ANTS for Data Warehouse Logical Design

Vittorio Maniezzo * Antonella Carbonaro * Matteo Golfarelli †
Stefano Rizzi †

* Department of Computer Science
University of Bologna, Italy
Email: {maniezzo, carbonar}@csr.unibo.it

† DEIS
University of Bologna, Italy
Email: {mgolfarelli, srizzi}@deis.unibo.it

1 Introduction

The problem addressed in this research, namely the vertical fragmentation problem (VFP), arises in the context of data warehouse design, when trying to minimize the system average query response time. It is a combinatorial optimization problem of significant actual interest which has so far received little attention from the optimization community. This paper describes the application of the ANTS approach, a variant of the Ant Colony Optimization (ACO) paradigm, to the VFP.

Data warehouses are primary commodities in the current software market, supporting the transformation of huge volumes of data into actionable business information, improving targeted marketing and business process re-engineering for customer value management. A data warehouse permits to retrieve summary data, derived from those present in operational information systems. Fundamental issues are flexible query interface and fast query response.

The design of a data warehouse starts with the identification of relevant data in the company information system; these data must be integrated, reorganized and possibly aggregated in order to be of effective use. After that, conceptual, logical and physical design phases are encompassed. The problem addressed in this paper belongs to logical design and has the objective of minimizing the query response time by reducing the number of disk pages to be accessed. This may be obtained by defining appropriate tables of aggregated data (*views*) and by including in them only the data which are actually requested by some query.

Storing aggregated data obviously leads to redundancy, thus generating a trade-off between effectiveness and amount of memory to be allocated. The algorithm presented in this paper directly addresses the optimization of this trade-off. To the best of our knowledge, no effective algorithmic solution has been presented so far in the literature for the problem of interest. The problem has been first described in [10], where no optimization algorithm is proposed. In [2], a related problem is described, aimed at building data indices to enhance performance in parallel implementations of data warehouses. In [5] the problem is formalized and a branch-and-bound approach is devised. A preliminary report on this research has been presented in [9].

2 The Vertical Fragmentation Problem

2.1 Background

The most widely accepted modeling technique for data warehouses, presented in [7], denotes data by means of an n-dimensional (hyper)cube, where each *dimension* corresponds to a characteristic of the data. Each element of the cube is usually associated with quantitative attributes, called *measures*, which are computed from the operational information system. Furthermore, each dimension is related to a set of *attributes* defining a hierarchy of aggregation levels. Elements of the cube can then be aggregated along these hierarchies, in order to retrieve summary values for measures.

More formally, a cube f is a 4-tuple $< Patt(f), Meas(f), Attr(f), R >$, where:

- $Patt(f)$ is a set of *dimensions*;
- $Meas(f)$ is a set of *measures*;
- $Attr(f)$ is a set of *attributes* (being the dimensions particular attributes, we have $Patt(f) \subseteq Attr(f)$);
- R is a set of functional dependencies $a_i \rightarrow a_j$ defined between pairs of attributes in $Attr(f)$, where $a_i \rightarrow a_j$ denotes both the case in which a_i directly determines a_j and the case in which a_i transitively determines a_j.

For example, a 3-dimensional cube with dimensions Store, Product and Date might represent the sales in a chain store; the measures could be Quantity and Revenue. In this case, for each product, each element of the cube would measure the quantity sold in one store in one day and the corresponding revenue. A possible aggregation could be that computing the total monthly revenue for each category of products.

The design of a data warehouse encompasses conceptual, logical and physical design. The objective of logical design, relevant for this paper, is the minimization of query response time. This is obtained by pre-defining the set of queries, called *workload*, that the system is likely to be asked to answer more often. Taking into account all possible queries is computationally infeasible, but it is possible to identify a reduced set of significant and frequent queries which are considered to be representative of the actual workload.

Given a cube f, an *aggregation pattern* (or simply a *pattern*) on f is a set p, $p \subseteq Attr(f)$, such that no functional dependency exists between any pair of attributes in p: $\forall a_i \in p(\nexists a_j \in p; a_i \rightarrow a_j)$. Each pattern on f determines a possible view to be materialized. Given a workload expressed as a set of queries, we will call *candidate views* those being potentially useful to reduce the workload execution cost. Let $Cand(f)$ be the set of the candidate views for cube f; each $v \in Cand(f)$ is defined by its pattern $Patt(v)$. For each cube f, the view at pattern $Patt(f)$ is always a candidate. We will denote with P the set of patterns of all the candidate views on all the cubes involved in the workload.

In presence of a memory constraint, only a subset of the candidate views can be actually materialized. Several techniques have been proposed to select the subset to be materialized in order to optimize the response to the workload [6]. All the approaches in the literature store, for each view $v \in Cand(f)$, all the measures in $Meas(f)$. In this paper, we evaluate how the solution can be further optimized by materializing views in *fragments* including measures requested together by at least one query. In fact, some queries on f may require a subset of $Meas(f)$; thus, it may be worth materializing fragments including only a subset of $Meas(f)$ (*partitioning*). On the other hand, the access costs for some queries may be decreased by materializing fragments which include measures taken from different cubes (*unification*).

With the term *fragmentation* we denote both partitioning and unification of views. The approach we propose in this paper is aimed at determining an optimal set of fragments to materialize from the candidate views. In order to specify objective and constraints of fragmentation, some further notation must be introduced.

Given a cube f and a workload Q, it is possible to partition the measures $Meas(f)$ into subsets (*minterms*) such that all the measures in a minterm are requested together by at least one query in Q and do not appear separately in any other query in Q. We call *terms* the sets of measures obtained as the union of any combination of minterms, even from different cubes (of course, all minterms are also terms). We denote with T the set of all terms.

The fragmentation problem can now be modeled over a *fragmentation array* $\Xi = [x_{ijk}]$, which is a tridimensional array of 0-1 binary variables whose dimensions correspond to the queries $q_i \in Q$, to the patterns $p_j \in P$ and to the terms $t_k \in T$, respectively. Each cell of the array corresponds to a fragment candidate to materialization; setting $x_{ijk} = 1$ means stating that query q_i will be answered accessing (also) the fragment defined by the measures in t_k and pattern p_j.

A value assignment for variables x_{ijk} is feasible if: (i) for every query, each measure required is obtained by one and only one fragment, and (ii) for every pattern, each measure is contained in one and only one fragment. The objective function to minimize is based on the number of disk pages to access in order to satisfy the workload.

2.2 Mathematical formulation

Problem VFP can be formulated as follows. Let $\mathcal{Q}$ be the index set of the queries in the workload and $\mathcal{P}$ the index set of the patterns in P. For every query q_i, $i \in \mathcal{Q}$, $\mathcal{P}_i$ denotes the subset of $\mathcal{P}$ containing the indices of all patterns p_j which are useful to solve query q_i and for which $p_j = Patt(v)$, where v is a candidate view for at least a cube f involved in query q_i, i.e., $v \in Cand(f)$.

The index set $\mathcal{T}$ contains the indices of the terms in T; we will further denote by $\mathcal{T}_i$ the subset of indices of the terms which contains at least one measure in $Meas(q_i)$, $i \in \mathcal{Q}$.

Problem VFP asks to minimize the workload execution cost, computed as the sum of the costs c_{ijk} for obtaining, for each query $i \in \mathcal{Q}$, the relevant term $k \in \mathcal{T}_i$ from pattern $j \in \mathcal{P}_i$.

Let x_{ijk} be a 0-1 variable which is equal to 1 if and only if query i is executed on pattern j to get the term k. Let y_{jk} be a 0-1 variable which is equal to 1 if and only if the pattern j is used to get the term k, in which case an amount b_{jk} of disk space out of the maximum available space amount B is needed. Problem VFP is then as follows.

$$(VFP) \quad z(VFP) = Min \quad \sum_{i \in \mathcal{Q}} \sum_{j \in \mathcal{P}_i} \sum_{k \in \mathcal{T}_i} c_{ijk} x_{ijk} \tag{1}$$

$$s.t. \quad \sum_{j \in \mathcal{P}_i} \sum_{k \in \mathcal{T}_i} x_{ijk} = 1 \qquad i \in \mathcal{Q} \tag{2}$$

$$\sum_{k \in \mathcal{T}} y_{jk} \leq 1 \qquad j \in \mathcal{P} \tag{3}$$

$$x_{ijk} \leq y_{jk} \qquad i \in \mathcal{Q}, j \in \mathcal{P}, k \in \mathcal{T}_i \tag{4}$$

$$\sum_{\substack{j \in \mathcal{P} \\ k \in \mathcal{T}}} b_{jk} y_{jk} \leq B \tag{5}$$

$$x_{ijk} \in \{0,1\} \qquad i \in \mathcal{Q}, j \in \mathcal{P}, k \in \mathcal{T} \tag{6}$$

$$y_{jk} \in \{0,1\} \qquad j \in \mathcal{P}, k \in \mathcal{T} \tag{7}$$

Equations (2) impose that each measure specified in a query must be obtained by one and only one pattern (thus, implicitly, that each query in the workload must be satisfied); inequalities (3) require that, in each pattern, a measure can belong to only one term; inequalities (4) link the x and y variables and inequality (5) is the memory knapsack constraint. Finally, constraints (6) and (7) are the integrality constraints.

By a linear relaxation of integrality constraints we get problem LVFP whose optimal solution value $z(LVFP)$ constitutes a lower bound to $z(VFP)$.

3 ANTS applied to vertical fragmentation

ANTS [8] is a technique to be framed within the ACO paradigm, whose first member called Ant System was initially proposed by Colorni, Dorigo and Maniezzo [1], [3], [4]. The main underlying idea of all ACO algorithms is that of parallelizing search over several constructive computational threads, all based on a dynamic memory structure incorporating information on the effectiveness of previously obtained results.

An *ant* is defined to be a simple computational agent, which iteratively constructs a solution for the problem to solve. Partial problem solutions are seen as *states*; each ant *moves* from a state ι to another one ψ, corresponding to a more complete partial solution.

At each step σ, each ant k computes a set $A_k^\sigma(\iota)$ of feasible expansions to its current state, and moves to one of these according to a probability distribution specified as follows.

For ant k, the probability $p_{\iota\psi}^k$ of moving from state ι to state ψ depends on the combination of two values:

1. the attractiveness $\eta_{\iota\psi}$ of the move, as computed by some heuristic indicating the *a priori* desirability of that move;
2. the trail level $\tau_{\iota\psi}$ of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an *a posteriori* indication of the desirability of that move.

The specific formula for defining the probability distribution of moving from a state to another one makes use of a set $tabu_k$ which indicates a problem-dependent set of infeasible moves for ant k. According to the ANTS approach [8] probabilities are computed as follows: $p_{i\psi}^k$ is equal to 0 for all moves which are infeasible (i.e., they are in the tabu list), otherwise it is computed by means of formula (8), where α is a user-defined parameter ($0 \leq \alpha \leq 1$).

$$p_{\iota\psi}^k = \frac{\alpha \cdot \tau_{\iota\psi} + (1-\alpha) \cdot \eta_{\iota\psi}}{\Sigma_{(\iota\nu)\notin tabu_k}(\alpha \cdot \tau_{\iota\nu} + (1-\alpha) \cdot \eta_{\iota\nu})} \tag{8}$$

The essential characteristic distinguishing ANTS from other ACO algorithm is the structural use of lower bounds during the search process. The lower bound used for the VFP was $z(LVFP)$. More specifically, at every step we compute the cost of the partial solution so far constructed and we remove from the mathematical representation of the problem all constraints of type (2) and (4) which are saturated by the incumbent solution and all variables which cannot belong to any feasible solution due to those already fixed. The lower bound is obtained as the sum of all dual variables associated with the remaining constraints, whose values were computed in the optimal solution of the linear relaxation of the whole problem (LVFP).

Based on the described elements, the ANTS metaheuristic that we implemented is the following.

ANTS algorithm

1. (Initialization)
   Compute a (linear) lower bound LB to the problem to solve.
   Initialize $\tau_{\iota\psi}$, $\forall(\iota,\psi)$ with the primal variable values.

2. (Construction)
   For each ant k **do**
      repeat
         compute $\eta_{\iota\psi}$, $\forall(\iota,\psi)$, as a lower bound to the cost of a complete solution containing ψ.
         choose the state to move to, with probability given by (8).
         append the chosen move to the k-th ant's set $tabu_k$.
      until ant k has completed its solution.
      carry the solution to its local optimum.
   end for.

3. (Trail update)
   For each ant move $(\iota\psi)$ **do**
      compute $\Delta\tau_{\iota\psi}$.
      update the trail matrix.
   end for.

4. (Terminating condition)
   If not(end-test) go to step 2.

Fig.3. Pseudo code for the ANTS algorithm

The ANTS algorithm has been coded in Microsoft Visual C++ and run on a Pentium III, 733 MHz machine working under Windows 98. As a linear programming solver, in order to compute the lower bounds we used CPLEX 6.6. The test set has been obtained from the TPC-D benchmark [11], which is a standard in the data warehousing field. In order to evaluate the algorithm effectiveness, we have defined a number of instances, all derived from the TPC-D by randomly selecting a progressively greater subset of the 40 queries.

Table 1 shows the preliminary results obtained so far. The table columns show: (*prob*) the problem name, where the number indicates how many queries were used to build the instance; (m) the number of constraints; (n) the number of variables; (*reduct*) the number of constraints which can be removed by a specific preprocessing routine; ($lvfp$) the lower bound $z(LVFP)$; ($t\ lvfp$) the cpu time to compute $z(LVFP)$; (zub) the percentual deviation from $z(LVFP)$ of the upper bound; ($t\ zub$) the cpu time to compute the ANTS upper bound.

Problems VFP3 and VFP5 are small-sized problems that we used to fine tune the algorithm elements. For all other problem dimensions we present three instances, which were obtained by randomly selecting the specified number of queries out of the possible 40. Notice the high variability of difficulty deriving from different query sets.

On the small instances, ANTS was able to identify the optimal solution, as testified by the fact that the lower bound has a cost equal to that of the best solution found by ANTS. On bigger instances, the distance between z(LVFP) and the best solution cost found by ANTS increases with the problem size: on those instances more CPU time than the 30 minutes allowed in this test is needed to get good quality results.

prob	m	n	reduct	lvfp	t lvfp	zub	t zub
VFP3	94	76	20	50475.0	0.05	0.00	7.34
VFP5	729	704	34	281816.2	0.22	2.18	1138.22
VFP10A	4780	5338	245	65190.0	0.66	0.00	568.37
VFP10B	360	358	5	33976.0	0.06	0.00	1367.23
VFP10C	592	512	97	116463.0	0.11	0.00	4.23
VFP15A	4962	5528	242	67971.0	0.82	0.12	1046.24
VFP15B	2365	2544	108	132775.7	0.39	0.72	384.38
VFP15C	6410	7118	317	286063.2	4.67	2.03	183.59
VFP20A	9466	10397	405	128094.4	10.77	3.28	428.39
VFP20B	7745	8285	326	165426.2	2.41	5.92	1634.53
VFP20C	36854	40108	1358	186915.7	100.46	10.77	823.48
VFP25A	25855	28121	874	173367.4	10.99	6.57	1772.16
VFP25B	8182	8766	314	169133.6	2.47	19.27	1003.80
VFP25C	57964	62733	1965	145471.3	254.90	33.98	204.55
VFP30A	44758	48133	1339	237247.4	118.36	22.00	1538.23
VFP30B	62973	67484	1735	208801.4	343.29	35.76	704.86
VFP30C	75489	81026	2206	178171.3	625.93	43.88	839.11

Table 1: ANTS results on the set of VFP test problems

References

[1] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of ECAL'91, European Conference on Artificial Life*. Elsevier Publishing, 1991.

[2] A. Datta, B. Moon, and H. Thomas. A case for parallelism in data warehousing and OLAP. In *Proc. IEEE First Int. Workshop on Data Warehouse Design and OLAP Technology*, 1998.

[3] M. Dorigo. *Optimization, Learning and Natural Algorithms*. Ph.D. Thesis. Polit. di Milano, 1992.

[4] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[5] M. Golfarelli, D. Maio, and Rizzi S. Applying vertical fragmentation techniques in logical design of multidimensional databases. In *Proc. 2nd Int. Conf. on Data Warehousing and Knowledge Discovery*, pages 11–23, 2000.

[6] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing Data Cubes Efficiently. In *Proc. ACM Sigmod Conf.*, Montreal, Canada, 1996.

[7] R. Kimball. *The data warehouse toolkit*. John Wiley & Sons, 1996.

[8] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358 – 369, 1999.

[9] V. Maniezzo, A. Carbonaro, M. Golfarelli, and Rizzi S. An ants algorithm for optimizing the materialization of fragmented views in data warehouses: preliminary results. In *Proc. 1st European Workshop on Evolutionary Computation in Combinatorial Optimization*, 2001.

[10] D. Munneke, K. Wahlstrom, and M. Mohania. Fragmentation of multidimensional databases. In *Proc. 10th Australasian Database Conf.*, pages 153–164, Auckland, 1999.

[11] F. Raab, editor. *TPC Benchmark(tm) D (Decision Support), Proposed Revision 1.0*. Transaction Processing Performance Council, San Jose, 1995.