi_p\psi_{j_1}} - uc_{\psi_p\psi_{j_2}}\right) + \text{frdep}_{j_1 p}\left(uc_{\psi_{j_1}\psi_p} - uc_{\psi_{j_2}\psi_p}\right) \\ + \text{frdep}_{pj_2}\left(uc_{\psi_p\psi_{j_2}} - uc_{\psi_p\psi_{j_1}}\right) + \text{frdep}_{j_2 p}\left(uc_{\psi_{j_2}\psi_p} - uc_{\psi_{j_1}\psi_p}\right) \end{array}\right)\end{aligned}$$

8.3.2 Change phase

The change phase is very similar to the Exchange. This phase consists of the following steps:

1. If the terminating condition is not satisfied yet, choose the next (or the first) fragment, f_j from a randomly sorted fragment list.
2. Examine the benefit of moving fragment f_j to all other feasible sites (sites which have still enough memory capacity to store f_j). Find the site s^* which provides the biggest positive benefit if f_j moves to it.
3. If such s^* site exists, change the location of fragment f_j to be stored at site s^*.
4. Go to step 1.

The terminating condition is satisfied when either all of the fragments are chosen from the list or the number of changes exceeds the value of the parameter CH_Max. The reason of using CH_Max is identical to the one explained for EX_Max. Here we do not use a variable such as 'the_best_change_found'. This is because changing the location of a single fragment has less impact on the total environment and it seems that the possibility of getting stuck in a local optimum of change phase is less than exchange one.

The benefit of moving fragment f_j to the site s_i in the allocation scheme Ψ is calculated by the following formula:

$$\text{CHbenefit}(\Psi, j, i) = \text{CHbenefit1}(\Psi, j, i) + \text{CHbenefit2}(\Psi, j, i)$$

Where CHbenefit1(Ψ, j, i) is the change benefit due to the COST1 and CHbenefit2(Ψ, j, i) is the change benefit according to the COST2. These benefits can be inferred from the formulas of EXbenefit. The procedure of moving fragment f_j to site s_i could be assumed as a special case of exchange procedure in which the second fragment is a virtual fragment residing in site s_i. A virtual fragment f_v could be defined as a fragment with the following features:

1) $\text{fragsize}_{f_v} = 0$
2) $\forall k\ 1 \leq k \leq l\ \ \text{trfr}_{k\, f_v} = 0$ and thus $\forall i\ 1 \leq i \leq n\ \ \text{partialcost1}_{i\, f_v} = 0$
3) $\forall k, j\ 1 \leq k \leq l, 1 \leq j \leq m\ \ Q_{k\, j\, f_v} = Q_{k\, f_v\, j} = 0$

And thus $\forall j \;\; 1 \le j \le m \;\; \text{frdep}_{j\ f_v} = \text{frdep}_{f_v\ j} = 0$

With these properties defined for f_v and assuming that $\psi_{f_v} = s*$, we can now calculate the CHbenefit1 as follows:

$$\begin{aligned}\text{CHbenefit1}\,(\Psi, j, i) &= \text{EXbenefit1}\,(\Psi, j, f_v)\\ &= \text{partialcost1}_{\psi_j\ j} + \text{partialcost1}_{\psi_{f_v}\ f_v}\\ &\quad -\text{partialcost1}_{\psi_{f_v}\ j} - \text{partialcost1}_{\psi_j\ f_v}\end{aligned}$$

Referring to the property2, $\text{partialcost1}_{\psi_{f_v}\ f_v} = \text{partialcost1}_{\psi_j\ f_v} = 0$

And we have:

$$\text{CHbenefit1}(\Psi, j, i) = \text{partialcost1}_{\psi_j\ j} - \text{partialcost1}_{i\ j}$$

On the other hand, CHbenefit2 is computed as:

$$\begin{aligned}\text{CHbenefit2}(\Psi, j, i) &= \text{EXbenefit2}(\Psi, j, f_v)\\ &= \text{frdep}_{j\ f_v}\left(uc_{\psi_j\psi_{f_v}} - uc_{\psi_{f_v}\psi_j}\right) + \text{frdep}_{f_v\ j}\left(uc_{\psi_{f_v}\psi_j} - uc_{\psi_j\psi_{f_v}}\right)\\ &\quad + \sum_{\substack{p=1\\ p\neq j}}^{m}\begin{pmatrix}\text{frdep}_{pj}\left(uc_{\psi_p\psi_j} - uc_{\psi_p\psi_{f_v}}\right) + \text{frdep}_{jp}\left(uc_{\psi_j\psi_p} - uc_{\psi_{f_v}\psi_p}\right)\\ +\text{frdep}_{p\ f_v}\left(uc_{\psi_p\psi_{f_v}} - uc_{\psi_p\psi_j}\right) + \text{frdep}_{f_v\ p}\left(uc_{\psi_{f_v}\psi_p} - uc_{\psi_j\psi_p}\right)\end{pmatrix}\end{aligned}$$

As property 3 implies:

$$\text{frdep}_{j\ f_v} = \text{frdep}_{f_v\ j} = \text{frdep}_{p\ f_v} = \text{frdep}_{f_v\ p} = 0$$

Therefore we have the following formula for CHbenefit2:

$$\begin{aligned}&\text{CHbenefit2}(\Psi, j, i)\\ &= \sum_{\substack{p=1\\ p\neq j}}^{m}\left(\text{frdep}_{p\ j}\left(uc_{\psi_p\psi_j} - uc_{\psi_p\ i}\right) + \text{frdep}_{j\ p}\left(uc_{\psi_j\ \psi_p} - uc_{i\ \psi_p}\right)\right)\end{aligned}$$

8.4 Pheromone updating

By the end of each iteration, all ants will have found a feasible solution that is improved with the local search mechanism. After calculating the costs of each allocation, some pheromone trails are evaporated to avoid unlimited accumulation of trails and allow the algorithm to forget previously made improper choices [32]. Then, all ants should leave some pheromone trails according to their achieved solution and its corresponding cost. Thus the total amount of accumulated pheromone trail is updated as [20]:

$$\tau_{ij}(t+1) = \rho.\tau_{ij}\,(t) + \sum_{k=1}^{\text{no\_of\_ants}} \Delta\tau_{ij}^{k}$$

Where $(1-\rho)$, with $0 < \rho < 1$, represents the evaporation and $\Delta\tau_{ij}^{k}$ is the amount of pheromone that ant k leaves on the coupling $\left(s_i, f_j\right)$. The computation of $\Delta\tau_{ij}^{k}$ can be done as follows [20]:

$$\Delta\tau_{ij}^{k} = \begin{cases}\frac{Q}{\text{COST}(\Psi^k)}, & \text{if } \psi_j^k = s_i\\ 0, & \text{otherwise}\end{cases}$$

where Q is a parameter representing the amount of pheromone deposited by an ant.

Updating the pheromone trails in this manner may cause the pheromone trail on some edges to become less than a lower bound τ_0 and thus avoid some couples (s_i, f_j) to be selected in the next iterations. So, at this step all pheromone trails are checked and revised with the following formula:

$$\tau_{ij}(t+1) = \begin{cases} \tau_0 & \text{if } \tau_{ij}(t+1) < \tau_0 \\ \tau_{ij}(t+1) & \text{otherwise} \end{cases}$$

Preparing the pheromone trails, we can now go to the next iteration. The best result achieved through all of the iterations is printed at the end of the algorithm.

9 The alternative algorithm for comparison

In order to evaluate the performance of our algorithm, we have chosen a modified version of the SE data allocation algorithm [1] to be compared with our proposed method. This algorithm has been reported to outperform other algorithms such as genetic data allocation, mean field annealing, and random search algorithms in a reasonable time.

9.1 Simulated evolutionary algorithm (original version)

The generic SE data allocation algorithm is as follows [1]:

Simulated Evolutionary Data Allocation Algorithm

```
(1)  Construct the first chromosome based on the problem data and perturb this
     chromosome to generate an initial population
(2)  Use the mapping heuristic to generate a solution for each chromosome
(3)  Evaluate the solutions obtained
(4)  No_of_generation s = 0
(5)  WHILE no_of_generations < MAX_GENERATION DO
(6)       Select chromosomes for next population
(7)       Perform crossover and mutation for these set of chromosomes
(8)       Use the mapping heuristic to generate solution for each
(9)         chromosome
(10)      Evaluate the solutions obtained
(11)      no_of_generation s = no_of_generation s + 1
(12) ENDWHILE
(13) Output the best solution found so far
```

The chromosome structure is defined as follows:

The number of genes in part a is equal to the total allocation limit and this part specifies the storage capacity limit. On the other hand, each gene in part b corresponds to a fragment specifying the priority of that fragment to be considered in the allocation procedure.

In part a each gene is designed to be a single bit. A value of 1 indicates that the corresponding allocation space is allowed to be used for this chromosome; otherwise the space could not be used. It must be checked whether the new effective allocation limit is enough for all fragments to be allocated. In the original definition of the algorithm this is simply done

by counting the number of 1s in part a and checking that this sum is greater than or equal to the total number of fragments.

After the creation of the first generation, the mapping heuristic is applied to extract the allocation scheme proposed by each chromosome. This mapping heuristic is explained in detail in [1].

At the end of each generation, the fitness of each chromosome is determined according to the computed cost of its achieved solution. For the next generation, the parents are selected with a probability proportional to their fitness. Performing the crossover and mutation procedures on each pair of parents yields two new children.

The process of creating a new generation and evaluating their corresponding allocations is done until the number of generations reaches a maximum limit. Then the best result achieved through all of the generations is given as the output of the algorithm.

9.2 Modified SE algorithm

Since the original definition of the SE algorithm [1] is not intended for a general situation, we have added and modified some parts in order to make the algorithm capable of dealing with our different test environments. The modified or added parts are listed below:

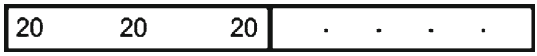
1. The original definition of this algorithm states that "the number of genes in part a is equal to the total allocation limit". This definition is ambiguous. Thus we have set the number of genes in part a to be equal to the number of sites.
2. In the original algorithm, each gene in part a is designed to be a single bit but the binary logic is only applicable in the situation where all fragments are of the same size and each site only saves one fragment. However, it can easily be noticed that the explained environment is too rare in reality. So we have changed this part of the algorithm and defined each gene ga_i in part a to have a value selected from the range $\left[0, \text{siteCap}_i\right]$. This value indicates the free space on site s_i in the corresponding chromosome.
3. With the modified definition of part a, the first chromosome is made with all genes in part a set to the site capacity of their corresponding sites. For the remaining chromosomes in the initial population, the value for each gene ga_i in part a is chosen randomly from $\left[0, \text{siteCap}_i\right]$.
4. The process of checking whether the new effective allocation limit is enough for all fragments to be allocated cannot be done in the way suggested by the original version of the algorithm. The reason is that the method mentioned above cannot be applied to the environments with heterogeneous fragment sizes. Thus we have modified the checking method and defined a *probably feasible chromosome* as one which obeys the following restriction:

$$\sum_{k=1}^{m} \text{fragSize}_k \leq \sum_{k=1}^{n} ga_k$$

If this criterion does not hold true for a chromosome, then the chromosome is obviously not a feasible one. But we should notice that this constraint does not guarantee a feasible chromosome. For example, suppose an environment with two fragments: f_1 and f_2 which should be allocated to three sites: s_1, s_2 and s_3. If we have the following condition:

$$\text{fragSize}_1 = \text{fragSize}_2 = 30$$

A chromosome such as the one shown below, despite its inability to violate the stated constraint, is obviously an infeasible one.



In our implementation of the SE data allocation algorithm, for each chromosome we first examine the previously stated condition and if it is violated, we reset all genes in part a to be identical to the values of part a of the first chromosome; otherwise we do not change it at this step. Later in the mapping phase, where an allocation is determined based on a chromosome, if a fragment is found at any step that cannot be assigned to any of the sites due to insufficient storage capacity, a random gene ga_i in part a is selected and its value is set to siteCap_i. The mapping procedure is then restarted.

5. As the representation of part a of the chromosomes has been modified, the mutation method should be changed as well. In the original version of the algorithm, mutation in part a sets the value of a selected gene to either 1 or 0. Here, we have defined the mutation process on part a of a chromosome to select a gene ga_i randomly and set its value to a random value chosen from $[0, \text{siteCap}_i]$.

Using these modifications, we have generated some test data and compared the modified SE algorithm with our proposed one.

10 Experiment environment

To evaluate the proposed ACO-DAP algorithm we tested it through a number of experiments. In each experiment, we let one parameter vary while fixing others. Our test data generation process follows some structured rules which are explained in Sect. 10–2.

10.1 The hardware/software configuration

The experiments are all done in an environment using a 1.86 GHz Intel Pentium M 750 processor with 1 Giga bytes of DDR2 RAM with Microsoft Windows XP Professional SP2 as the operating system. The ACO-DAP and SE algorithms were implemented in the programming environment provided by MATLAB 2006. The algorithms are then tested by the same test data generated by the rules which are defined in the next section.

10.2 Test data generation

To compare our algorithm with the SE algorithm mentioned above, we have implemented a test data generator which gets number of fragments m, number of sites n and some other parameters (which will be defined later) as input and creates a random DAP instance as follows:

Fragment sizes

For each fragment f_j the fragment size fragSize_j is chosen randomly[5] from $\left[\frac{c}{10}, \frac{20\times c}{10}\right]$. Where c is an input parameter defined as:

[5] All of our random selections are based on a uniform random distribution.

$$c \in A = \{x | x \in N \wedge 10 \leq x \leq 1{,}000\}$$

In fact the c parameter is an approximation of the average of the fragment sizes.

Site capacities

We have designed our test data generator to create a rigorous site capacity constraint where site capacities are chosen to be very low. However, we have followed a specific strategy to create a situation in which for every fragment (even the last one) there would exist a site that still has enough memory capacity for storage. This condition should hold true regardless of the order in which fragments are considered to be allocated.

In this strategy, we first assume to have m fragments all being the same size of the largest fragment. Then for each site s_i a random number p_i is chosen which shows the number of the fragments mentioned above (big size fragments) which can be saved in it. The p_i value is chosen randomly from $\left[1, 2 \times \frac{m}{n} - 1\right]$. Note that $p_i \in N$. However, we want to have a somehow rigorous site capacity constraint in which $\sum_{i=1}^{n} p_i = m$. Thus whenever we assign p_i to a site s_i, we should calculate the number of remaining fragments (fragments without any space to be stored in) as follows:

$$rf_i = m - \sum_{q=1}^{i} p_q$$

We should make sure that the number of remaining sites (sites, s_r without any p_r defined over yet) is less than rf_i.

As long as this condition does not hold, we choose another random value for p_i as mentioned before. Notice that for the last site s_n, we directly set the value of p_n using the following formula:

$$P_n = m - \sum_{q=1}^{n-1} p_q$$

Now that every site s_i is associated with a p_i, the value for siteCap_i is determined as follows:

$$\text{siteCap}_i = p_i \times \max_{1 \leq j \leq m} \left(\text{fragSize}_j\right)$$

Transmission costs:

The test data generator gets the parameter UCN (Unit transmission Cost between two Neighboring sites) as an input and randomly generates the value for $UC_{i_1\ i_2}$ from [UCN, $n \times$ UCN], where n is the number of sites.

Site-fragment dependency:

To generate the STFR matrix, transaction frequencies and transaction-fragment dependency should be defined first. The test data generator receives an input parameter RPT $(0 < \text{RPT} \leq 1)$ which shows the probability of a transaction being requested at a site. Using this parameter, a list of sites at which a transaction t_k is requested is defined probabilistically. Then for each site s_i in this list, the frequency of requesting transaction t_k (freq_{ik}) is determined through a uniformly distributed random value in the range [1, 1,000]. For any other site s_r that does not require t_k, the value for freq_{ik} is simply set to zero.

Similarly, for the transaction-fragment dependency, the test data generator receives another input parameter APF $(0 < \text{APF} \leq 1)$, which denotes the probability of a fragment being accessed by a transaction. With this parameter, a list of fragments being accessed by a transaction t_k is defined probabilistically and for each fragment f_j in this list, the transaction-fragment dependency trfr_{kj} is chosen from a uniformly distributed random value in $[0, \text{fragSize}_j]$. For those fragments f_r that are not in the t_k's accessing fragments list, the value of trfr_{kr} is set to zero.

Having $\text{FREQ}_{n\times l}$ and $\text{TRFR}_{l\times m}$ matrices, the site-fragment dependency matrix $\text{STFR}_{n\times m}$ is calculated as:

$$\text{STFR}_{n\times m} = \text{FREQ}_{n\times l} \times \text{TRFR}_{l\times m}$$

Inter-fragment dependency:

To generate the $\text{FRDEP}_{m\times m}$ matrix, the test data generator requires a parameter APFS which denotes the probability of a certain transaction causing some data transmission from the location of a specific fragment to the location of another given fragment. Increasing APFS yields more affinity and cohesion between fragments and intensifies the effect of COST2 on the final COST.

Using APFS parameter, for each pair of fragments (f_{j_1}, f_{j_2}), a list of transactions $T^{j_1\ j_2}$ which urge data transmission from the site containing f_{j_1} to the site containing f_{j_2} is defined probabilistically. Then, for each transaction t_k in $T^{j_1\ j_2}$ the value of $q_{kj_1j_2}$ is chosen randomly from $[0, \text{fragSize}_{j_1}]$. For other transactions t_r where $t_r \notin T^{j_1\ j_2}$, the value for $q_{r\ j_1\ j_2}$ is set to zero. Having generated the $Q_{l\times m\times m}$ matrix, the $\text{QFR}_{l\times m\times m}$ matrix and thus $\text{FRDEP}_{m\times m}$ are calculated using the equations mentioned in Sect. 4.3.

11 Experimental results

As stated earlier, we have compared our algorithm with a modified version of the SE data allocation algorithm. The comparison is done according to 210 different configurations of the DAP. In each problem instance, the number of fragments, m, and the number of sites, n, are chosen from [3, 50] with the constraint that n should be less than or equal to m. 20 different values for the number of fragments and sites are selected from [3, 50], which yields 210 problem configurations. The comparison criteria are the solution cost as well as the algorithm's running time, and the ultimate goal is to observe the effect of the size of the problem on the quality of achieved solutions.

For each test, the number of fragments and sites are defined and the test data generator is used to create a problem instance. We have fixed other input parameters of the data generator to the values shown in Table 2.

The values for the probability parameters RPT and APF are chosen to be as close as possible to those in real problems. We have experimentally examined different values for the APFS parameter and chose the value which usually results in solutions with nearly equal values for COST1 and COST2.

The maximum number of iterations (or generations in the SE algorithm) is set to 200 for all algorithms. Increasing this value may yield better solutions but since the goal of the experiments was to conduct a comparison, this value was deemed to be sufficient.

We have tested three versions of our ACO data allocation algorithm against the SE algorithm:

Table 2 Selected values for the parameters

Parameter description	Parameter name	Value
Approximation of the average fragment size	c	10
Unit transmission cost between two neighbor sites	UCN	1
Number of transactions	l	20
Probability of a transaction being requested at a site	RPT	0.7
Probability of a fragment being accessed by a transaction	APF	0.4
Probability of a transaction necessitates data transmission between two sites (other than the originating site)	APFS	0.025

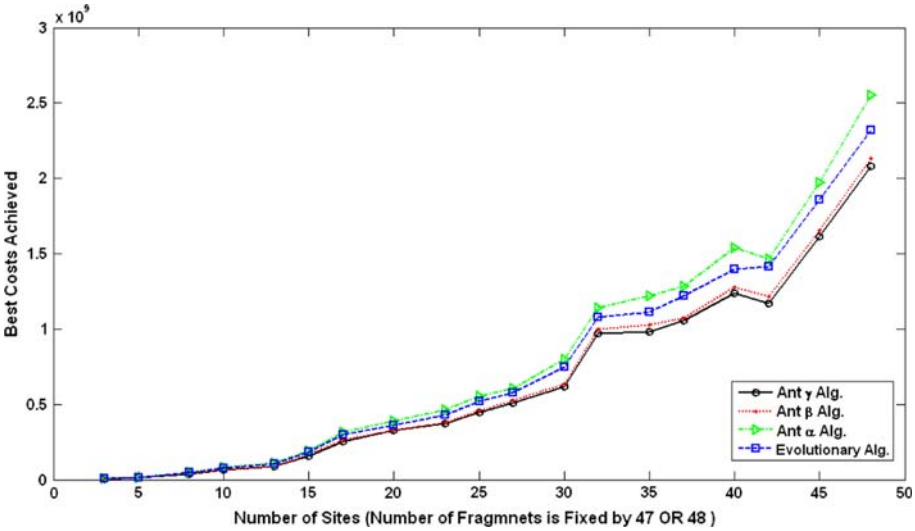


Fig. 2 Evaluating the results achieved by the algorithms in a type1 comparison

The α **version**: In this version we set CH_Max and EX_Max to zero which means no usage of the local search.

The β **version:** In this version we have CH_Max $= m$ and EX_Max $= 3$ which causes a slightly lightweight local search.

The γ **version:** Finally in this version we set CH_Max $=$ EX_Max $= m$ and thus used the maximum power of our local search method.

The results are organized into two types of diagrams: in type1 we have fixed the number of fragments and tested the algorithms with variable numbers of sites. On the other hand, in type2 the number of sites is fixed and the number of fragments varies. Thus a total number of 40 diagrams are generated from which we have selected the following four diagrams (Figs. 2, 3, 4, 5).

According to the above diagrams (Figs. 2, 3, 4, 5) in most cases compared to the SE algorithm the proposed ACO-DAP algorithm (using local search procedure) provides better results. This superiority increases as the number of sites or fragments increases, which implies that the ACO data allocation algorithm is more scalable than the alternate SE data allocation algorithm. However, the higher quality solutions appear only at the cost of higher time

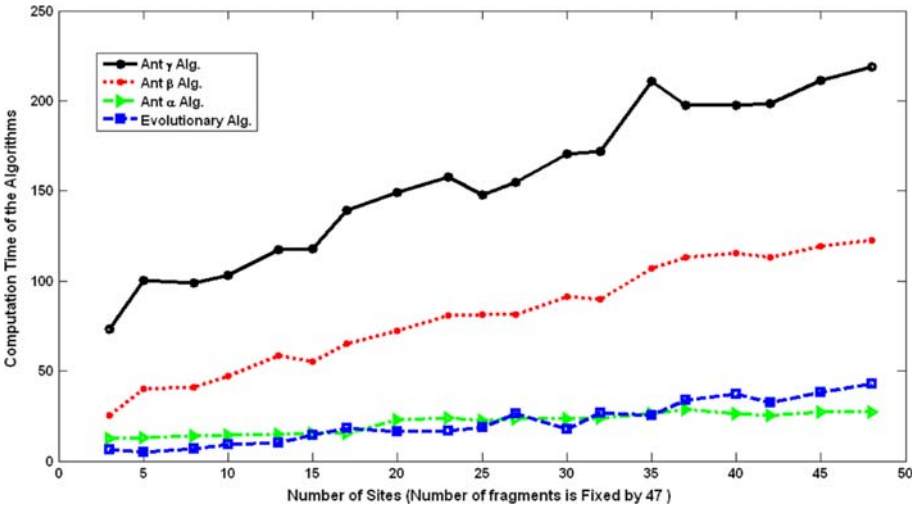


Fig. 3 Evaluating the computation time of the algorithms in a type1 comparison

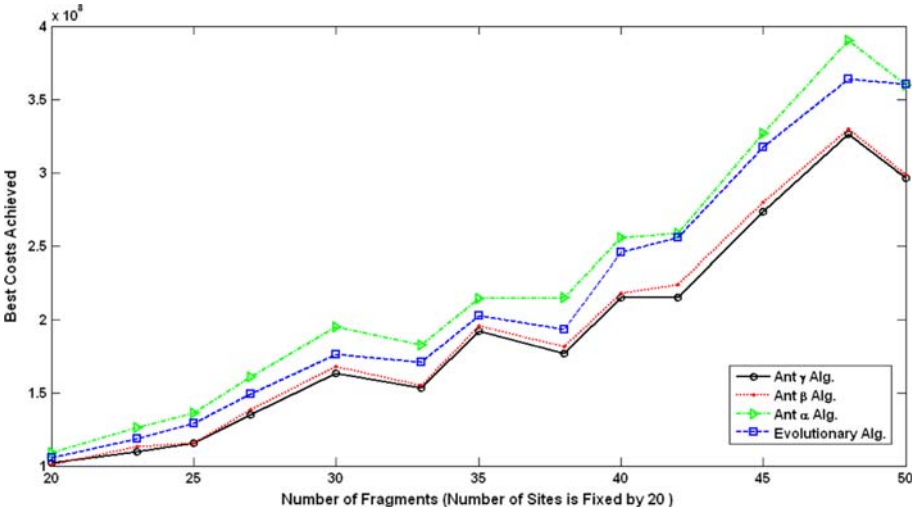


Fig. 4 Evaluating the results achieved by the algorithms in a type2 comparison

complexity of our algorithm. In order to address this problem the defined CH_Max and EX_Max parameters help the designer to achieve a desirable tradeoff between solution quality and algorithm's running time. Consequently, the merit of our proposed ACO-DAP algorithm can be distinguished with its high quality solutions, scalability and flexibility.

By further analyzing the diagrams, it can be seen that the γ version of the ACO data allocation algorithm outperforms the others. However, this algorithm is the most time consuming version. Thus γ version is recommended whenever the quality of the solution is the main concern and not the computation time. One should note that in most cases these computations are done occasionally and the results are used for a long period of time. On the other hand, in

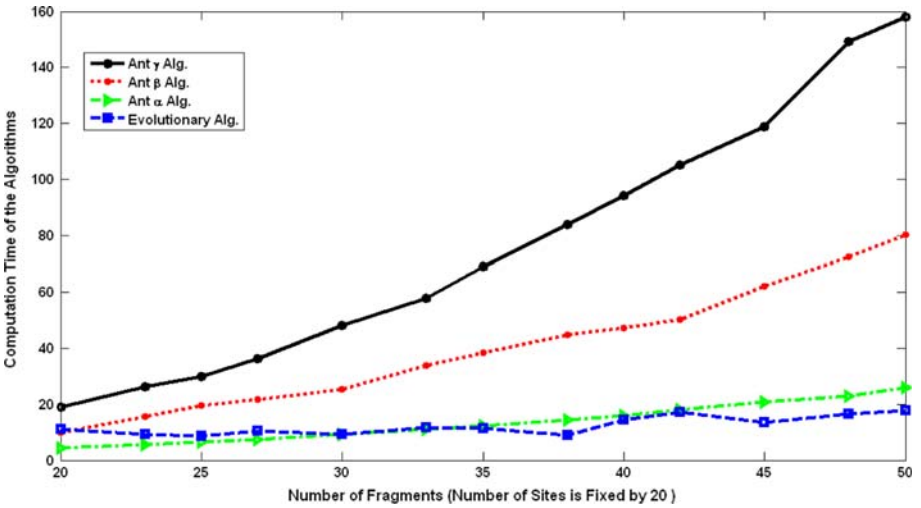


Fig. 5 Evaluating the computation time of the algorithms in a type2 comparison

a shorter period of time the β version provides better solutions than the SE algorithm (notice how close is the β curve to the γ). Therefore if the main concern is the computation time, one can use customized values for the CH_Max and EX_Max in order to achieve a desirable time complexity and solution quality tradeoff. Finally, the α version as it can be seen in the diagrams, gives the worst solution among the four curves. This fact reveals one of the merits of applying local search in the algorithm.

12 Conclusion

In this paper, we addressed prominent issues of non-replicated data allocation in distributed database systems with memory capacity constraint. We took query optimization and integrity enforcement mechanisms in the formulation of the DAP into consideration. We also proposed a new data allocation algorithm called ACO-DAP which had been defined based on ACO meta-heuristics and combined it with a local search procedure.

The main contribution of this work is the clear definition provided for the DAP which not only considers the query optimization and integrity enforcement mechanism but also exploits the ACO concepts to solve the problem.

In our experimental studies, we have evaluated and compared the optimality of solutions and the execution time of three different versions of the ACO-DAP algorithm with that of a revised version of one of the most successful algorithms in this field known as SE algorithm [1]. The results prove the high scalability of the proposed algorithm and we found the ACO-DAP algorithm to have superior transaction response times to SE in DAPs of various sizes especially large ones. However, depending on the amount of applied local search, the time complexity of different versions of the ACO-DAP algorithm may become worse than that of the SE algorithm.

The impact and the influence of the local search mechanism which exist on the ACO-DAP algorithm, on the overall results and consequently on the computation time, have been studied in several experiments. Our experimental results demonstrated the merits of using the local

search procedure in achieving appropriate data allocation schemes. These results also show the flexibility of our algorithm which provides a diverse range of solution quality and time complexity trade-offs.

In the future, we plan to extend our algorithm to provide solutions for the DAP with replication. Additionally, other plausible direct extensions would be completing the problem formulation, expansion of our experiments in order to cover a larger variety of values for other parameters and rearrangement of algorithms for DAP based on other meta-heuristics such as Particle Swarm Optimization or Neural Networks.

References

1. Ahmad I, Karlapalem K, Kwok YK et al (2002) Evolutionary algorithms for allocating data in distributed database systems. Int J Distrib Parallel Databases 11(1):5–32. doi:10.1023/A:1013324605452
2. Bell DA (1984) Difficult data placement problems. Comput J 27(4):315–320
3. Bell D, Grimson J (1992) Distributed database systems. Addison-Wesley Longman Publishing Co., Inc, Boston
4. Brunstrom A, Leutenegger ST, Simha R (1995) Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workloads. ICASE: Institute for Computer Applications in Science and Engineering
5. Buchholz S, Buchholz T (2004) Replica placement in adaptive content distribution networks. In: SAC '04: proceedings of the 2004 ACM symposium on applied computing, Nicosia, pp 1705–1710
6. Ceri S, Pelagatti G (1984) Distributed databases principles and systems. McGraw-Hill, Inc., New York
7. Chu WW (1969) Optimal file allocation in a multiple computer system. IEEE Trans Comput 18(10):885–889
8. Cook SA, Pachl JK, Pressman IS (2002) The optimal location of replicas in a network using a READ-ONE-WRITE-ALL policy. Distrib Comput 15(1):57–66
9. Corcoran AL, Hale J (1994) 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
10. Daellenbach HG, George JA, McNickle DC (1983) Introduction to operations research techniques (2nd edn). Allyn and Bacon, Boston
11. Di Caro G, Dorigo M (1998) An adaptive multi-agent routing algorithm inspired by ants behavior. In: Proceedings of PART98-5th annual Australasian conference on parallel and real-time systems, Singapore, pp 261–272
12. Dorigo M, Stutzle T (2004) Ant colony optimization. MIT Press, Cambridge
13. Frieder O, Siegelmann HT (1997) Multiprocessor document allocation: A genetic algorithm approach. IEEE Trans Knowl Data Eng 9(4):640–642
14. Gu X, Lin W (2006) Practically realizable efficient data allocation and replication strategies for distributed databases with buffer constraints. IEEE Trans Parallel Distrib Syst 17(9):1001–1013
15. Ibrahim H (2005) Checking integrity constraints in a distributed database. Encyclopedia of database technologies and applications, pp 66–73
16. Koopmans TC, Beckmann MJ (1957) Assignment problems and the location of economics activities. Econometrica 25:53–76
17. Laning LJ, Leonard MS (1983) File allocation in a distributed computer communication network. IEEE Trans Comput 32(3):232–244
18. Lee Z, Su S, Lee C et al (2003) A heuristic genetic algorithm for solving resource allocation problems. Knowl Inf Syst 5(4):503–511
19. Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. Inf J Comput 11(4):358–369
20. Maniezzo V, Colorni A (1999) The ant system applied to the quadratic assignment problem. IEEE Trans Knowl Data Eng 11(5):769–778
21. Mei A, Mancini LV, Jajodia S (2003) Secure dynamic fragment and replica allocation in large-scale distributed file systems. IEEE Trans Parallel Distrib Syst 14(9):885–896
22. Menon S (2005) Allocating fragments in distributed databases. IEEE Trans Parallel Distrib Syst 16(7):577–585
23. Merkle D, Middendorf M (2003) Ant colony optimization with global pheromone evaluation for scheduling a single machine. Appl Intell 18(1):105–111

24. Navathe S, Ceri S, Wiederhold G et al (1984) Vertical partitioning algorithms for database design. ACM Trans Database Syst 9(4):680–710
25. Ozsu T, Valduriez P (1999) Principles of distributed database systems, 2nd edition
26. Peterson C, Soderberg B (1989) A new method for mapping optimization problems onto neural networks. Int J Neural Syst 1(1):3–22
27. Ram S, Marsten RE (1991) A model for database allocation incorporating a concurrency control mechanism. IEEE Trans Knowl Data Eng 3(3):389–395
28. Sahni S, Gonzalez T (1976) P-complete approximation problems. J ACM 23(3):555–565
29. Sarathy R, Shetty B, Sen A (1997) A constrained nonlinear 0–1 program for data allocation. Eur J Oper Res 102(3):626–647
30. Shahabi C, Khan L, McLeod D (2000) A probe-based technique to optimize join queries in distributed internet databases. Knowl Inf Syst 2(3):373–385
31. Stutzle T (1997) MAX-MIN ant system for the quadratic assignment problem. In: Technical report AIDA-97-4, FG Intellectik, FB Informatik, TU Darmstadt
32. Stutzle T, Dorigo M (1999) ACO algorithms for the quadratic assignment problem, pp 33–50
33. Taillard E (1995) Comparison of iterative searches for the quadratic assignment problem. Location Sci 3:87–105
34. Ulus T, Uysal M (2003) Heuristic approach to dynamic data allocation in distributed database systems. Pakistan J Inform Technol 2(3):231–239
35. Zhu Q, Tao Y, Zuzarte C (2005) Optimizing complex queries based on similarities of subqueries. Knowl Inf Syst 8(3):350–373

Author Biographies

Rosa Karimi Adl Received the B.S. and M.S. degrees in Computer Engineering-Software from Shahid Beheshti University, Iran in 2005 and 2007, respectively. She is currently a Ph.D. student under Prof. Keneth Barker's supervision at the department of Computer Science at the University of Calgary. Her research interests include distributed database systems, artificial intelligence, and privacy preserving data repositories.

Seyed Mohammad Taghi Rouhani Rankoohi received his bachelor's degree in mathematics from Tehran University, Bachelor's and Master's degree in informatics, and the D.E.S.S degree in Teleinformatics from Pierre and Marie Curie University in Paris. He is an assistant professor at Shahid Beheshti University (SBU). He has mostly lectured on File Engineering and Databases at SBU and other universities such as Sharif University of Technology and Tehran University. His publications include several papers published in Iranian journals and conference proceedings, four translations of textbooks into Farsi, as well as seven authored books, all of which are widely used as textbooks in Iran. He has received the "Book of the Year" award in 1994 and 2003 .Two of his books have also been honored as the "Selected Academic Book of the Year" by Tehran University in 1992 and 2002. His research interests include Databases Systems, Web-DBMS integration and File Engineering. Reading modern literature is among his hobbies.