Knowl Inf Syst (2012) 30:435–455
DOI 10.1007/s10115-011-0384-6

REGULAR PAPER

A new vertical fragmentation algorithm based on ant collective behavior in distributed database systems

Mehdi Goli · Seyed Mohammad Taghi Rouhani Rankoohi

Received: 25 January 2010 / Revised: 7 October 2010 / Accepted: 28 January 2011 /
Published online: 24 February 2011

Abstract Considering the existing massive volumes of data processed nowadays and the distributed nature of many organizations, there is no doubt how vital the need is for distributed database systems. In such systems, the response time to a transaction or a query is highly affected by the distribution design of the database system, particularly its methods for fragmentation, replication, and allocation data. According to the relevant literature, from the two approaches to fragmentation, namely horizontal and vertical fragmentation, the latter requires the use of heuristic methods due to it being NP-Hard. Currently, there are a number of different methods of providing vertical fragmentation, which normally introduce a relatively high computational complexity or do not yield optimal results, particularly for large-scale problems. In this paper, because of their distributed and scalable nature, we apply swarm intelligence algorithms to present an algorithm for finding a solution to vertical fragmentation problem, which is optimal in most cases. In our proposed algorithm, the relations are tried to be fragmented in such a way so as not only to make transaction processing at each site as much localized as possible, but also to reduce the costs of operations. Moreover, we report on the experimental results of comparing our algorithm with several other similar algorithms to show that ours outperforms the other algorithms and is able to generate a better solution in terms of the optimality of results and computational complexity.

Keywords Distributed database design · Data fragmentation · Vertical fragmentation · Optimal fragment numbers · Minimum transaction access cost · Ant clustering algorithm

M. Goli (✉) · S. M. T. Rouhani Rankoohi
Electrical & Computer Engineering Department,
Shahid Beheshti University, Tehran, Iran
e-mail: m.goli@mail.sbu.ac.ir

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

1 Introduction

Due to day technological advancement in hardware, software, and computer networks, as well as the decentralization of organizations, which sometimes are even distributed across several continents, the needs of the clients for data processing systems have increased to the extent that in many cases, centralized systems cannot satisfy the number of demands anymore. Hence, the tendency toward using distributed systems builds up over time [5].

Design of distributed database systems is basically a multi-criteria optimization problem with several inter-related problems such as data fragmentation and data allocation. Considering a range of existing approaches to each of these problems, design of distributed database systems is highly complicated and usually provides a theoretical justification for the use of heuristic methods. In order to design distributed database systems, a combination of operations is required, which is both in the design of centralized database systems and distribution-allocation aspects [8]. The distribution-allocation aspects include data acquisition, data replication, and most importantly data fragmentation, and data allocation [6].

The data fragmentation problem consists of decomposing a relation or set of relations into several fragments in such a way that different user applications preferably run on at most one fragment. Data fragmentation is generally designed in two ways: horizontal fragmentation and vertical fragmentation. There is also a mixed fragmentation, which is an integration of both. Among these, the vertical fragmentation problem has the highest complexity in terms of being NP-Hard [18]. Therefore, this paper sets out to tackle this problem. There are many algorithms that solve this problem, which are mainly split into two categories: attribute-based fragmentation and transaction-based fragmentation [9]. In the first category, attribute provides the underlying basis of fragmentation; while in the second, transaction is the basis of fragmentation. There are more advantages to the second category: Firstly, it is more consequential to use the transaction as the basis of fragmentation. More importantly, as the number of attributes increases in the first category, the computational costs grow exponentially. Since this problem is avoided in the second category, we use transaction-based approach to fragment a relation [9].

Due to the high complexity of the vertical fragmentation problem, we need to resort to heuristic algorithms in order to solve it. Most different types of existing heuristic algorithms in vertical fragmentation are greedy ones, and only recently, stochastic algorithms such as genetic algorithms have been widely used. One of the most significant advantages of stochastic algorithms is their scalability. In particular, one class of stochastic algorithms, which has not yet been addressed to solve the vertical fragmentation problem, is a class of algorithms based on ant collective behavior. Being an important property for the vertical fragmentation problem, scalability makes ant-behavior-based algorithms the most desirable candidates to solve vertical fragmentation problem.

There are some similarities between the data clustering problem and the vertical fragmentation problem provided that the clusters are considered as regions of the attribute pattern space; similarities between data clustering problem and vertical fragmentation problem should be easily traceable. Regions, with more dense patterns, form clusters, which are separated by regions of lower density. This motivates us to take advantage of these algorithms. The data clustering problem lies in the fact that the organization of the data into clusters should be based on their similarities in such a way that the similarities among the data in each cluster exceed those of the other data. However, an important difference distinguishes vertical fragmentation problem from data clustering. One of the most important issues in vertical fragmentation problem is to find an optimum number of fragments in order to decrease the access cost of transactions; whereas in data clustering algorithms, the number of clusters is

usually fixed [8]. Therefore, we need to apply some changes so as to adapt these algorithms to the data fragmentation problem.

In this paper, we present an algorithm for finding a solution to vertical fragmentation problems which is optimal in most cases, in terms of a mixture of ant clustering algorithms used in data clustering and new heuristic methods. In order to minimize the costs of accessing the required attributes by means of transactions, our proposed algorithm fragments a relation to an extent that each transaction uses one fragment or as few fragments as possible, to access the required attributes.

The remainder of the paper is structured as follows. Section 2 takes a quick glance at the existing vertical fragmentation algorithms. Through the rest of this paper, a number of new concepts has been presented, which are subsequently defined in Sect. 3. Section 4 presents a formal definition of vertical fragmentation and a cost function, followed by a brief description of ant clustering as well as data clustering problem (which is similar to vertical fragmentation problem) in Sects. 5 and 6, respectively. Sections 7 and 8 are devoted to our proposed algorithms, and Sect. 9 refers to an alternative algorithm called GSGR-GA [15] with which our algorithms have been compared. Section 10 evaluates our algorithm both theoretically and experimentally. Experimental environment as well as methods of producing test data is also explained in this section. Finally, the conclusion is presented in Sect. 11.

2 Previous works

At this stage, we briefly explain the earlier related works on vertical fragmentation, which are also referred to as vertical partitioning or attribute partitioning. There is a wide variety of applications for vertical partitioning, namely, in all settings in which the performance of a system is affected by the interaction between data and transaction [34]. This had been first encountered in file optimization problem, which has been thoroughly studied in the literature [2,3,12,27,31,42] and then gradually started to appear in the database management systems so as to enhance the performance of transactions.

Our study concerns the work pioneered by Stocker and Dearnley [13,44] in which they discuss the implementation of a self-reorganizing database management system that carries out attribute clustering and show that attribute partitioning is beneficial where the storage costs are comparably lower than the cost of accessing data. In order to create clusters that minimize the cost of accessing data, Kennedy [26] employs mathematical models to present an optimal approach to attribute partitioning through computation of the cost of accessing data, which exploit transactions. Later, Hoffer and Severance [24] define a measure of pairwise affinity between any pair of attributes, based on which partitioning is carried out via Bond Energy Algorithm [32]. One of the disadvantages of their method is that it leaves the actual task of clustering for the designer. In [23], Hoffer introduces another method, where by using non-linear, zero-one programing, the linear combination of storage, retrieval, and update costs can be minimized. Yet, following another approach, Eisner and Severance [16] decompose each record into two record segments, i.e., primary and secondary record segments. First, they compute the cost of access, transfer, and storage of attributes as a non-linear function of the costs of each segment and then try to minimize it—a task which can be very costly. This model is extended in [31] by adding a blocking factor to the segments and taking advantage of programing techniques. Apart from suffering the disadvantages of the previous model, their work also suffers from lack of an accurate cost of access. Later, partitioning problem of hierarchal databases was addressed by Schkolnic [41], where the access time for a given pattern had been minimized.

In 1979, being a combinatorial problem, vertical partitioning proved to be NP-Hard in most cases [18]. Therefore, all the previous algorithms for which an optimal solution had been proposed, proved to suffer from high computational complexity. Hence, they were decided to be unsuitable particularly for large-scale problems. Subsequently, most of the algorithms that were introduced attempted to make use of heuristic methods to solve the vertical partitioning problem, examples of which are as follows. Hammer and Niamir [19] presented a heuristic greedy algorithm that offers near-optimal solutions based on attribute characteristics and usage pattern of the file. Navathe et al. [33] presented a binary iteration algorithm, in each of its iterations. There are two fragments that optimize a cost function to fit a specific application environment. By including the number, length, and selectivity of attributes and the cardinality of the relation in computing the cost function, Cornell and Yu [10] extended their algorithms, in an attempt to reduce the number of disk accesses in the optimal solution. Ceri et al. [7] proposed a divide and conquer approach, where a division was considered as an algorithm in [33] and introduced a conquering tool, which uses a detailed cost model. Navathe et al. also presented a graphical approach in which all fragments are consecutively generated after each iteration; hence, the complexity is reduced [34].

Thus far, a general objective function that could evaluate the "goodness" of the partitions, obtained as the result of an algorithm, has been absent. In 1993, Chakravarthyt et al, presented a formal method for evaluating the results of vertical partitioning algorithms by introducing such an objective function called partition evaluator (PE) [8].

As stated before, the existing algorithms for vertical fragmentation have been classified into attribute-based fragmentation and transaction-based algorithms. The algorithms mentioned so far were all grouped in attribute-based class. In 1993, by using branch and bound method, Chu et al. [9] presented a transaction-based optimal binary partitioning algorithm to minimize the number of accesses to the disk. In another attempt, Pérez et al. [37] were able to offer a heuristic simulated annealing with threshold accepting algorithm, which could simultaneously carry out data fragmentation along with fragment allocation within their related sites. Their objective was to minimize the cost of transferring, accessing, migrating, and storing fragments in sites. A genetic algorithm was developed by Song et al. [43], which generated optimal partitioning by finding an optimum transactions access path to attributes. In [15], Du et al. suggested a fragmentation algorithm that applied grouping- based genetic algorithms [17] and took advantage of the PE given by [8] in order to find an optimum solution.

In this paper, a transaction-based algorithm based on ant clustering meta-heuristic is presented to handle vertical fragmentation problem. Our algorithms adopt PE as their objective function to be further compared with the GRGS-GA algorithm [15], which has been already differentiated from genetic and greedy algorithms [15] and reported to have obtained optimum solutions, mainly for large-scale problems. Through categorizing vertical fragmentation algorithms based on the algorithm type e.g. greedy algorithms, optimal approaches etc., we have introduced a new category of data fragmentation, namely vertical fragmentation algorithms based on ant behavior.

3 Preliminaries

This section introduces some of the basic notations, concepts, and definitions that are used in this paper.

3.1 Pertinent parameters considered

Table 1 summarizes the key notations that are used in this paper.

Table 1 Notation description

Notations	Meaning
N_A	Arity of a relation (number of attributes in a relation)
N_T	Number of important transactions in a system
N_F	Number of generated fragments
N_{F_i}	ith fragment ($1 \leq i \leq N_F$)
A_i	ith attribute vector ($1 \leq i \leq N_A$)
T_k	kth transaction ($1 \leq k \leq N_T$)
X_{ik}	Binary variable to determine whether attribute A_i is accessed by transaction T_K
f_k	Frequency of transaction
aik	kth component of attribute A_i
$P_{A_i T_k}$	Priority of access of transaction T_K to attribute A_i
$M_{A_i T_k}$	Measure of need for transaction T_k to attribute A_i
$D_{A,B}$	Measure of dissimilarity between attributes A and B
n_{F_i}	Number of attributes in fragment N_{F_i}
$\|R_{itk}\|$	Number of relevant attributes which fragment k can access remotely with respect to fragment i by transaction t
$\|S_{it}\|$	Number of attributes in fragment i that transaction t accesses
AUM	Matrix denoting access patterns of transactions to attributes

3.1.1 Fragment-related parameters

Assuming that relations are decomposed into N_F fragments, ith fragment ($1 \leq i \leq N_F$) is denoted by N_{Fi}, and number of its attributes is denoted by n_{Fi}.

3.1.2 Transaction-related parameters

A determining factor in assigning attributes to fragments is the access patterns of transactions to attribute. In a typical environment, there is a high number of transactions running. Therefore, only the important transactions, i.e. 20% of active transactions, which perform 80% of the total access to data, are taken into account [30]. For N_T number of such transactions in the environment, kth transaction ($1 \leq k \leq N_T$) is denoted by T_k and frequency of its execution is shown with f_k. These transactions make a N_T-dimensional space called the transaction space, in which each transaction T_k represents a dimension. Accesses to attributes by transactions are represented by $N_A \times N_T$ binary matrix AUM, where the value 0 shows that for a component x_{ij}, transaction j does not need to access attribute i; however, value 1 shows the opposite. The attributes that are accessed by transaction T_k are called its relevant attributes, and the fragment that contains the most relevant attributes is called the local fragment for T_k [8]. Number of relevant attributes of T_k which are located in its local fragment i is shown by $|S_{it}|$, and number of those that are not located in their local fragment i and should be accessed remotely by transaction T_k is shown with $|R_{ikt}|$.

3.1.3 Attribute-related parameters

Assuming that each relation that needs to be fragmented has N_A attribute, ith attribute is denoted by $A_i (1 \leq i \leq N_A)$. In the transaction space, each attribute is considered as a vector

so that the more the priority of a transaction, the closer its relevant attributes are. Therefore, attributes that are located near each other are the ones which are accessed together by some transactions and have a high chance of appearing in a fragment together.

3.2 Basic concepts

In the following section, the basic concepts used in our work are explained.

Definition 1 *Measure of need for transaction T_k to attribute A_i*

For each relevant attribute A_i of transaction T_k, the degree of need for T_k to A_i represented by $M_{A_iT_k}$ is equal to the frequency of the execution of T_k,i.e.

$$M_{A_iT_k} = x_{ik} \times f_k$$

Definition 2 *Access Priority of Transaction T_k to attribute A_i*

The access priority of transaction T_k to an attribute A_i is a fraction of the measure of need for transaction T_k to the measure of need for all transactions i.e.

$$P_{A_iT_k} = \frac{M_{A_iT_k}}{\sqrt{\sum_{j=1}^{N_T} M_{A_iT_j}^2}}$$

The higher is the measure of need for transaction T_k to A_i, the more is the access priority of transaction T_k to attribute A_i. Note that $\sum_{k=1}^{N_T} {P_{A_iT_k}}^2 = 1$.

Definition 3 *Attribute Vector A_i*

Each attribute A_i is defined as a normal vector in the transactions space such that each vector has N_T components and the value of each component (a_{ik}) is equal to $P_{A_iT_k}$.

Example 3.1 consider the following attribute vector A_i

	T_1	T_2	T_3	T_4
A_i	0.8	0.6	0	0

As it can be seen, there are four important transactions named T_1, T_2, T_3, and T_4 in a system and the access numbers of attribute A_i for transactions T_1, T_2, T_3, and T_4 equal 0.8, 0.6, 0, and 0, respectively.

Definition 4 *Measure of dissimilarity between two attributes*

This is defined as the cosine distance between attributes A and B. Since all attribute vectors are normal, the measure of dissimilarity between attributes A and B is equal to the following:

$$D_{A,B} = a_1.b_1 + a_2.b_2 + \cdots + a_{N_T}.b_{N_T}$$

Attributes that are accessed together by one or more transactions have high similarity and low dissimilarity with each other than with the other attributes.

4 Problem specification

4.1 Data fragmentation in distributed database systems

For designing distributed database systems, the cost of transaction processing is reduced by localizing data transaction references as well as reducing the number of global references to data. Therefore, vertical fragmentation is defined as dividing relation R into fragments $R_1, R_2, \ldots, R_n$ with the aim that each transaction runs on one fragment as far as possible, which results in minimization of the cost of transaction processing [36].

4.2 Cost model

Generally, two types of cost models are considered for the evaluation of vertical fragmentation algorithms [15]:

1. Models with cost functions based on transaction access analysis on a model DBMS.
2. Models with cost functions based on an empirical assumption.

In each of these models, the input to the function is *AUM* matrix. The design of the first type of the model depends on the database being queried. In this model, the access paths chosen by the query optimizer—such as joining methods and different scans—are used to calculate the cost model. Thus, it has high accuracy in estimating the costs of the fragmentation algorithm. The design of the second model is more general and intuitive, as it focuses more on the costs affected by the partitioning process.

Considering the top-down nature of distributed databases design and given that the information about the physical parameters is available not until the implementation stage, application of the second method seems to be more efficient for our purpose. Therefore, the cost model in this paper applies the partition evaluator (PE) cost function [8], which is a model based on empirical assumption and uses the square-error criterion that is commonly applied to clustering strategies. To evaluate how "good" an algorithm is, the function of PE cost model computes its two constituting components, namely, relevant remote attribute access cost and irrelevant local attribute access cost. Both these costs are calculated via square-error results.

4.2.1 Irrelevant local attribute access cost ($\mathbf{E_M}^2$)

This is a measure of the number of irrelevant attributes in the local fragments of the transactions, which affects the cost as the irrelevant attributes increase the retrieval cost in transaction processing. This is particularly important when dealing with a high number of tuples which is calculated as follows [8]:

$$E_M^2 = \sum_{i=1}^{N_F} \sum_{t=1}^{N_T} \left[f_t^2 \times |s_{it}| \times \left(1 - \frac{|s_{it}|}{n_i}\right)\right]$$

4.2.2 Relevant remote attribute access cost ($\mathbf{E_R}^2$)

This is a measure of the number of transactions' relevant attributes that are located in a remote fragment. This will affect the cost since there is an extra cost for accessing the relevant attributes in non-local fragments which is calculated as follows [8]:

$$E_R^2 = \sum_{t=1}^{N_T} \min \sum_{i=1, i\neq k}^{N_F} \left[f_t^2 \times |R_{itk}| \times \left(\frac{|R_{itk}|}{n_{itk}^r}\right)\right]$$

Thus, transaction processing costs calculated by PE are the sum of local and remote transaction processing, i.e.

$$\text{PE} = {E_M}^2 + {E_R}^2.$$

In general, it is impossible for transactions to gain access to only one fragment with no irrelevant attributes. This is because transactions generally tend to access to intersecting yet different sets of attributes. Moreover, given the distributed nature of the problem and the transactions that run at different sites, it would be impossible to avoid access to a remote fragment. Therefore, as the ideal transaction is impracticable to achieve, the objective of attributes partitioning is to minimize the local and remote transaction processing costs. A more detailed explanation of PE can be found in [8]

As mentioned in Sect. 4, our proposed algorithm consists of different steps; for each, based on its cost calculated above, the desirability of the correspondent fragmentation scheme is calculated.

5 General concepts of ant clustering algorithms

As the name suggests, an algorithm, which is based on ants' behavior, try to simulate the ant conduct in nature. Despite the seeming simplicity of ants' actions as well as their consequent implementation, these models are actually complex organized systems and are especially suitable for optimization problems and problems with distributed nature. Algorithms based on ant behavior have a wide range of applications. Namely, they are used for solving various problems such as data mining [30], assignment [40,45,46], scheduling [47], data allocation [1] and document clustering [20,22,38].

There are some advantages to algorithms based on ant behavior. Firstly, the distributed nature of these algorithms results in scalability that is one of the most important issues in clustering problems [40]. Secondly, as for large-scale problems, these algorithms run faster than other algorithms [21]. Moreover, due to their stochastic nature, these algorithms can explore space problem more thoroughly than greedy algorithms in which only local search is used. Thus, algorithms based on ant behavior are capable of finding close to optimal solutions [11].

A group of algorithms based on ant behavior, which use ant clustering meta-heuristic, is referred to ant clustering algorithm. They are all derived from "basic" ant clustering algorithm presented by Deneuborg et al. [14], where by using a simple local interaction with no centralized control or global representation of the environment, they proposed a basic model to interpret ants' behavior [4].

6 Data clustering problem

Data clustering is a technique of unsupervised learning, commonly used for statistical data analysis in many fields, such as document clustering [25,35], data mining [28,39], plan recognition [48] etc. Ant clustering algorithms have been successfully applied to the data clustering problem [21], which is known to be an NP-Hard problem [18]. Formally, data clustering is defined as follows [8]:

Data clustering problem for n attributes in a d-dimensional metric space is that of partitioning attributes into m groups, or clusters, such that the attributes in a cluster are more similar to each other than to attributes in other clusters.

By introducing a measure of dissimilarity between data objects, Lumer and Faieta [29] succeeded to extend a basic model of ant clustering algorithm for data clustering problem for the first time. We refer to their algorithm as Lumer and Faieta ant clustering (LFAC) algorithm. The pseudo code of this algorithm is presented here [29]:

Algorithm LFAC

```
(1) Initialize the objects and ants' positions on the 2D-Grid randomly
(2) REPEAT
(3)         FOR EACH ant ant_i DO
(4)                 Move ant_i,
(5)                 IF ant_i does not carry any objects THEN pickup an object o_i
                    by considering the probability of P_pickup_i
(6)                 ELSE (ant_i is already carrying an object o_j) drop o_j
                    by considering the probability of P_drop_i
(7) UNTIL optimum solution is acquired.
(8) Print out the generated solution
```

The probability of picking or dropping an object is computed as follows [29]:

$$P_{\text{pickup}_i} = \left(\frac{K_p}{K_p + f(i)}\right)^2$$

$$P_{\text{drop}_i} = \left(\frac{f(i)}{K_d + f(i)}\right)^2$$

Where $f(i)$ is a measure of similarity between an object and other objects in the neighborhood, which increases once their disparity (or distance in feature space) decreases; K_p and K_d are, respectively, pickup and dropping threshold parameters.

Considering clusters as regions of attribute pattern space in which dense patterns are separated by regions of lower attribute density, we can adopt the LFAC algorithm for vertical fragmentation problem [8]. As we can see in LFAC algorithm, the goal is to find clusters that not only minimize the intra-cluster variation, but they also maximize the inter-cluster variation. However, besides clustering goal in vertical fragmentation, during the transaction processing, we need to find optimal number of clusters that minimize the access cost of transactions of attributes. In other words, the significance of number of clusters as an important factor which affects the trade-off between local and remote transaction processing costs is realized [8]. Therefore, the best fragmentation that is to achieve the first goal does not necessarily attain the second. Hence, in some cases, we must slightly sacrifice the similarity between attributes and merge some of the fragments in order to achieve optimum number of fragments and thus, reduce the access cost of transaction processing.

As a result, in addition to the similarity between attributes, there is an extra concept in vertical fragmentation problem that must be considered in order to extend the Lumer and Faieta's model for the data fragmentation problem. We add the cost model presented in Sect. 4.2 so as to meet the second goal of vertical fragmentation.

7 Proposed algorithms

We proposed a new algorithm based on ant behavior for vertical fragmentation problem called hybrid ant clustering algorithm (HACA). In our proposed algorithm, according to ant clustering meta-heuristic, each ant compares a candidate attribute with its 8 surrounding cells—also referred to as the neighborhood space—, to decide whether it should be picked up or dropped. In order to find similar attributes to assign to a cluster, a measure of average similarity between candidate attribute and the other attributes in its neighborhood is used. To facilitate the increase both in the quality of responds and the algorithm's speed, an eight-bit short memory is considered for each ant, so that each ant can remember the last eight

attributes that it has dropped. Hence, instead of moving through the grid and searching for the best location, each ant, after picking up an attribute, compares the selected attribute with the last eight attributes that it has dropped to find the most similar one. The ant memory is then updated after its carried attribute is dropped.

Here, we add two new concepts to LFAC algorithm in order to solve the vertical fragmentation problem. Firstly, we consider a 2-bit memory called current fragment number (CFN) and Optimum Fragment Number (OFN) for each attribute, where the former shows the index number of the fragment in which the attribute is located in the current iteration, and the latter shows the index number of the optimal fragment from the start. Secondly, we add PE function to evaluate the generated solution in each iteration, and if any, we replace it with the better one. Besides this, in order to prevent spending large amounts of time searching for new attributes to be picked up, each ant is loaded with a new attribute immediately after it drops its load. In other words, after dropping an attribute, the ant randomly chooses another attribute from the index of all attributes that are not being carried. It then moves to the right position and tries to pick it up. In case of failure, another attribute is chosen randomly.

The generic HACA algorithm for vertical fragmentation problem is described below:

Algorithm HACA

(1) Initialize the Attributes positions on the $2D$-Grid randomly
(2) Create a feasible solution as an *optimum_solution* by the use of Solution Constructor
(3) $TT = 30$
(4) $TTCounter$=0
(5) $Optimum\_Solution\_Cost = \infty$
(6) REPEAT
(7) FOR EACH ant ant_iDO
(8) IF ant_i does not carry any objects THEN
(9) IF exists attribute A_i in *UnseenAttributes* THEN
(10) BEGIN
(11) Load ant_iAttributeA_i
(12) Remove $A_i$from *UnseenAttributes*
(13) END
(14) ELSE (*UnseenAttributes* is empty)
(15) Load ant_i if possible, Attribute A_i (not carried) considering the probability of P_{pickup_i}
(16) ELSE (ant_i is already carrying an Attribute A_i)
(17) BEGIN
(18) Find the most similar attribute to A_i in ant_i_*memory*
(19) Possibly drop A_i near it by considering the probability of P_{drop_i}
(20) END
(21) END FOR
(22) IF *UnseenAttributes* is empty THEN
(23) BEGIN
(24) Evaluate the cost of the *current_solution*
(25) IF *Current_Solution_Cost*<*Optimum_Solution_Cost* THEN
(26) BEGIN
(27) $TTCounter$ =0
(28) FOR EACH Attribute A_i DO
(29) $OFN_{A_i} = CFN_{A_i}$
(30) $Optimum\_Solution\_Cost = Current\_Solution\_Cost$
(31) END
(32) END
(33) ELSE ($Current\_Solution\_Cost >= Optimum\_Solution\_Cost$)
(34) $TTCounter = TTCounter + 1$
(35) UNTIL $Optimum\_Solution\_Cost \neq \infty$ && $TTCounter > TT$
(36) Print out the optimum solution

8 Description of our proposed algorithm

8.1 Initialization

In the initialization step, a primary solution is generated as follows. First, all the attributes are put into a list called *UnseenAttributes*. Then, in each iteration, as long as there is an attribute in the *UnseenAttributes*, each ant, if unloaded, is loaded by a randomly selected attribute from the list without considering the probability of picking up that attribute. The attribute is then removed from the list. If the ant is loaded, it tries to unload its attribute in the same cluster as that of the most similar attribute (MSA) to A_i in its short-term memory. In the latter case, A_i is located in MSA's cluster with probability P_{drop}, and the amount of CFN for A_i is changed to MSA's CFN. Otherwise, A_i is dropped in a random-free location in the grid, and its CFN is set to a new fragment number.

8.2 Fragmentation

The main difference between the initialization and the fragmentation step is that in the former, we ignore the probability of picking up an attribute as that is defined only after the attributes are located on the grid. In this step, a solution is constructed as follows:

In each iteration, each ant_j, if unloaded, tries to pickup an attribute A_i among all attributes that are not being carried; if loaded, the dropping operation is carried out as in the initialization step. Similar to the canonical algorithm, the probability of picking up and dropping the attributes is based on the measure of average similarity of A_i with the other attributes in its neighborhood. However, we modified the similarity measure function presented in LFAC algorithm [29] by considering some of concepts related to the data fragmentation problem. In the following, the modified similarity measure function is described.

8.2.1 Similarity measure function

The similarity measure function computes the degree of similarity of the attributes being compared, based on their neighboring being accessed by transactions. The modified measure of similarity between the candidate attribute and its neighbors', during picking up or dropping, is calculated as follows:

$$f(i) = \max\left(0, \frac{1}{s^2} \sum_{\{c_{ij} \in s | c_{ij} \neq \text{Null}\}} \left(1 - \frac{D_{i,C_{ij}}}{\propto \mu}\right)\right)$$

Where $f(i)$ is the similarity measure, s is the neighborhood space for candidate attribute i, D_{i}, C_{ij} is the degree of dissimilarity between attribute i and its neighbors, and α is the tune-up accuracy parameter which is used in adaptive tuning of the scale of similarity among attributes.

Value of α varies between (0, 1) such that it is initially set to 0.1, and after a number of iterations, if the number of attributes being dropped is less than the drop threshold, α is increased by 0.01 [20]. As a results, at first, the attributes that have a high degree of similarity are clustered together, and if there are no such attributes, α parameter is increased gradually in order to find a scale of similarity between attributes. The parameter μ, which equals average distance between all attributes, is calculated as below [20]:

$$\mu = \frac{2}{N_A\,(N_A - 1)} \sum_{i=1}^{N_A} \sum_{j=1}^{i-1} d\,(i.j)$$

8.3 Evaluation

At the end of each iteration, the resulting fragmentation is evaluated by PE function. If the new fragmentation is better than the previous optimal fragmentation, value of CFN is replaced by OFN, and the new fragmentation is considered as an optimal one.

8.4 Termination

In order to determine the required number of iterations, we consider a termination threshold parameter (TT) so that if after TT consequent number of iterations no better solution than that of the OFN is obtained, then the algorithm terminates. Termination threshold is calculated as follows:

$$\mathrm{TT} = \alpha \times \frac{N_{\mathrm{Att}}}{\left(p_{\mathrm{pickup}} \times N_{\mathrm{ant}}\right)}$$

Determining α is case dependent, and it increases with the size of the problem. The best value for α in our case is in range [1,2].

9 An alternative algorithm to be compared with proposed algorithms

To compare the accuracy and the quality of obtained results in our proposed algorithm, we compared our algorithm with GRGS-GA whose scalability and accuracy of result are greater than other existing algorithms [15].

9.1 GRGS-GA

The following shows the pseudo code the GRGS-GA algorithm:

Algorithm GRGS-GA

```
(1) Construct chromosomes to generate an initial population
(2) Generate a solution for each chromosome by using the mapping heuristic
(3) Evaluate the Generated solutions
(4) Number_Of_Generations = 0
(5) REPEAT
(6)      Use Tournament Selection to choose chromosomes for generating the next population
(7)      Perform crossover and mutation for these set of chromosomes
(8)      Generate a solution for each chromosome by using the mapping heuristic
(9)      Evaluate the Current Generated solutions
(10)     Number_Of_Generations = Number_Of_Generations + 1
(11) UNTIL Number_Of_Generations = Max_Number_Of_Generations
(12) Print out the optimum solution
```

This algorithm uses grouping-based method in genetic algorithms. The structure of each chromosome is defined as below:

1	2	…	i	i+1	…	n-1	n
F_1	F_2	…	F_i	F_{i+1}	…	F_{n_1}	F_n

Each cell presents an attribute while their contents (F_i) show the fragment number each attribute belongs to. In order to prevent encoding redundancy in this algorithm, they used a restricted growth (RG) string constraint in constructing chromosomes and performing group-oriented genetic algorithm (GA) operations, in such a way that:

$$F_{i+1} \leq (\max(F_1, F_2, \ldots F_i) + 1), \quad 0 < i < n, F_1 = 1$$

In this algorithm, first, a population of chromosomes is created by a constructor function. The constructor function makes use of a function named *rectifier* to apply RG string constraint, which has been referred above. In [15], the *rectifier* is explained in detail. At the end of iteration, an evaluation function is run on the new population to determine the fitness of the chromosomes. The evaluation function used in this algorithm is the one which is presented in [8]. To create the next generation, chromosomes are selected based on their fitness and the cross-over and the mutation operations are carried out on them. The details of this algorithm are presented in [15]. The operations are repeated until the number of generations reaches a maximum limit. The best solution is printed out at the end of the algorithm.

10 Evaluation of algorithms

In this section, we present the results of our proposed algorithms' evaluation. The two types of the evaluations carried out are as follows:

1. Theoretical evaluation
2. Experimental evaluation

10.1 Theoretical evaluation

For this evaluation, we compare computational complexity of our proposed algorithm with that of GRGS-GA. Therefore, we first calculate the computational complexity of each of them.

10.1.1 Computational complexity of HACA

Following our previous explanations of HACA, in order to evaluate this algorithm, first, we randomly locate attributes of each ant in a grid and then, evaluate the process of picking up an attribute. In this process, first, each ant calculates P_{pickup} for any candidate attribute, and then a random number between (0, 1) is selected. If P_{pickup} is less than or equal to the random number, the attribute is picked up by the ant; otherwise, the above processes will be followed for the $n-1$ remaining attributes. If the condition is not satisfactory, these operations will be continued until all the attributes will have been investigated. Due to the stochastic nature of the pickup process, the complexity of each ant's pickup operation is calculated as its expectation (E(pickup)). The probability of P_{pickup} for attribute i being less than or equal to the random number is presented by $\mathrm{P}_{\mathrm{Att}_i}$. As these probabilities are independent from each other, E(pickup) is obtained as follows:

$$E\,(\text{pickup}) = 8 \times \left(\sum_{i=1}^{N_{\text{ATT}}} \frac{1}{N_{\text{ATT}}} \times P_{\text{Att}_1} + \left(1 - P_{\text{Att}_1}\right) \sum_{i=1}^{N_{\text{ATT}}-1} \frac{1}{N_{\text{ATT}} - 1} \times P_{\text{Att}_2} \times 2 + \ldots \right.$$
$$\left. + \left(1-P_{\text{Att}_1}\right)\left(1-P_{\text{Att}_2}\right)\cdots\left(1 - P_{\text{Att}_{n-1}}\right) \sum_{i=1}^{N_{\text{ATT}}-(N_{\text{ATT}}-1)} \frac{1}{N_{\text{ATT}}-(N_{\text{ATT}}-1)} \times P_{\text{Att}_n} \times n \right)$$

However, selection of an attribute by each ant depends on some other factors such as the location of each attribute, the pattern of the located attributes in the grid, the priority of transaction accesses to the attributes, the degree of similarity between the candidate attribute and its neighbors, and so on. Hence, we cannot have enough information for P_{pickup} precise calculation. Thus, it is assumed that P_{pickup} has a random value of X, and value of the randomly selected number is Y. The probability of a random variable being less than a random value is as follows:

$$P_{\text{Att}_i} = \{P\,(Y \leq X) \mid X, Y \in [0, 1]\} = 1/2$$

Therefore,

$$E\,(\text{pickup}) = 8 \times \left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots + \frac{n}{2^n}\right)$$

Thus, when $n \to \infty$, $E\,(\{\text{pickup}\}) = 16$.

Since each attribute vector has dimension N_T, in order to compute the degree of dissimilarity between each two attributes, N_T operations are carried out. Hence, number of required operations to pick up an attribute is equal to the following:

$$N_{OP_{\text{pickup}}} = 16 \times N_T$$

Secondly, in dropping process, $f(i)$ is computed for each of the eight attributes in the ant's memory. After finding the most similar attribute, in order to compute the P_{drop}, $f(i)$ is computed for the eight neighboring-free spots near the most similar attribute(MSA), and then, a random number between (0, 1) is selected. If P_{drop} is less than or equal to the random number, the carried attribute (A_i) is dropped in the same cluster as that of MSA, otherwise A_i is dropped in a random-free location in the grid. Hence, the number of operations is equal to the following:

$$N_{OP_{\text{drop}}} = 64 \times N_T$$

Following each iteration, a comparison between the current solution and the best solution is made by PE so as to select the best one. As presented in PE formula, the computational complexity of PE is equal to the following:

$$N_{OP_{PE}} = (N_F \times N_T \times N_{\text{Att}}) + \left(N_F^2 \times N_T \times N_{\text{Att}}\right)$$

Thus, the total number of operations in HACA algorithm are computed as follows:

$$\begin{aligned} N_{OP_{\text{Total}}} = N_I\,(N_{\text{ANT}}\,((16 \times N_T) + (64 \times N_T)) + (N_F \times N_T \times N_{\text{Att}}) \\ + \left(N_F^2 \times N_T \times N_{\text{Att}}\right)\big) \end{aligned}$$

Hence, the computational complexity of HACA is equal to the following:

$$O\left(\max\left((N_I \times N_{\text{Ant}} \times N_T), (N_I \times N_F^2 \times N_T \times N_{\text{Att}})\right)\right)$$

10.1.2 Computational complexity of GRGS-GA

In [15], the computational complexity of GSGR-GA has not been computed, therefore, in order to compare it with the computational complexity of our proposed algorithms; we compute its computational complexity first.

As mentioned before, in this algorithm, first, a population of chromosomes is generated by a constructor function, named as *rectifier*, which acts as follows.

For each fragment number x that is greater than the previous numbers in a chromosome, it finds the smallest possible number between the last investigated number and x; then, the number replaces each gene that equals x. Thus, in the worst case, the number of operations is equal to the following:

$$N_{\mathrm{REC}} = \sum_{i=1}^{\mathrm{N_{ATT}}} \sum_{j=\mathrm{i}}^{\mathrm{N_{Att}}} 1 = \frac{\left(\mathrm{N_{Att}^2} + \mathrm{N_{Att}}\right)}{2}$$

where N_{REC} is the number of instruction carried out for imposing RG string constraint.

Next, by using a PE, the accuracy of generated fragments in the population is evaluated; based on which, they are sorted in the population. Hence, the number of the required instructions is equal to the following:

$$N_P = N_{\mathrm{pop}} \times \left(N_{\mathrm{ATT}} + N_{\mathrm{REC}} + \left(N_F^2 \times N_T \times N_{\mathrm{Att}}\right)\right) + N_{\mathrm{POP}} \log N_{\mathrm{POP}}$$

Next, in each iteration, with the help of tournament selection $(N_{\mathrm{POP}} - N_{\mathrm{Elitism}}/2)$, some pairs of chromosomes are selected so as to generate $N_{\mathrm{POP}} - N_{\mathrm{Elitism}}$ children, i.e. two children for each pair. Here, N_{POP} refers to the number of existing chromosomes in the population, and N_{Elitism} is the number of chromosomes with maximum fitness in the population. Then, the cross-over operations are carried out on each selected pair of chromosomes with probability P_c. Based on the length of each chromosome, the number of operations in this section is $2 \times N_{\mathrm{ATT}}$. In order to impose the RG string constraint on each child, the rectifier that is being run requires $2 \times N_{\mathrm{REC}}$ units of instruction to impose restriction on two children. In some cases, having occurred with rate P_m, instead of cross-over operations, the mutation operation is run on one of the parents to create a child, and then the rectifier imposes the restriction. In this case, number of operations is equal to $3 + N_{\mathrm{REC}}$. At the end, in the survival selection, through best fitting selection, the children are substituted by a new population where chromosomes are sorted based on their fitness. Hence, required units of instruction is equal to the following:

$$\begin{aligned} N_{op} = {} & N_G \times (N_{\mathrm{POP}} - N_{\mathrm{Elitism}})\left((2 \times 2) + P_C \times (2 \times (N_{\mathrm{ATT}} + N_{\mathrm{REC}})\right) \\ & + P_m \times (3 + N_{\mathrm{REC}}) + (N_{\mathrm{POP}} - N_{\mathrm{Elitism}}) + \left(N_{\mathrm{POP}} \times \left(N_F^2 \times N_T \times N_{\mathrm{Att}}\right)\right) \\ & + N_{\mathrm{POP}} \log N_{\mathrm{POP}}) \end{aligned}$$

Therefore, the total number of operations in this algorithm is equal to the following:

$$N = N_p + N_{op}$$

As in [15], their proposed parameter values are $N_{\mathrm{Elitism}} = 20$, $P_c = 0.8$, and $P_m = 0.01$; the computational complexity of their algorithm is given by the following:

$$O\left(\max\left(\left(N_G \times N_{\mathrm{POP}} \times N_{\mathrm{ATT}}^2\right), \left(N_G \times N_{\mathrm{Pop}}^2 \times N_F^2 \times N_T \times N_{\mathrm{Att}}\right)\right)\right)$$

10.1.3 Comparing computational complexity of HACA and GRGS-GA

As stated in [5,33,36], an "optimal" solution is possibly closer to a full relation than to a set of fragments, each of which consists of a single attribute. Therefore, in an optimal solution, a relation is likely to be partitioned into a few fragments; thus, N_F can be overlooked against the other variables. As mentioned in [9], applying 80/20 rule, we can reduce the number of important transaction in a system to be disregarded in comparison with the number of attributes. Although the concepts of population and ants are not the same, the growth rate of their

respective parameters is roughly the same. Thus, they are comparable with each other. The remaining parameters are the number of iterations/generations and the number of attributes, which affect the size and complexity of the problem space. Furthermore, the concept of N_I and N_G is also the same, and they too can be compared with each other. However, as the experimental results confirm, N_{Ant} and N_I are less than N_{POP} and N_G, respectively. Moreover, the complexity in HACA grows linearly with respect to N_{Att}, while in GRGS-GA, it grows non-linearly. Thus, the effect of increasing the number of attributes on computational cost and the scalability of algorithm in GRGS-GA are much more than that of in HACA. Based on what went on, we can see that the complexity of HACA is less than that of GRGS-GA, and hence, it is more scalable than GRGS-GA.

10.2 Experimental evaluation

Since GRGS-GA generates more optimal solutions and is more scalable than other existing fragmentation algorithm [15], it is experimentally compared with our algorithm. In our experiment, the goal is to investigate the effect of the scale of problem on the accuracy of obtained results.

10.2.1 Test data generation

Problem generator In order to test the algorithm, a PG function is used to create problem instances. First, upon receiving the number of transactions and the number of attributes, the PG function randomly generates each of the components of the attribute usage matrix. Then, it creates the transaction frequency by selecting a number between [1,999] for each transaction. Next, based on the information generated so far, it constructs a problem instance and determines the actual optimum solution for each case. Thus, complex pattern of accesses of transactions to attributes is generated.

10.2.2 Experimental results

In this experiment, the accuracy of the obtained results by GRGS-GA and HACA is compared in terms of the average cost of transaction processing. The experiment is carried out on five data fragmentation configurations, which are generally divided into two classes of configurations:

Class 1: This class contains two of the problem instances which are used in [3,7,23,28]. The first one contains 10 attributes and 8 important transactions, while the other contains 20 attributes plus 15 important transactions. Both algorithms in this class are run 100 times for each problem instance.

Class 2: This class contains three problem instances which are generated by SG. The first one contains 100 attributes plus 40 important transactions, the second has 500 attributes and 60 important transactions, and yet, the last one contains 800 attributes and 80 important transactions. Both algorithms in this class are run 15 times for each problem instance.

Details of each problem instance are shown in Table 2.

In each problem instant mentioned above, number of ants, number of iterations and size of grid have different values, but the other input parameters are constant in all of the problem instances. The values of these parameters are shown in Table 3.

The average results obtained for each problem instance through each algorithm are shown in Table 4.

Table 2 Problem instances details

Problem instance	Optimum cost $\times 10^4$	Optimum no. of fragment
10Att-8Tran	0.5820	3
20Att-15Tran	0.4627	4
100Att-40Tran	53.2641	15
500Att-60Tran	127.6348	25
800Att-80Tran	296.4351	35

Table 3 Selected values for the parameters

Description	Parameters	Values
Constant value of pick up process in HACA algorithm	K_p	0.1
Constant value of dropping process used in HACA algorithm	K_d	0.1
Considered neighborhood space for each attribute in HACA algorithm	S	5
Number of memory cells for each ant used in HACA algorithm	$NoAntMem$	8
Number of selected chromosomes using Best Fit Selection in GRGA-GS	N_{Elitism}	20
Probability of performing cross-over operations in GRGS-GA	P_C	0.8
Probability of performing mutation operations in GRGS-GA	P_m	0.01

Table 4 Evaluation of results achieved by algorithms

Algorithm name	Experiment results	10Att-8Tran	20Att-15Tran	100Att-40Tran	500Att-60Tran	800Att-80Tran
HACA	Number of iterations	**23.7**	**32.8**	**469.7**	**2950.2**	**6441.9**
	Number of ants	3	5	20	60	95
	TT	10	10	20	30	50
	Average number of fragments	**3.03**	**4.06**	**15.26**	**26.33**	**37.46**
	Average cost $\times 10^4$	**0.5826**	**0.4654**	**54.8176**	**131.5080**	**320.8516**
	No. of times hitting actual optimum	**97**	**94**	**13**	**10**	**8**
GRGS-GA	Number of generations	25.1	36.9	604.7	3491.3	8012.5
	Size of population	100	150	200	250	300
	TT*	10	10	50	100	150
	Average number of fragments	3.05	4.09	15.40	27.60	38.73
	Average cost*10^4	0.5830	0.4667	55.5943	139.8508	333.7661
	No. of times actual optimum is hit	95	91	12	7	4

* The TT values for GRGS-GA have been given based on [15]

In Table 4, the average cost of access to attributes, the optimal value for average number of fragments, the maximum number of times actual optimum is hit, and the optimal number of iterations/generations for each problem instance are shown in bold.

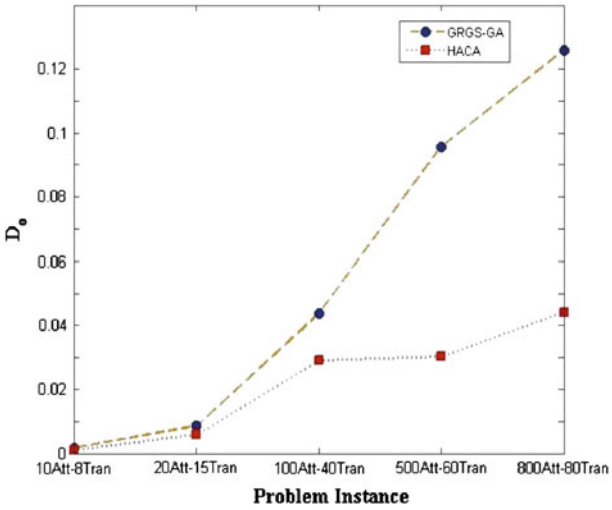

Fig. 1 deviation from optimum solution

As shown in the experimental results, the number of iterations in HACA is less than the one in GRGS-GA. This means that HACA converges to the optimum solution faster than GRGS-GA. Moreover, among 245 runs for each of the algorithms for the above problem instances, HACA hits the optimum solution 222 times, while GRGS-GA hits it 209 times. In other words in 90.6% of times, HACA is successful to find the optimum solution, and in 9.4%, it finds a suboptimum solution while these numbers for GRGS-GA are 85.4 and 14.6%, respectively. Also, based on the average cost of obtaining solutions, the deviation from optimum solution (D_O) in each problem instance is calculated as follows:

$$D_O = \frac{(\text{AverageCost} - \text{OptimumCost})}{\text{OptimumCost}}$$

Figure 1 shows the deviations from optimum solutions for both algorithms.

As shown in Fig. 1, in each problem instance, the quality of the solution in HACA is higher than that of GRGS-GA. On the other hand, for small-scale problems, the differences between the solutions obtained by HACA and GRGS-GA are low. However, as the size of problem increases, difference between the accuracy of solutions obtained by HACA and GRGS-GA is remarkable.

11 Conclusion

In this paper, we address the prominent issue of vertical fragmentation problem in distributed database systems. We propose a new vertical fragmentation algorithm based on the heuristic method resulted from the integration of ant clustering algorithm and an optimizing number of fragments function.

In our theoretical and experimental evaluation studies, we have assessed the computational complexity as well as the accuracy of the obtained results from our proposed algorithm and compared them with those of GRGS-GA algorithm. These evaluations have revealed that our proposed algorithm possesses a superior complexity in comparison with GRGS-GA,

which in terms of both computational cost and further scalability make it a less expensive algorithm than GRGS-GA. In addition, it affords to generate a much more accurate solution for particularly large-scale problems than GSGRGA.

Future studies could examine other criteria such as considering replication in vertical fragmentation problem. Moreover, other heuristic methods like support vector machine (SVM), particle swarm optimization (PSO), and self-organizing map (SOM), which have proved successful in clustering problems, could be used to handle vertical fragmentation problem.

References

1. Adl KR, RouhaniRankoohi SMT (2009) A new ant colony optimization based algorithm for data allocation problem in distributed databases. J Knowl Inf Syst (Springer)
2. Babad M (1977) A record and file partitioning model. Commun ACM 20(1):29–31
3. Benner H (1967) On designing generalized file records for management information systems. In: Proceedings of the fall joint computer conference, pp 291–303
4. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems, institute studies in the sciences of complexity. Oxford University Press, Santa Fe
5. Ceri S, Plagatti G (1984) Distributed databases principles and systems. McGraw-Hill Book Company, New York
6. Ceri S, Navathe SB, Weiderhold G (1983) Distribution design of logical database schemas. IEEE Trans Softw Eng 9(4):487–503
7. Ceri S, Pernici S, Weiderhold G (1989) Optimization problems and solution methods in the design of data distribution. Inf Sci 14(3):261–272
8. Chakravarthyt S, Varadarajan R, Navathe SB, Muthuraj J (1993) A formal approach to the vertical partitioning problem in distributed database design. In: Proceedings of parallel and distributed information systems (PDIS-2) IEEE, pp 26–34
9. Chu WW, Fellow IEEE, Ieong IT (1993) A transaction-based approach to vertical partitioning for relational database systems. IEEE Trans Softw Eng 19(8):804–8012
10. Cornell D, Yu P (1987) A vertical partitioning algorithm for relational databases. In: Proceeding of third international conference on data engineering, pp 30–35
11. Cui X, Potok TE, Palathingal P (2005) Document clustering using particle swarm optimization. In: IEEE transaction on swarm intelligence symposium(SIS) proceedings, pp 185–191
12. Day H (1956) An optimal extracting from a multiple file data storage system: an application of integer programming. Oper Res 13(3):482–494
13. Dearnley P (1974) Model of a self-organizing data management system. Comput J 17(1)
14. Deneuborg JL (1990) The dynamics of collective sorting robot-like ants and ant-like robots. In: 1st international conference on simulation of adaptive behaviour: from animals to animats, vol 1. MIT Press, pp 356–363
15. Du J, Alhajj R, Barker K (2006) Genetic algorithms based approach to database vertical partition. J Intell Inf Syst 26:167–183
16. Eisner M, Severance D (1976) Mathematical techniques for efficient record segmentation in large shared databases. J ACM 23(4)
17. Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, England
18. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, USA
19. Hammer M, Niamir B (1979) A heuristic approach to attribute partitioning. In: Proceedings ACM SIGMOD international conference on management of data
20. Handl J, Meyer B (2002) Improved ant-based clustering and sorting in a document retrieval interface. In: Proceeding of the 7th internatioanl conference on parallel problem solving from nature, pp 913–923
21. Handl J, Knowles J, Dorigo M (2003) On the performance of ant-based clustering. Front Artif Intell Appl 104:204–213
22. Hoe K, Lai W, Tai T (2002) Homogenous ants for web document similarity modeling and categorization. In: Proceedings of the third international workshop on ant algorithms, LNCS, vol 2463. Springer, Berlin, pp 256–261
23. Hoffer J (1976) An integer programming formulation of computer database design problems. Inf Sci 11:29–48

24. Hoffer J, Severance D (1975) The uses of cluster analysis in physical database design. In: Proceeding of 1st international conference on VLDB, Framingham, pp 69–86
25. Jing L, Ng MK, Huang JZ (2010) Knowledge-based vector space model for text clustering. J Knowl Inf Syst (Springer) 25(1):35–55
26. Kennedy R (1973) The use of access frequencies in database organization. PhD Dissertation, The Wharton School, University of Pennsylvania
27. Kennedy SR (1972) A file partition model. Technical report in information science
28. Kranen P, Assent I, Baldauf C, Seidl T (2010) The ClusTree: indexing micro-clusters for anytime stream mining. J Knowl Inf Syst (Springer)
29. Lumer E, Faieta B (1994) Diversity and adaption in populations of clustering ants. In: 3rd international conference on simulation of adaptive behaviour: from animals to animats, vol 3. MIT Press
30. Lumer E, Faieta B (1995) Exploratory database analysis via self-organization. Unpublished manuscript. Results summarized in
31. March S, Severance D (1977) The determination of efficient record segmentation and blocking factors for share data files. ACM Trans Database Syst 2(3):279–296
32. McCormick W, Schweitzer P, White T (1972) Problem decomposition and data reorganization by a clustering technique. Oper Res
33. Navathe S, Ceri S, Wiederhold G, Dou J (1984) Vertical partitioning algorithms for database design. ACM Trans Database Syst 9(4)
34. Navathe SB, Ra M (1989) Vertical partitioning for database design: a graphical algorithm. ACM SIGMOD Record 18(2):440–450
35. Ni X, Quan X, Lu X, Wenyin L, Hua B (2010) Short text clustering by finding core terms. J Knowl Inf Syst (Springer)
36. Ozsu MT, Valduriez P (1999) Principles of distributed database systems. Printice Hall, Englewood Cliffs
37. Pérez J, Pazos R, Frausto J, Romero D, Cruz L (1998) Vertical fragmentation and allocation in distributed databases with site capacity restrictions using the threshold accepting algorithm. Parallel Distributed Comput Syst, Las Vegas, pp. 210–213
38. Ramos V, Merelo JJ (2002) Self-organized stigmergic document maps: environments as a mechanism for context learning. In: Proceedings of the first Spanish conference on evolutionary and bio-inspired algorithm, pp 284–293
39. Sakuma J, Kobayashi S (2009) Large-scale k-means clustering with user-centric privacy-preservation. J Knowl Inf Syst (Springer)
40. Sarathy R, Shetty B, Sen A (1997) A constrained nonlinear 0–1 program for data allocation. Eur I Oper Res 102:626–647
41. Schkolnic M (1977) A clustering algorithm for hierarchical structures. ACM TODS 1(2):27–44
42. Seppala Y (1967) Definition of extraction files and their optimization by zero-one programming. BIT 7(3):206–215
43. Song SK, Gorla N (2000) A genetic algorithm for vertical fragmentation and access path selection. Comput J 43(1)
44. Stocker M, Dearnley A (1973) Self-organizing data management systems. Comput J 16(2)
45. Stutzle T (1997) MAX-*MIN* Ant system for the qadratic assignment problem. Technical report AIDA-97-4, FG Intellectik, FB Informatik, TU Darmstadt, Germany
46. Stutzle T, Dorigo M (1999) ACO algorithms for the quadratic assignment problem. In: Corne D, Dorigo M, Glover FNew ideas in optimization. McGraw-Hill, Maidenhead
47. Taillard ED (1995) Comparison of iterative searches for the quadratic assignment problem. Locat Sci 3:87–105
48. Takacs B, Demiris Y (2009) Spectral clustering in multi-agent systems. J Knowl Inf Syst (Springer)

Author Biographies

Mehdi Goli Received the B.S. and M.S. degrees in Computer Engineering-Software from ShahidBeheshti University, Iran in 2005 and 2009, respectively. He is currently studying another M.Sc. degree at the department of computer science at the University of Edinburgh. His research interests include distributed database systems, data integration and exchange artificial intelligence, and Xml databases.

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 associate professor at ShahidBeheshti 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.