Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design

Alberto Caprara, Matteo Fischetti, and Dario Maio

Reprint

from

IEEE Transactions on Knowledge and Data Engineering

Volume 7, Number 6
December 1995

50 YEARS OF SERVICE
IEEE COMPUTER SOCIETY
1946 - 1996

Washington ◆ Los Alamitos ◆ Brussels ◆ Tokyo

PUBLICATIONS OFFICE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1314 USA

Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design

Alberto Caprara, Matteo Fischetti, and Dario Maio, *Member, IEEE*

***Abstract*—The *index selection problem* (ISP) is an important optimization problem in the physical design of databases. The aim of this paper is to show that ISP, although NP-hard, can in practice be solved effectively through well-designed algorithms. We formulate ISP as a 0-1 integer linear program and describe an exact branch-and-bound algorithm based on the linear programming relaxation of the model. The performance of the algorithm is enhanced by means of procedures to reduce the size of the candidate index set. We also describe heuristic algorithms based on the solution of a suitably defined knapsack subproblem and on Lagrangian decomposition. Finally, computational results on several classes of test problems are given. We report the exact solution of large-scale ISP instances involving several hundred indexes and queries. We also evaluate one of the heuristic algorithms we propose on very large-scale instances involving several thousand indexes and queries and show that it consistently produces very tight approximate (and sometimes provably optimal) solutions. Finally, we discuss possible extensions and future directions of research.**

***Index Terms*—Index selection problem, relational database, physical database design, 0-1 integer linear programming, branch-and-bound algorithm, heuristic algorithm.**

1. Introduction

The aim of *physical database design* (PDD) is to define an appropriate set of access structures for a *database* (DB), offering a good compromise between mass storage occupation and time required for information retrieval and maintenance. PDD is strictly dependent upon the *database management system* (DBMS) target, so the designer has to take into account, among others, two basic aspects:

- the access structures supported by the DBMS; e.g., hash functions, links, inverted indexes, clustering of data, etc.;
- the strategies used in accessing the data; e.g., join algorithms, index intersection methods, etc.

PDD is always a complex task due to the large number of possible choices, even if the DBMS offers only a limited set of features.

The *index selection problem* (ISP) is a particularly important phase of PDD and consists of choosing the DB indexes to be created in order to globally minimize the response time for a given DB workload. In some cases this choice is constrained by the amount of memory available for storing the indexes. With appropriate assumptions, to be discussed later, ISP can be formulated as follows. We are given m *queries*, say $Q_1, \ldots, Q_m$, along with n candidate *indexes*, say $\mathcal{F}_1, \ldots, \mathcal{F}_n$. Each index $\mathcal{F}_j$ has an associated *maintenance cost*, c_j, and requires d_j memory units to be stored. Each query Q_i can access the DB data by utilizing at most one index, the corresponding *execution cost* being μ_i if no index is used, or γ_{ij} if index $\mathcal{F}_j$ is utilized. Let $g_{ij} := \max\{0, \mu_i - \gamma_{ij}\}$ be the *gain* for using index $\mathcal{F}_j$ for query Q_i. ISP then consists of selecting a subset $S^* \subseteq \{1, \ldots, n\}$ (where $j \in S^*$ means that index $\mathcal{F}_j$ has to be constructed and stored) such that $\sum_{j \in S^*} d_j$ does not exceed a given bound D on the memory available for the indexes. The objective is the minimization of the overall cost for answering all the queries, computed as $\sum_{i=1}^{m} \min\{u_i, \min_{j \in S^*} \gamma_{ij}\} + \sum_{j \in S^*} c_j$, or equivalently the maximization of the *net gain* $\sum_{i=1}^{m} \max\{g_{ij} : j \in S^*\} - \sum_{j \in S^*} c_j$.

ISP is an NP-hard optimization problem [9]. It can be viewed as a combinatorial optimization problem since its solution has to be chosen from among a finite number of possible configurations. Therefore, explicit and complete enumeration of all the possible index subsets is, in principle, a possible way to solve ISP. This method is however impractical in most cases, since the computational effort grows exponentially with the number of candidate indexes for selection.

Both heuristic and exact approaches for index selection in a relational DB environment have been proposed in the literature, see, e.g., [2], [3], [4], [5], [12], [13], [19], [21], and [22]. Solution methods and formalizations for the index selection problem for files can be found in [1], [15], [16], [17], [18], [27], [28], and [29].

Heuristic algorithms give an approximate solution to the problem, in the sense that they do not guarantee that the index subset they select is the one which minimizes the estimated execution cost of the DB workload. As noted in [13], in practical applications one is not really interested in finding the *optimal* solution to the problem, since its formulation is affected by some approximation. On the other hand, the designer is interested in finding a solution "not too far" from the optimum, and none of the proposed heuristic algorithms provides an effective way to estimate the difference between the given solution and the optimal one.

The exact approaches so far proposed in the literature suffer from the fact that they consist of complete enumeration of the possible index subsets, hence are very time consuming even for small problem instances.

Manuscript received July 24, 1992; revised Apr. 20, 1993.

A. Caprara and D. Maio are with the DEIS, University of Bologna, viale Risorgimento 2, 40136 Bologna, Italy; e-mail: alberto@pramet4.cineca.it and dmaio@deis.unibo.it.

M. Fischetti is with the DEI, University of Padova, via Gradenigo 6/A, 35131 Padova, Italy; e-mail: fisch@dei.unipd.it.

To order reprints of this article, e-mail: transactions@computer.org, and reference IEEECS Log Number K95084.

The aim of this paper is to show how our specific version of ISP, although NP-hard, can be solved exactly through a well-designed algorithm based on combinatorial optimization techniques. For another example of a combinatorial optimization approach to PDD, see [11], where the unified problem of record segmentation and access path selection is considered.

Experimental results prove that in practical cases the time required for the solution is much smaller than that necessary for complete enumeration of all the index combinations. In addition, the algorithm we propose provides a heuristic solution and a bound on the optimal solution which are improved during execution. Therefore, stopping the algorithm when the difference between these two values becomes sufficiently small ensures that the heuristically selected index set is a "good" one. We report successful computation of our exact algorithm on large-scale randomly-generated instances involving several hundreds of indexes and queries. These random instances are generated so as to closely simulate the structure and the workload of real-world DBs.

We also present a heuristic algorithm intended for very large-scale instances and capable of determining very tight approximate solutions in short computing time. The performances of this heuristic are evaluated on random instances involving several thousands of indexes and queries.

The paper is organized as follows. Section II gives the basic assumptions and definitions used in the sequel. In Section III, we give a mathematical formulation for ISP, and use it to derive a branch-and-bound algorithm for its optimal solution. We do not assume the reader is familiar with optimization techniques, so this section is intended to give an outline of the main ideas underlying the algorithm we have implemented. For a comprehensive introduction to combinatorial optimization the interested reader is addressed to, e.g., [25]. The mathematical formulation of Section III is improved upon in Section IV, where some numerical examples are also reported to illustrate the improvements. We also discuss the way the improved model, which contains a large number of constraints, can be solved in practice. Heuristic algorithms and reduction procedures are introduced in Section V. Section VI describes a decomposition approach for computing approximate solutions of very large-scale instances. In Section VII, extensive experimental results are reported. Finally, in Section VIII, we show how our approach can be generalized by relaxing some of the assumptions on which the model is based.

II. Basic Assumptions and Definitions

In the literature on index selection different assumptions as to the form of queries in the workload, the execution techniques for these queries, the cost functions, the physical placement of data, etc., lead to different formulations of the problem. In this respect the most general assumptions are made in [13], even if the use of a single index per table in accessing the data is still assumed. In this paper we deal with relational ISP on the following assumptions:

a) for each relation, the data allocation criterion is known;[1]
b) for every query, at most one index is used to retrieve the tuples of each relation involved (the one leading to the minimum cost);
c) every join query is performed by using separable join methods (for an exhaustive work on separability, see [30]);
d) for a data modification query (i.e., "insert," "delete," or "update" query), the cost for each index update does not depend upon the access path selected.

Hypotheses a), c), and d) correspond to a widely-used approach to the problem, which allows practical use of analytical formulas. Hypothesis b) is instead verified by particular classes of DBMS as, for example, those derived from IBM System R. As a consequence of b), our work differs from those considering index intersection methods, see, e.g., [2], [3], [19], and [30].

The mathematical model we propose in this paper has two attractive properties, namely:

- It is applicable whatever the structure of the indexes to be selected (for multiattribute indexes, the only requirement is that each index spans a set of attributes over a single relation);
- It poses no restriction on the set of cost functions that can be used to define the problem instance; therefore one can take into account, e.g., attribute dependences or nonuniform value distributions.

Some of the assumptions a)-d) can be relaxed without affecting the validity of the solution technique we propose. This will be discussed in Section VIII.

Our next step is characterization of an ISP instance, requiring the specification of DB statistics and workload. For every query different costs have to be evaluated, each associated with a possible access path. We introduce two definitions:

- *execution cost* of a single-table query: the global cost for the following operations
  - accessing the *tuple identifiers* (TIDs) of the index chosen to answer the query, if any;
  - accessing the tuples of the relation involved;
  - updating the tuples of the same relation, if required;
- *index maintenance* (or *update*) *cost* in a query: the cost for deleting, in an index, the TIDs corresponding to the old tuples and/or for inserting in the same index the TIDs corresponding to the new tuples.

Notice that also CPU costs can be considered, together with I/O costs, since no particular restriction on cost functions is made.

The main consequences of assumptions a)-d) above are the following:

a′) the physical placement of the data is known, hence it can be taken into account in cost evaluation;
b′) for every single-table query, the execution cost corresponding to each index set can be obtained from the exe-

1. Usually, a primary attribute is chosen and the file storing the relation is sorted according to the values of this attribute.

cution costs of the configurations having at most one index for the same table (see [13]);

c′) every join query can be substituted with an equivalent set of single-table queries, one for each relation involved in the join;

d′) the cost of each index update for a query can be computed independently of the other indexes selected.

We next describe the notation used throughout. Let

- m = number of single-table queries in the workload, obtained after substitution (c′);
- n = number of candidate indexes for selection;
- $M = \{1, \ldots, m\}$;
- $N = \{1, \ldots, n\}$;
- $\mathcal{Q}_i (i \in M)$ = ith query;
- $\mathcal{F}_j (j \in N)$ = jth candidate index;
- $g_{ij} (i \in M; j \in N)$ = gain for using index $\mathcal{F}_j$ for query $\mathcal{Q}_i$, with respect to the execution cost when no candidate index is selected; if this gain is negative, g_{ij} is set to 0 (notice that $g_{ij} = 0$ if $\mathcal{F}_j$ and $\mathcal{Q}_i$ are associated with different relations);
- $c_j (j \in N)$ = global maintenance cost of index $\mathcal{F}_j$, i.e., the weighted sum, involving every query $\mathcal{Q}_i$, of the maintenance costs of $\mathcal{F}_j$; in this sum the weights are equal to the query frequencies (c_j can also take into account the cost due to periodic reorganization of $\mathcal{F}_j$);
- $d_j (j \in N)$ = secondary storage space required by index $\mathcal{F}_j$;
- D = total amount of secondary memory available for the indexes.

To simplify notation, for $i \in M$ and $j \in N$ we will write "query i" and "index j" instead of "query $\mathcal{Q}_i$" and "index $\mathcal{F}_j$," respectively.

We assume that $d_j (j \in N)$ and D are positive integers, and $c_j > 0$ $(j \in N)$. In addition, for each query $i \in M$ let

$$J_i := \{j \in N : g_{ij} > 0\}$$

contain the indexes j that are candidates to answer query i. Symmetrically, for each index $j \in N$ let

$$I_j := \{i \in M : g_{ij} > 0\}$$

contain the queries i which can take advantage of the availability of index j.

ISP can therefore be formulated as follows.

ISP: *Find an index subset $S^* \subseteq N$ such that*

$$\sum_{j \in S^*} d_j \leq D \tag{1}$$

and

$$z(S^*) := \sum_{i \in M} \max\{g_{ij} : j \in S^*\} - \sum_{j \in S^*} c_j \tag{2}$$

is a maximum (where $\max \varnothing = 0$ is assumed).

ISP generalizes two well-studied combinatorial optimization problems: the *knapsack problem* (KP) and the *uncapacitated* (or *simple*) *plant location problem* (UPLP).

KP is formally defined as follows. We are given a set of n items, the jth of which has a *weight* d_j and a *profit* π_j, $j \in N = \{1, \ldots, n\}$. The problem, in its maximization form, consists of selecting a subset of items $S^* \subseteq N$ whose overall weight $\sum_{j \in S^*} d_j$ does not exceed a given *capacity* D, and such that the overall profit $\sum_{j \in S^*} \pi_j$ is maximized. Alternatively, KP can be formulated in minimization form as follows. Each item $j \in N$ has an associated *cost* (instead of profit) $\bar{\gamma}_j$. The problem consists of selecting an item subset $\bar{S} \subseteq N$ with $\sum_{j \in \bar{S}} d_j \geq \bar{D}$, where $\bar{D}$ is an assigned threshold, such that the overall cost $\sum_{j \in \bar{S}} \bar{\gamma}_j$ is minimized. The two formulations are equivalent, as one can easily see by defining $\bar{D} = \sum_{j \in N} d_j - D$ and $\bar{\gamma}_j = \pi_j$ for $j \in N$. This correspondence ensures $\bar{S} = N \setminus S^*$. For a comprehensive treatement of KP and related problems, see [24].

KP is a very special case of ISP, which arises when the sets $I_j (j \in N)$ are pairwise disjoint, i.e., when every query has a positive gain for one index only. Indeed, in this case the objective function (2) can be written, for any $S \subseteq N$, as $z(S) = \sum_{j \in S} \pi_j$, where the profits are defined as $\pi_j := \sum_{i \in I_j} g_{ij} - c_j$.

UPLP is defined as follows. We are given a set M of m clients, and a set N of n possible sites where plants can be located. Each client i, $i \in M$, can be served by at most one plant, whereas a plant can serve different clients. When client i is served by plant j, a profit g_{ij} is gained. Let c_j be the fixed cost incurred if opening a plant in site j. UPLP consists of choosing a subset S^* of sites in which the plants have to be opened, so as to maximize the difference between the overall gain and the fixed costs, i.e., the objective function $z(S^*) := \sum_{i \in M} \max\{g_{ij} : j \in S^*\} - \sum_{j \in S^*} c_j$.

Clearly, UPLP arises from ISP when the memory bound is not imposed, or when it does not affect the optimal ISP solution, i.e., when $D \geq \sum_{j \in \bar{S}} d_j$, where $\bar{S}$ is an index set which maximizes (2) in absence of constraint (1).

Both KP and UPLP are known to be NP-hard problems [14]; hence so is ISP, even when the memory constraint (1) is not imposed.

III. A Branch-and-Bound Algorithm for ISP

We next derive a mathematical formulation of the problem, suitable for computation. A *linear programming* (LP) problem is an optimization problem in which the objective function and all the constraints are linear in the unknown variables. This problem is known to be polynomially solvable [20]. A *0-1 integer linear programming* (0-1 ILP) problem is an LP problem amended by the additional requirement that the unknown variables must attain either the value 0 or 1. This problem is NP-hard [14].

We have chosen the following 0-1 ILP model for ISP. Let us define the decision variables:

$$y_j = \begin{cases} 1, & \text{if index } j \text{ is selected} \\ 0, & \text{otherwise,} \end{cases} \qquad j \in N$$

$$x_{ij} = \begin{cases} 1, & \text{if index } j \text{ is used to query } i \\ 0, & \text{otherwise,} \end{cases} \qquad i \in M;\ j \in J_i.$$

ISP then reads

$$z(ISP) := \max\left[\sum_{i \in M}\sum_{j \in J_i} g_{ij}x_{ij} - \sum_{j \in N} c_j y_j\right] \tag{3}$$

subject to

$$\sum_{j \in N} d_j y_j \le D \tag{4}$$

$$\sum_{j \in J_i} x_{ij} \le 1,\ i \in M \tag{5}$$

$$\sum_{i \in I_j} x_{ij} \le |I_j| y_j,\ j \in N \tag{6}$$

$$x_{ij} \in \{0, 1\},\ i \in M;\ j \in J_i \tag{7}$$

$$y_j \in \{0, 1\},\ j \in N. \tag{8}$$

Interpretation of the objective function (3) is immediate. Constraint (4) imposes the upper bound D on the total memory available to store the selected indexes. Constraints (5) allow at most one index to be used for each query, whereas the "logical" constraints (6) impose selection of the indexes j that are used for at least one query i (since $x_{ij} = 1$ for some query $i \in I_j$ forces $y_j = 1$). Finally, (7) and (8) require the decision variables to be 0-1 valued.

The remaining part of the section is devoted to the description of an effective solution technique for the 0-1 ILP model (3)-(8). Although any known technique for solving at optimality an NP-hard optimization problem such as ISP requires exponential computing time in the worst case, effective algorithms can often be designed which allow the solution of large-scale instances within short computing times. For example, very large instances of the so-called *traveling salesman problem*, involving up to more than 2,000 cities, have recently been solved to optimality through specialized algorithms [26].

We propose a *branch-and-bound* algorithm for ISP, based on the LP relaxation obtained from model (3)-(8) by weakening the binary constraints (7) and (8) into

$$0 \le x_{ij} \le 1,\ i \in M;\ j \in J_i \tag{7'}$$

$$0 \le y_j \le 1,\ j \in N. \tag{8'}$$

Let LPR be the resulting model, defined through (3)-(6) and (7')-(8'). Since LPR is an LP problem, it can be solved efficiently through, e.g., the Khachian algorithm [20]. In practice the simplex algorithm is generally used, although it has a worst-case exponential performance. Let (x^*, y^*) be an optimal solution to LPR, and z(LPR) its value. Since any feasible solution to ISP is a feasible solution to LPR as well, z(LPR) is an upper bound on z(ISP), the optimal ISP solution value. Moreover, if (x^*, y^*) happens to be 0-1 valued (i.e., if constraints (7) and (8), although not imposed, are satisfied) then it defines a feasible ISP solution which is guaranteed to be optimal since its value equals the upper bound. Assume now that (x^*, y^*) is not 0-1 valued. If $y_j^* \in \{0, 1\}$ for all $j \in N$, then again ISP has been solved[2] since one can always redefine the optimal x-variables x_{ij}^* so as to attain 0-1 values only, without changing the objective function value. This is obtained by setting $x_{ij}^* = 1$ for all (i, j) pairs such that $y_j^* = 1$ and $g_{ij} = \max\{g_{ih} : y_h^* = 1\}$ (in case more than one maximum exists, one is chosen arbitrarily).

Therefore, ISP is not solved only if fractional y-variables exist in the optimal solution to LPR. In this case, we select one such variable, say y_j, and *branch* on it by fixing its value to 0 or to 1, respectively. In this way we split the original ISP problem into two ISP subproblems: in the first, we fix $y_j = 0$, i.e., index j is no longer a candidate for selection; in the second we fix $y_j = 1$ and hence force index j to be selected.

For each subproblem the procedure is then iterated by solving the LPR defined by (3)-(6), (7'), and (8'), and amended by the variable-fixing constraints resulting from branching.

This *divide and conquer* solution strategy can be represented through a binary *branch-decision tree* in which every node corresponds to an ISP subproblem, and the edges to the variable-fixing constraints. For instance, in the example of Fig. 1 the root node of the tree, node 1, represents the original ISP problem. Assuming that y_{j_1} is the chosen fractional y-variable, one constructs the two ISP subproblems associated with the son nodes 2 and 3 by fixing, respectively, $y_{j_1} = 0$ and $y_{j_1} = 1$. The procedure is iterated on these two nodes. In the figure, y_{j_2} is the fractional variable chosen after solving the LPR relaxation of subproblem 2. Fixing its value to 0 or 1 then produces subproblems 4 and 5, respectively. Note that subproblem 4 has both y_{j_1} and y_{j_2} fixed to 0, whereas subproblem 5 has y_{j_1} fixed to 0 and y_{j_2} to 1.

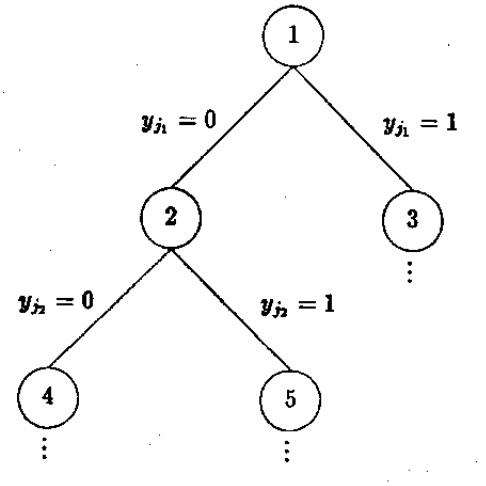


Fig. 1. A simple branch-decision tree.

2. This case can only arise when two or more selected indexes have the same gain for a query.

Let α be any given node of the branch-decision tree, and let ISP_α and LPR_α denote the ISP instance associated with α and its LPR relaxation, respectively. Moreover, let z^* be the value of the best ISP solution available (initially, z^* = value of a heuristic solution; see Section V). We say that node α is *fathomed* when ISP_α need not be further subdivided into smaller subproblems. This occurs in the following cases:

1) The optimal solution (x^α, y^α) of LPR_α has no fractional y-variables. In this case (x^α, y^α) defines an optimal solution for ISP_α as well. Therefore we compare its value, say $z(LPR_\alpha)$, with z^* and update z^* if an improved solution has been found.
2) $z(LPR_\alpha) \le z^*$: In this case the node is fathomed since there is no hope of finding an ISP solution better than z^* by further subdividing node α (recall that $z(LPR_\alpha)$ is an upper bound on the value of each feasible solution to ISP_α).

The branch-and-bound algorithm terminates when all the generated branch-decision nodes either produced son nodes, or were fathomed. Since there are at most $2^{n+1} - 1$ nodes, the algorithm terminates in a finite number of iterations. In practice, the number of branch-decision nodes that require to be generated is much smaller, since the fathoming criteria allow large portions of the tree to be pruned (see the computational analysis given in Section VII).

IV. Improving the Basic Model

The effectiveness of the branch-and-bound algorithm greatly depends upon the "quality" of LPR, the relaxation solved at each branch-decision node. Indeed, one is interested in finding 0-1 valued LPR solutions early in the branching process, as well as in having tight upper bounds $z(LPR_\alpha)$. In this perspective we look for new valid constraints to be added to the model (3)-(8), which are redundant as long as the binary constraints (7) and (8) are imposed, but capable of enhancing the "quality" of LPR. We will consider two classes of such additional constraints.

The first class is derived from the substructure of ISP leading to the uncapacitated plant location problem, and contains the valid constraints

$$x_{ij} \le y_j, j \in N; i \in I_j. \tag{9}$$

For each fixed index j, the $|I_j|$ constraints (9) impose $y_j = 1$ whenever there exists $i \in I_j$ with $x_{ij} = 1$. In this respect, constraints (9) play a role analogous to constraints (6). Note, however, that constraints (9) forbid the occurrence of fractional points which are not cut off by (6). Indeed, let x_{ij}^* and y_j^* be the optimal decision variables in the LPR relaxation without constraints (9) imposed. Because of the objective function (3) and the memory constraint (4), the y-variables will attain the minimum (nonnegative) value permitted by (6), i.e., $y_j^* = \frac{1}{|I_j|}\sum_{i \in I_j} x_{ij}^*$. On the other hand, when the additional constraints (9) are imposed, these values are no longer feasible, and will be replaced by the higher values $y_j^* = \max_{i \in I_j}\{x_{ij}^*\}$. To illustrate this point, let us consider the following small numerical instance of ISP proposed in [4], involving six queries and five indexes:

$$(g_{ij}) = \begin{bmatrix} 490 & 0 & 0 & 0 & 0 \\ 110 & 130 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 400 \\ 0 & 0 & 280 & 380 & 80 \\ 0 & 0 & 0 & 0 & 125 \end{bmatrix} \quad \begin{matrix} m = 6 \\ n = 5 \\ I_1 = \{1,2\} \\ I_2 = \{2\} \\ I_3 = \{4,5\} \\ I_4 = \{5\} \end{matrix}$$

$$(c_j) = [20 \quad 120 \quad 40 \quad 240 \quad 25] \qquad I_5 = \{4, 5, 6\}$$

$$(d_j) = [10 \quad 5 \quad 10 \quad 8 \quad 6]$$

$$D = 19.$$

For this instance the optimal ISP solution can be found by inspection, and selects index 1 for queries 1 and 2, and index 5 for queries 4, 5, and 6. The value of this solution is $z(ISP) = 1{,}160$. Solving the LPR relaxation without constraints (9) produces the upper bound $z(LPR) = 1{,}376$ on $z(ISP)$, and the optimal (fractional) solution

$$y_1^* = \frac{7}{10},\ y_2^* = 0,\ y_3^* = 1,\ y_4^* = 0,\ y_5^* = \frac{1}{3};$$

$$x_{11}^* = 1,\ x_{21}^* = \frac{4}{10},\ x_{43}^* = x_{53}^* = 1,\ x_{65}^* = 1,$$

and $x_{ij}^* = 0$ for all the remaining x-variables. Note that constraints (9) are violated by this solution for the (i, j) pairs (1, 1) and (6, 5). Adding constraints (9) then leads to the tighter upper bound $z(LPR) = 1{,}280$, which corresponds to the (still fractional) solution

$$y_1^* = 1,\ y_2^* = 0,\ y_3^* = \frac{3}{4},\ y_4^* = 0,\ y_5^* = \frac{1}{4};$$

$$x_{11}^* = x_{21}^* = 1,\ x_{43}^* = x_{53}^* = \frac{3}{4},\ x_{45}^* = x_{55}^* = x_{65}^* = \frac{1}{4},$$

and $x_{ij}^* = 0$ for all the remaining x-variables.

The second class of additional constraints we consider derives from the knapsack substructure of ISP, and contains the following *cover* inequalities (see, e.g., [25], Chapter II.2). Let $S \subseteq N$ be any index subset such that $\sum_{j \in S} d_j > D$. Because of the memory bound, not all the indexes in S can be selected; hence one has the valid constraint

$$\sum_{j \in S} y_j \le |S| - 1. \tag{10}$$

Again, imposing these additional constraints results in a strengthened LPR relaxation. As an illustration, in the numerical instance discussed above, the fractional solution available after imposing (9) violates the cover inequality (10) for $S = \{1, 3\}$. Adding this single constraint produces the new bound $z(LPR) = 1{,}182$, corresponding to the solution

$$y_1^* = 1,\ y_2^* = 0,\ y_3^* = 0,\ y_4^* = \frac{3}{8},\ y_5^* = 1;$$

$$x_{11}^* = x_{21}^* = 1,\ x_{54}^* = \frac{3}{8},\ x_{55}^* = \frac{5}{8},\ x_{45}^* = x_{65}^* = 1,$$

and $x_{ij}^* = 0$ for all the remaining x-variables. This new solution violates (10) for $S = \{1, 4, 5\}$. The addition of this new constraint leads to $z(\text{LPR}) = 1{,}160$ and

$$y_1^* = 1,\ y_2^* = 0,\ y_3^* = 0,\ y_4^* = 0,\ y_5^* = 1;$$

$$x_{11}^* = x_{21}^* = 1,\ x_{45}^* = x_{55}^* = x_{65}^* = 1,$$

and $x_{ij}^* = 0$ for all the remaining x-variables, representing the optimal solution to ISP. Notice that in this case no branching is needed since the solution of LPR for the initial ISP instance is 0-1 valued.

The above two classes of additional constraints contain a huge number of members, namely $\sum_{j \in N} |I_j| = O(nm)$ for class (9) and $O(2^n)$ for class (10). Therefore, embedding them all *explicitly* in the model is impractical even for small values of n. On the other hand, as shown in the previous example, only a few of them are really needed when a given numerical instance is considered. Thus one can use the following iterative scheme, in which the additional constraints (9) and (10) are added at run-time to the current LPR.

ALGORITHM ADDCUTS:

```
Set-up the initial LPR model with constraints
  (4)-(6), (7'), and (8') only;
repeat
  solve the current LPR model, and let (x*,y*)
  be its optimal solution;
  if there exist inequalities (9) which are
  violated by (x*,y*) then
    add them all to the current LPR
  else
    find, if any, a cover inequality (10) which
    is violated by (x*,y*), and add it to the
    current LPR
  end if
until no violated constraint has been found
end
```

In this way, at every iteration we add to the current LPR only those constraints which are "active," in the sense that they cut off the current LPR optimal solution (x^*, y^*) and hence strengthen the current relaxation. At the end of the algorithm, (x^*, y^*) is guaranteed to be optimal for the overall model that includes *all* the additional constraints (9) and (10), even though only a few of them have been explicitly added to the initial LPR model. Note that, in the above algorithm, a sequence of LP problems has to be solved, each obtained from the previous one by adding some new constraints. These LP problems (except the first one) need not be solved from scratch, but can take advantage from re-optimization techniques such as the dual simplex algorithm; see, e.g., [25]. Re-optimization is also of use during the branch-and-bound algorithm, where each subproblem is obtained from its father in the branch-decision tree, by adding a single variable-fixing constraint of the type $y_j = 0$ or $y_j = 1$.

The key point of the above scheme is the identification of violated constraints belonging to classes (9) and (10). This is an easy task for class (9), as it requires checking of $x_{ij}^* > y_j^*$ for each $i \in I_j$, $j \in N$. As to class (10), the exhaustive enumeration and check of all possible constraints is impractical due to their exponential number. Therefore a more sophisticated identification procedure for this class is needed, which we briefly outline in the sequel (for more details, see [10], and also [25], Chapter II.6).

We look for a subset S^* of N with $\sum_{j \in S^*} d_j > D$, which maximizes the *degree of violation*

$$\delta(S^*) := \sum_{j \in S^*} y_j^* - (|S^*| - 1):$$

if $\delta(S^*) \le 0$, then *all* the constraints (10) are satisfied by the current (x^*, y^*); otherwise, we have found the most violated constraint (10) to be added to the current LPR problem. Determination of S^* is itself an optimization problem, which can be formulated as the following instance of the knapsack problem:

$$\delta(S^*) = \max\left[\sum_{j \in N} y_j^* z_j - \sum_{j \in N} z_j + 1\right] = 1 - \min \sum_{j \in N} (1 - y_j^*) z_j \quad (11)$$

subject to

$$\sum_{j \in N} d_j z_j \ge D + 1 \quad (12)$$

$$z_j \in \{0, 1\},\ j \in N \quad (13)$$

where $z_j = 1$ if $j \in S^*$, $z_j = 0$ otherwise ($j \in N$). Notice that, in the above problem, each index j has a "cost" $\bar{\gamma}_j := 1 - y_j^* \ge 0$, and a "weight" $d_j > 0$. Therefore one can fix the z-variables associated with the 0-1 valued y-variables as follows:

1) fix $z_h = 1$ for all indexes h with $y_h^* = 1$, since including h in S^* does not change objective function (11) while it increases the left-hand side of (12);
2) fix $z_h = 0$ for all indexes h with $y_h^* = 0$; indeed, one has

$$\delta(S^*) = 1 - \sum_{j \in N \setminus \{h\}} \bar{\gamma}_j z_j - \bar{\gamma}_h z_h \le 1 - \bar{\gamma}_h z_h,$$

with $\bar{\gamma}_h = 1 - y_h^* = 1$, i.e., choosing $z_h = 1$ cannot lead to a set S^* with $\delta(S^*) > 0$.

As a result of the above variable-fixing, one is allowed to solve a "restricted" knapsack problem having the decision variables z_j associated with the indexes j with $0 < y_j^* < 1$ only (usually, a small fraction of n), with considerable saving in the computing time required to determine S^*.

V. HEURISTIC ALGORITHMS AND REDUCTION PROCEDURES

We next describe a heuristic solution computation that we apply at each node α of the branch-decision tree. This allows earlier fathomings in the decision tree, since a "good" solution to ISP is made available from the very beginning of the branch-and-bound algorithm.

Let (x^α, y^α) be the optimal LPR_α solution, as determined by applying the algorithm ADDCUTS of Section IV. If this solution is 0-1 valued, no heuristic search is needed. Otherwise, we construct a feasible (and hopefully almost optimal) solution to ISP_α by applying two heuristic algorithms: the best of the heuristic solutions obtained is then compared with the value z^* of the best available ISP solution, for possible updating of z^*.

Both heuristics we propose rely on the assumption that the values y_j^α give a measure of the "likelihood" of having index j selected in the optimal ISP_α solution. Let

$$J^1 := \{j : y_j^\alpha = 1\}$$

$$J^0 := \{j : y_j^\alpha = 0\}$$

$$J^F := \{j : 0 < y_j^\alpha < 1\}.$$

The first heuristic we propose, H1, selects all the indexes in J^1, as well as those belonging to the subset K of J^F that maximizes the "global likelihood" $\sum_{j \in K} y_j^\alpha$. This calls for solving of the following instance of the knapsack problem:

$$\max \sum_{j \in J^F} y_j^\alpha z_j$$

$$\sum_{j \in J^F} d_j z_j \le D - \sum_{j \in J^1} d_j$$

$$z_j \in \{0, 1\},\ j \in J^F,$$

where $z_j = 1$ if and only if $j \in K$. Since J^F typically contains only a few indexes, the solution of this problem is not computationally heavy in practice.

In our second heuristic, H2, we start by sorting the index set according to decreasing values of y_j^α, breaking ties by ranking the indexes with smaller d_j first. Let $j_1, \ldots, j_n$ be the resulting index permutation, with $y_{j_1}^\alpha \ge y_{j_2}^\alpha \ge \ldots \ge y_{j_n}^\alpha$. We determine the smallest h such that

$$\sum_{i=1}^{h} d_{j_i} > D,$$

where j_h plays the role of a "critical" index that cannot be stored when the previous indexes (which are more likely to be part of the optimal solution) are selected. Given a small integer parameter v (e.g., $v = 3$), we then construct a bucket B containing the $2v+1$ indexes $j_{h-v}, j_{h-v+1}, \ldots, j_h, j_{h+1}, \ldots, j_{h+v}$ and initialize a heuristic solution Q by taking all the indexes $j_1, j_2, \ldots, j_{h-v-1}$. Q is then enlarged by taking the subset B^* of B with

$$\sum_{j \in B^*} d_j \le D - \sum_{j \in Q} d_j \qquad (14)$$

that leads to the maximum objective function value, i.e., that maximizes

$$\sum_{i \in M} \max\{g_{ij} : j \in Q \cup B^*\} - \sum_{j \in Q \cup B^*} c_j. \qquad (15)$$

B^* is determined by enumerating all the subsets of B and evaluating, for each of them, (14) and (15). Notice that (15) can equivalently be written as

$$-\sum_{j \in Q} c_j + \left(\sum_{i \in M} \max\{g_i^q, \max\{g_{ij} : j \in B^*\}\} - \sum_{j \in B^*} c_j \right), \qquad (16)$$

where $g_i^q := \max\{g_{ij} : j \in Q\}$. Therefore, for any given $B^* \subseteq B$ one can evaluate (15) in $O(m|B^*|) = O(m)$ time, plus $O(m|Q|) = O(mn)$ time for the initial computation of the g_i^qs. Since there are $2^{2v+1} - 1$ possible subsets of B, the overall time complexity of H2 is $O(2^{2v}m + mn)$ time, plus $O(n \log n)$ for the initial sorting.

We next address reduction procedures to reduce the number of candidate indexes for selection. We will describe two simple criteria to determine whether, for any given index j, one necessarily has $y_j = 0$ in the optimal ISP solution. We define the following quantities:

- $G_j^{max} := \sum_{i \in I_j} g_{ij} - c_j$; G_j^{max} is the maximum increase in the objective function due to the selection of index j;
- $G_j^{min} := \sum_{i \in I_j^*} (g_{ij} - g_i^*) - c_j$, where

  $I_j^* := \{i \in M : g_{ij} > g_i^*\}$ and $g_i^* := \max\{g_{ih} : h \in N;\ h \ne j\}$;

  G_j^{min} gives the minimum increase in the objective function due to the selection of index j, since I_j^* contains the queries for which j performs better than any other index.

The first criterion we use allows the fixing of $y_j = 0$ for all indexes j such that

$$G_j^{max} \le 0, \qquad (17)$$

i.e., excluding from the selection all those indexes whose maintenance cost is larger than the maximum gain they can lead to.

The second criterion is based on the concept of dominance between two indexes. We say that a given index j is *dominated* by an index $h \ne j$ when any solution including index j can be improved by replacing j with h (or simply by removing j if h is already selected). Clearly, the existence of such an index h is a sufficient condition to set $y_j = 0$ (a tie-break rule has to be used in order to avoid "tautological" cycles arising when, e.g., j and h are identical). A very simple way to check dominance has been described in [5] and in [13], and consists of checking whether the conditions

$$d_j \ge d_h \qquad (18)$$

$$c_j \ge c_h \qquad (19)$$

$$g_{ij} \le g_{ih},\ i \in M \qquad (20)$$

hold. We propose a strengthening of conditions (18)-(20), based on the following considerations. Let us consider any given solution with $y_j = 1$. Two cases can occur:

1) $y_h = 1$: In this case the elimination of index j increases the objective function by at least the amount

$$\delta_j^- := c_j - \sum_{i \in \bar{I}} (g_{ij} - g_{ih}), \qquad (21)$$

where $\tilde{I} := \{i \in I_j : g_{ij} > g_{ih}\}$ contains the queries for which index j performs better than h (notice that δ_j^- depends upon both j and h). Indeed, in the worst case, j is replaced by h as the best selected index for all the queries in $\tilde{I}$.

2) $y_h = 0$: In this case,. and assuming $d_h \le d_j$, one can first select index h, with an increase in the objective function value of at least G_h^{min}, and then remove index j, with a further increase of at least δ_j^-.

Therefore, index j is dominated by h when both conditions (18) and

$$\min\{\delta_j^-, G_h^{min} + \delta_j^-\} \ge 0 \tag{22}$$

hold. It is not hard to see that j is not dominated by h when either $G_h^{max} < G_j^{max}$, or when j and h belong to different DB relations. These properties can be used to reduce the number of index pairs (j, h) that need to be considered for possible dominance.

A Pascal-like description of the overall reduction procedure we propose follows.

ALGORITHM REDUCTION

```
eliminate all the indexes j for which (17) is
  satisfied;
for each relation k in the DB do
  sort the set of n_k (say) indexes associated
  with relation k, according to decreasing val-
  ues of G_j^max, and let j_1, ..., j_nk be the result-
  ing index permutation;
  for each pair a, b ∈ {1, ..., n_k}, a < b, do
    let h := j_a and j := j_b;
    if conditions (18) and (22) hold then
      eliminate index j since it is dominated by h
    end if
  end for
end for
end
```

VI. Solving Very Large-Scale Instances

The branch-and-bound algorithm described in the previous sections is intended for medium/large-scale ISP instances, involving up to hundreds of queries and indexes. The algorithm performance on these instances is quite satisfactory, as shown through computational experiments in Section VII.

We next discuss how our approach can be used to tackle very large-scale instances, involving several hundreds of relations, i.e., thousands of queries and indexes. For these instances relaxation LPR of Section III (even without the additional constraints (9) and (10)) cannot be solved effectively, due to the excessive number of constraints and variables involved. One has therefore to resort to some kind of decomposition to "break" the overall ISP into a number of smaller subproblems to be solved independently of each other. Actually, this kind of decomposition arises quite naturally for ISP with no memory restriction imposed, each subproblem being associated with a different DB relation. Indeed, for each query the only candidate indexes correspond to attributes of the same relation. For example, in the numerical instance of Section IV two relations are involved, the first associated with queries 1, 2, and 3 and indexes 1 and 2, and the second with queries 4, 5, and 6 and indexes 3, 4, and 5. Therefore all the entries g_{ij} not contained in the "domain" of these two relations are equal to 0.

To be more specific, let p denote the number of relations in the DB, $P := \{1, \ldots, p\}$ their set, and M_k and N_k the set of queries and the set of indexes associated with relation k, respectively. The initial formulation (1) and (2) can then be rewritten as follows:

ISP (with no memory restriction): Find p index subsets $S_k^* \subseteq N_k$ $(k \in P)$ *such that*

$$z(S_1^* \cup \ldots \cup S_p^*) := \sum_{k \in P} z(S_k^*) := \sum_{k \in P} \left[\sum_{i \in M_k} \max\{g_{ij} : j \in S_k^*\} - \sum_{j \in S_k^*} c_j \right]$$

is a maximum.

Since the sets S_k^* can clearly be determined independently of each other, the overall ISP can be solved by applying, in turn, our branch-and-bound algorithm to each subproblem, say ISP_k, to determine the optimal set S_k^*. (Notice that every subproblem defines an instance of the uncapacitated plant location problem, see Section II, hence it is still NP-hard.) Moreover, the reduction procedure of Section V can be improved on when no memory bound is imposed, as condition (18) is no longer worth checking, whereas one can fix $y_h = 1$ whenever the addition of index h improves the objective function value, i.e., when $G_h^{min} > 0$.

Decomposition also arises when the memory bound D, although present, is not "tight," in that it does not affect the optimal ISP solution. This can happen in practice either because D is quite large, or because of large c_j costs for the indexes.

When the memory restriction is active, decomposition can still be obtained through *Lagrangian relaxation*, a technique to deal with hard optimization problems in which the removal of a small subset of constraints results in a much easier problem. (For a comprehensive description on the use of Lagrangian relaxation in combinatorial optimization see, e.g., [25].) In our case the approach consists of dropping constraint (4) in formulation (3)-(8), while modifying the objective function (3) so as to "penalize" the solutions that do not satisfy the memory constraint. This is obtained by adding the penalty term $D - \sum_{j \in N} d_j y_j$ to the objective function, weighted by a nonnegative parameter (*Lagrangian multiplier*) λ. The resulting *relaxed problem*, say R_λ, then reads

$$z(R_\lambda) := \max\left[\sum_{i \in M} \sum_{j \in J_i} g_{ij} x_{ij} - \sum_{j \in N} c_j y_j + \lambda \left(D - \sum_{j \in N} d_j y_j \right) \right]$$

$$= \lambda D + \max\left[\sum_{i \in M} \sum_{j \in J_i} g_{ij} x_{ij} - \sum_{j \in N} (c_j + \lambda d_j) y_j \right]$$

subject to (5)-(8). For any fixed $\lambda \ge 0$, R_λ defines an instance of ISP with no memory bound and with modified maintenance costs $c_j + \lambda d_j$. As such, R_λ decomposes into p smaller subproblems (one for each relation).

The main property of Lagrangian relaxation is that, for any $\lambda \geq 0$, $z(R_\lambda)$ gives an upper bound on the value $z(ISP)$ of the optimal ISP solution. Indeed, $z(R_\lambda)$ is computed over an enlarged domain, resulting from the removal of constraint (4), and with respect to an objective function attaining a higher value for all the feasible ISP solutions, since the added penalty term $\lambda(D - \sum_{j \in N} d_j y_j)$ is nonnegative for these solutions.

For large values of λ the modified maintenance costs, $c_j + \lambda d_j$, become sufficiently large to ensure that the optimal solution to R_λ, say (x^λ, y^λ), satisfies the memory constraint, thus defining an approximate ISP solution of value

$$LB(\lambda) := \sum_{i \in M} \sum_{j \in J_i} g_{ij} x_{ij}^\lambda - \sum_{j \in N} c_j y_j^\lambda .$$

Notice that $LB(\lambda)$ is only defined when $D - \sum_{j \in N} d_j y_j^\lambda \geq 0$. Since

$$z(ISP) \leq z(R_\lambda) = \sum_{i \in M} \sum_{j \in J_i} g_{ij} x_{ij}^\lambda - \sum_{j \in N} c_j y_j^\lambda + \lambda \left(D - \sum_{j \in N} d_j y_j^\lambda \right),$$

the approximation error $z(ISP) - LB(\lambda)$ cannot exceed

$$GAP(\lambda) := \lambda \left(D - \sum_{j \in N} d_j y_j^\lambda \right). \quad (24)$$

Therefore one is interested in choosing for λ the smallest nonnegative value such that $\sum_{j \in N} d_j y_j^\lambda \leq D$, thus obtaining an approximate solution that minimizes $GAP(\lambda)$. This value, say λ^*, can be found in an effective way by using an iterative technique akin to binary search: λ^* is first set to 0; then, iteratively, R_{λ^*} is solved, and λ^* is increased or decreased depending upon the sign of the quantity $D - \sum_{j \in N} d_j y_j^{\lambda^*}$. The procedure is stopped when λ^* is known with a sufficient precision.

Typically, λ^* is of the same order of magnitude as the ratio $z(ISP)/D$; hence, from (24), the estimated percentage gap $100 \cdot GAP(\lambda^*)/z(R_{\lambda^*})$ is of the same order of magnitude as the percentage of unused memory, defined as $100 \cdot (D - \sum_{j \in N} d_j y_j^{\lambda^*})/D$.

In order to illustrate how the above heuristic works, let us consider the sample instance reported in Section IV. Let $S_\lambda := \{j \in N : y_j^\lambda = 1\}$ denote the optimal index set for R_λ. The heuristic then computes:

λ	S_λ	$z(R_\lambda)$	$LB(\lambda)$	$D - \sum_{j \in N} d_j y_j^\lambda$
0	{1, 3, 5}	1,420	–	−7
100	∅	1,900	0	19
50	{1, 5}	1,310	1,160	3
25	{1, 3}	1,295	–	−1
38	{1, 3}	1,282	–	−1
44	{1, 5}	1,292	1,160	3
41	{1, 5}	1,283	1,160	3
40	{1, 5}	1,280	1,160	3
39	{1, 3}	1,281	–	−1

and returns (for $\lambda^* = 40$) the approximate solution $S_{\lambda^*} = \{1, 5\}$, of value $LB(\lambda^*) = 1{,}160$, as well as the upper bound $z(R_{\lambda^*}) = 1{,}280$ on the optimal value $z(ISP) = 1{,}160$. In this small instance the estimated percentage gap is quite large ($\approx 9.4\%$), since a large portion of the available memory ($\approx 16\%$) is not used by the selected indexes. For more realistic instances the gap is however much smaller, see Section VII.

VII. Experimental Results

The effectiveness of our branch-and-bound algorithm, as well as that of the heuristics of Sections V and VI, have been evaluated through computational experiments. All the proposed algorithms have been implemented in Fortran, and run on a Sun SPARC-2 workstation. As to the solution of the linear programming relaxations, we used Marsten's Fortran package XMP [23]. The solution of knapsack problems has been carried-out by using the Fortran routines given in [24].

Since we had no access to real-world ISP instances with a significantly large number of relations, we decided to validate the performance of our algorithm on random instances. These instances closely simulate the structure and the workload of real-world DBs, with overall statistics very close to those reported by [31] for a real-world relational DB. The random generation procedure is outlined in the Appendix.

We first analyzed the behavior of our branch-and-bound algorithm on instances involving up to 20 relations. Since each relation has, on average, 15 indexes and 25 queries, these instances involve up to approximately 300 indexes and 500 queries. Memory bound D has been set to a fixed fraction, μ, of the overall memory required to store the DB relations. We have considered three values for μ (μ = 10%, 20%, and 30%), and three values for the number of relations p (p = 10, 15, 20). For each of the nine resulting data sets, 10 random instances have been generated and solved.

The corresponding average (and, in parentheses, worst) results are reported in Table I. The table gives:

- the average (maximum) computing time, in CPU seconds, for the overall branch-and-bound algorithm;
- the average (maximum) number of explored nodes in the branching tree;

TABLE I
BRANCH-AND-BOUND ALGORITHM

p	μ	B&B time	B&B nodes	Root time	Root gap %
10	10%	636.3 (1889.6)	318 (791)	8.0 (10.9)	0.262 (0.628)
	20%	40.8 (50.8)	35 (55)	7.4 (10.3)	0.002 (0.024)
	30%	1.3 (3.3)	1 (1)	1.3 (3.3)	0.000 (0.000)
15	10%	638.4 (1452.7)	225 (471)	13.0 (18.5)	0.023 (0.083)
	20%	182.6 (401.3)	41 (87)	12.5 (23.0)	0.001 (0.001)
	30%	31.2 (95.3)	17 (67)	5.3 (12.0)	0.000 (0.000)
20	10%	2482.9(3894.4)	425 (807)	29.9 (58.0)	0.015 (0.038)
	20%	575.8 (1523.1)	127 (263)	9.6 (15.4)	0.001 (0.004)
	30%	32.3 (148.0)	6 (25)	3.8 (6.5)	0.000 (0.000)

Average (worst) results over 10 instances. Times are given in Sun SPARC-2 CPU seconds.

TABLE II
SIMPLIFIED BRANCH-AND-BOUND ALGORITHM S-BAB

p	μ	B&B time	B&B nodes	Root time	Root gap %
10	10%	>2106.5 (>2564.4)	>10000	0.9 (1.1)	13.80 (17.83)
	20%	>1988.5 (>2195.3)	>10000	0.7 (0.9)	3.17 (5.19)
	30%	>2310.3 (>2495.1)	>10000	0.7 (0.8)	1.76 (2.05)
15	10%	>3070.9 (>3348.6)	>10000	1.5 (2.0)	15.79 (19.30)
	20%	>2719.3 (>3025.7)	>10000	1.0 (1.2)	2.08 (3.29)
	30%	>3030.2 (>4465.8)	>10000	1.0 (1.1)	1.07 (1.57)
20	10%	>4429.8 (>4865.4)	>10000	2.8 (3.1)	14.03 (16.86)
	20%	>3565.4 (>4107.3)	>10000	1.5 (1.7)	3.83 (4.68)
	30%	>4397.0 (>5120.8)	>10000	1.4 (1.6)	1.03 (1.64)

Average (worst) results over 10 instances. For all instances S-BAB ran out of memory after 10,000 branch-decision nodes. Times are given in Sun SPARC-2 CPU seconds.

- the average (maximum) time spent at the root node of the branching tree, including that spent in the two heuristics described in Section V;
- the average (maximum) percentage gap := 100 · (UB − LB)/UB, computed at the root node, between the upper bound UB = z(LPR) of Section IV and the value LB of the best of the two heuristic solutions of Section V.

As expected, the algorithm exhibits better performance for higher values of μ, i.e., when the given memory bound is not too tight. This case, on the other hand, appears to be the most realistic one because allocating to the indexes less than 20% of the data memory seems too restrictive a choice in the design of a real-world DB. In all the cases, the CPU time spent to compute the optimal ISP solution is acceptable in the context of design. The reduction procedure of Section V was quite successful, as it removed (on average) more than 30% of the candidate indexes. As to the approximate solution and the upper bound computed at the root node, they are always very good, and require small computing time. Notice that the gap reported in the table is an upper bound on the *percentage* error of the approximate solution. Therefore, stopping the algorithm right after the root-node computation results in a very effective heuristic.

In order to evaluate the effect of the improvements to the LPR relaxation described in Section IV, we implemented a simplified branch-and-bound algorithm, S-BAB, obtained from our original branch-and-bound code by inhibiting the search for violated inequalities (9) and (10). Table II shows the performance of S-BAB on the same data-set as in Table I. For all instances S-BAB did not succeed in finding a provably optimal solution, and ran out of memory after having generated and solved more than 10,000 branch-decision nodes. Comparison of Tables I and II clearly demonstrates the effectiveness of having the additional constraints (9) and (10) present in the model.

We next tackled very large-scale instances involving up to 1,000 relations (i.e., up to 15,000 indexes and 25,000 queries) by using the heuristic approach of Section VI. Table III gives the corresponding average (worst) results, computed over 10 random instances. As in Table I, we report computing time, in Sun SPARC-2 CPU seconds, and percentage gap computed as $100 \cdot \mathrm{GAP}(\lambda^*)/z(\mathrm{R}_{\lambda^*})$, see (23) and (24). The table shows that the performances of the heuristic are very good, since very tight (and sometimes provably optimal) solutions are computed in short computing time. The computing time decreases when μ increases. Moreover, it grows almost linearly with the number of relations. This is not surprising, since decomposition breaks a p-relations ISP instance into p 1-relation subinstances. Therefore, even larger ISP instances can be solved by the heuristic within acceptable computing time.

VIII. EXTENSIONS AND CONCLUSIONS

In the previous sections we have derived a mathematical model and resolution algorithms for index selection, on assumptions (a)-(d) of Section II. In this section we briefly discuss the basic concepts underlying possible extensions of our 0-1 ILP model, that allow us to relax some of these assumptions.

TABLE III
LAGRANGIAN HEURISTIC

p	μ	Time		Gap%	
10	10%	3.2	(6.3)	0.346	(0.680)
	20%	1.5	(2.6)	0.074	(0.281)
	30%	0.5	(2.1)	0.001	(0.004)
15	10%	3.9	(6.2)	0.042	(0.165)
	20%	2.3	(4.7)	0.018	(0.045)
	30%	1.1	(1.9)	0.011	(0.044)
20	10%	6.0	(10.7)	0.021	(0.043)
	20%	3.0	(4.7)	0.007	(0.024)
	30%	1.7	(3.4)	0.017	(0.033)
50	10%	12.3	(16.8)	0.086	(0.268)
	20%	7.7	(9.1)	0.008	(0.016)
	30%	2.9	(7.0)	0.010	(0.039)
100	10%	27.8	(31.3)	0.033	(0.086)
	20%	16.4	(19.1)	0.007	(0.027)
	30%	9.3	(15.5)	0.016	(0.027)
500	10%	149.7	(173,5)	0.014	(0.039)
	20%	83.9	(90.5)	0.012	(0.028)
	30%	45.2	(74.5)	0.014	(0.024)
1,000	10%	296.6	(352.8)	0.017	(0.061)
	20%	150.1	(159.7)	0.006	(0.017)
	30%	86.9	(139.5)	0.015	(0.027)

Average (worst) results over 10 instances. Times are given in SUN SPARC-2 CPU seconds.

The simplifying assumption a) states that, in index selection, the primary access has already been chosen. This allows us to solve separately the two basic PDD problems, namely, data allocation and index selection, which would be intrinsecally mutually dependent (this so-called *two-step approach* has been described in [8]). In order to make an optimal choice of the primary access (e.g., hashing on an attribute, primary attribute with primary index, etc.) for the DB relations together with the secondary index set, the designer can operate as follows. First, the quantities g_{ij}, c_j, and d_j have to be computed also for every primary access of interest. Notice that, for a primary access j, c_j represents the additional global maintenance cost for the relation, and d_j represents the increase in the secondary storage space required by the relation, both with respect to the case in which no primary access is chosen. The 0-1 ILP model can then be generalized by introducing a y-variable (and the associated x-variables) for each primary access and for each secondary index, and by adding to the formulation (3)-(10) the additional constraints

$$\sum_{j \in NP_k} y_j \leq 1, k \in P \tag{25}$$

where P is the set of the DB relations, and NP_k the set of the considered primary accesses for relation k. These constraints require us to choose, at most, one primary access for each relation. The main limit of this formulation is that the physical placement of data is unknown when a problem instance is computed: this may result in some approximations in evaluating the quantities g_{ij} and c_j.

Relaxing assumption b) allows us to deal with the use of more than one index to access a relation during the execution of a query (e.g., in TID intersection list methods). In this case, when answering a query it is possible to use the index subset leading to the minimum execution cost. Relaxing assumption c), instead, allows us to deal with the use of nonseparable join algorithms (e.g., nested loop). In this case, when answering a join query it is possible to use the indexes (one, at most, for every relation in the join) that minimize the global cost of the join, the order of relation accesses depending upon *all* the indexes chosen. If both assumptions b) and c) are relaxed, it is possible to consider join queries performed by using more than one index per join relation, and by fixing the order of relation accesses after choosing the global set of indexes to be used. In each case above (b), c), or both, relaxed), one has to consider the use of a subset of indexes for answering a query, and the execution cost (or gain) of a query cannot be expressed as a linear function of the y-variables associated with the indexes (i.e., an index does not have a predefined gain for each query). In order to obtain again a 0-1 ILP formulation for the problem, let us indicate with K_i the family of the index subsets usable in query i, and with g_{ik} the gain of its kth element. Moreover, let us indicate with I_j the set of the queries i for which index j belongs to at least one subset $k \in K_i$, and with $K_{ij} \subseteq K_i$ the family of the possible index subsets for query i containing the index j. We associate a y-variable y_j with every index j, and an x-variable x_{ik} with every index subset $k \in K_i$. The new model is obtained from (3)-(10) by replacing sets J_i with sets K_i, and by substituting constraints (6) and (9) with the new constraints

$$\sum_{i \in I_j} \sum_{k \in K_{ij}} x_{ik} \leq |I_j| y_j, j \in N \tag{26}$$

and

$$\sum_{k \in K_{ij}} x_{ik} \leq y_j, j \in N, i \in I_j \tag{27}$$

respectively. In this formulation, the number of constraints is the same as in the initial model (3)-(10), while a larger number of x-variables is required.

In the cases in which assumption a) is eliminated together with one of the assumptions b) and c) (or both), a correct 0-1 ILP model can be obtained immediately by combining the two generalized models described above. We note that, in absence of the memory constraint, the decomposition property described in Section VI holds whenever separable join methods are used; therefore our approach to very large-scale problems is still usable when some of the assumptions a) and b) are relaxed.

The form of the models outlined above is similar to that of the initial model, therefore the branch-and-bound algorithm we propose in this paper is conceptually still valid for the extensions. This proves the generality of the 0-1 ILP approach to ISP. However, the presence of new types of constraints and the large number of variables in the new models require specific algorithms for an effective resolution, either exact or heuristic. Further research will follow two main directions:

- investigation of the possible extensions of the combinatorial optimization approach to PDD for object-oriented and knowledge base systems, whose importance for advanced application is increasing;
- new solution techniques for the extended mathematical formulations outlined in this section.

APPENDIX

We next outline the procedure we used to generate random ISP instances that closely simulate the structure and the workload of real-world DBs (see Section VII).

We first set-up the structure of the DB. Basically, this calls for defining:

- the number of relations, p (an input parameter);
- the number of tuples and attributes in each relation, defined as uniformly random integer in range [10000, 100000] and [5, 25], respectively;
- for each attribute:
  - the type (*integer* with probability 30%, and *string* with probability 70%);
  - the dimension in bytes (equal to 4 for integer attributes, and uniformly random in range [5, 50] for strings);
  - the number of possible values attained in the tuples of the relation (equal to the number of tuples with probability 30%, and uniformly random integer in range [1/10, 1/100] times the number of tuples with probability 70%);
  - the dimension d_j (in 4-Kbyte memory pages) of the corresponding index j, computed by assuming a B+-tree structure, 4-byte TIDs, and a padding factor equal to 70%;
- the dimension (in 4-Kbyte memory pages) of each relation, computed by assuming a padding factor equal to 90%;

We then specified the DB workload by defining, among others:

- the number of queries involved on each relation, defined as a uniformly random integer in range [10, 20];
- for each query:
  - the frequency, computed as a uniformly random integer in range [1, 10];
  - the type: "select," "update," "delete," and "insert" with probability 50%, 30%, 10%, and 10%, respectively;
  - for the "select" case, the number of the relations in the query (a random integer belonging to an exponential distribution with expected value 2); we treat each k-relation join as k single-table queries;
  - for the "update" case, the number of updated attributes (a random integer belonging to an exponential distribution with expected value 2), and the corresponding indexes (chosen among the indexes of the relation according to an exponential distribution, the first indexes of the relation being more likely to be chosen);
  - the number of index-processable predicates (a random integer belonging to an exponential distribution with expected value 5), and the corresponding indexes (chosen among the indexes of the relation according to an exponential distribution);
  - the number of attribute values satisfying each predicate, computed as a random integer belonging to an exponential distribution with expected value 10.

The above probability distributions led to realistic instances, with overall statistics very close to those reported by [31] for a real-world relational DB.

After generating the above data, we computed the quantities g_{ij} and c_j that appear in the objective function of our optimization problem. We made the following assumptions:

- The time required by a query only depends upon I/O operations, hence can be expressed by the number of pages transferred to/from the main memory;
- The attribute values are independent of each other, and uniformly distributed among the tuples of each relation;
- The number of pages containing T tuples in a relation stored in P pages is given by the widely-used Cardenas' formula ([6]):

$$\Phi(P, T) = P\left(1-\left(1-\frac{1}{P}\right)^{T}\right);$$

- Each index maintenance cost in a query is computed assuming an *unordered scan* of the index (see [13]).

Moreover, two separable join methods for executing join queries were considered: *sort merge* (see [30]) and *nested loop* with a predefined nesting level for each join relation (see [5]).

Fuller details on the instance generation are given in [7]. We stress the fact that the assumptions we made in the instance generation can be changed without affecting the validity of the solution approach we propose (provided conditions a) to d) of Section II hold).

ACKNOWLEDGMENTS

Paolo Ciaccia significantly contributed to the development of the instance generation program. Paolo Tiberio and Paolo Toth read earlier versions of this paper and provided useful suggestions. Moreover, we are very grateful to the referees for their work.

This work has been partially supported by MURST 40%, and by CNR (the Italian National Council of Research).

REFERENCES

[1] H.D. Anderson and P.B. Berra, "Minimum cost selection of secondary indexes for formatted files," *ACM Trans. Database Systems*, vol. 2, pp. 68-90, 1977.

[2] E. Barcucci, R. Pinzani, and R. Sprugnoli, "Optimal selection of secondary indexes," *IEEE Trans. Software Engineering*, vol. 16, pp. 32-38, 1990.

[3] E. Barcucci, A. Chiuderi, R. Pinzani, and E. Rodella, "Optimal selection of secondary indices in relational databases," *Proc. Int'l Symp. Computer and Information Sciences I*, M. Baray and B. Özgüç, eds., pp. 157-168. Elsevier, 1991.

[4] R. Bonanno, D. Maio, and P. Tiberio, "An approximation algorithm for secondary index selection in relational database physical design," *The Computer J.*, vol. 28, pp. 398-405, 1985.

[5] F. Bonfatti, D. Maio, and P. Tiberio, "A separability-based method for secondary index selection in physical database design," *Methodology and Tools for Data Base Design*, S. Ceri, ed., pp. 149-160. North Holland, 1983.

[6] A.F. Cardenas, "Analysis and performance of inverted data base structures," *Comm. ACM*, vol. 18, pp. 253-263, 1983.

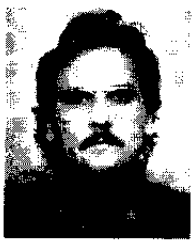
[7] A. Caprara, "Un algoritmo esatto per la selezione di indici secondari nel progetto fisico relazionale," Tesi di Laurea, DEIS, Univ. of Bologna, 1991.

[8] P. Ciaccia and D. Maio, "On the optimal ordering of multiple-field tables," Technical Report CIOC-CNR 86, Univ. of Bologna, 1992.

[9] D. Comer, "The difficulty of optimum index selection," *ACM Trans. Database Systems*, vol. 3, pp. 440-445, 1978.
[10] H.P. Crowder, E.L. Johnson, and M.W. Padberg, "Solving large-scale zero-one linear programming problems," *Operations Research*, vol. 31, pp. 803-834, 1983.
[11] P. De, J.S. Park, and H. Pirkul, "An integrated model of record segmentation and access path selection for databases," *Information Systems*, vol. 13, pp. 13-30, 1988.
[12] B.J. Falkowski, "Comments on an optimal set of indices for a relational database," *IEEE Trans. Software Engineering*, vol. 18, pp. 168-171, 1992.
[13] S. Finkelstein, M. Schkolnick, and P. Tiberio, "Physical database design for relational databases," *ACM Trans. Database Systems*, vol. 13, pp. 91-128, 1988.
[14] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
[15] M. Hatzopoulos and J.G. Kollias, "On the optimal selection of multilist database structures," *IEEE Trans. Software Engineering*, vol. 10, pp. 681-687, 1984.
[16] M. Hatzopoulos and J.G. Kollias, "On the selection of a reduced set of indexes," *The Computer J.*, vol. 28, pp. 406-408, 1985.
[17] W.F. King, "On the selection of indices for a file," IBM Research Report RJ 1641, San Jose, Calif., 1974.
[18] J.G. Kollias, "A heuristic approach for determining the optimal degree of file inversion," *Information Systems*, vol. 4, pp. 307-318, 1979.
[19] M.Y.L. Ip, L.V. Saxton, and V.V. Raghavan, "On the selection of an optimal set of indexes," *IEEE Trans. Software Engineering*, vol. 9, pp. 135-143, 1983.
[20] L.G. Khachian, "A polynomial algorithm in linear programming," *Soviet Mathematics Doklady*, vol. 20, pp. 191-194, 1979.
[21] D. Maio, C. Sartori, and M.R. Scalas, "Architecture of a physical design tool for relational DBMSs," *Computer Aided Data Base Design*, A. Albano, V. De Antonellis, and A. Di Leva, eds., pp. 115-130. North Holland, 1985.
[22] D. Maio, C. Sartori, and M.R. Scalas, "A modular user oriented decision support for physical database design," *Decision Support Systems*, vol. 3, pp. 155-163, 1987.
[23] R. Marsten, "The design of the XMP linear programming library," *ACM Trans. Math. Software*, vol. 7, pp. 481-497, 1981.
[24] S. Martello and P. Toth, *Knapsack Problems—Algorithms and Computer Implementations*. Reading, Mass.: John Wiley & Sons, 1990.
[25] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*. Reading, Mass.: John Wiley & Sons, 1988.
[26] M.W. Padberg and G. Rinaldi, "Optimization of a 532-city travelling salesman problem," *Operations Research Letters*, vol. 6, pp. 1-8, 1987.
[27] A. Putkonen, "On the selection of access path in inverted database organization," *Information Systems*, vol. 4, pp. 219-225, 1979.
[28] M. Schkolnick, "The optimal selection of secondary indices for files," *Information Systems*, vol. 1, pp. 141-146, 1975.
[29] M. Stonebraker, "The choice of partial inversions and combined indices," *Int'l J. Computer and Information Sciences*, vol. 3, pp. 167-188, 1974.
[30] K.-Y. Whang, G. Wiederhold, and D. Sagalowicz, "Separability—An approach to physical database design," *IEEE Trans. Computers*, vol. 33, pp. 209-222, 1984.
[31] P.S. Yu, M.-S. Chen, H.-U. Heiss, and S. Lee, "On workload characterization of relational database environments," ***IEEE Trans. Software Engineering***, vol. 18, pp. 347-355, 1992.

Alberto Caprara received the Laurea in electronic engineering from the University of Bologna in 1991. For his Laurea he was awarded the Francini Prize in 1992 from the A.E.I. (Italian Electrical and Electronic Association). He is currently a PhD student at the Department of Electronics, Computer Science, and Systems of the University of Bologna. His research interests include branch-and-bound algorithms, polyhedral combinatorics, and physical database design.

Matteo Fischetti received the Laurea in electronic engineering from the University of Bologna in 1982. In 1987, he received a PhD in system engineering, and for his PhD thesis, he was awarded the first prize of the Transportation Section of the ORSA (Operations Research Society of America). He is currently a professor in the Department of Electronics and Computer Science of the University of Udine. His research interests include combinatorial optimization theory and applications, integer programming, and polyhedral combinatorics.



Dario Maio received the Laurea in electronic engineering from the University of Bologna in 1975. From 1978 to 1980, he received a fellowship from the C.N.R. (Italian National Research Council) for participation in the Air Traffic Control Project. He is currently a professor in the Department of Electronics, Computer Science, and Systems of the University of Bologna. His research interests include distributed computer systems, computer performance evaluation, database design, information systems, neural networks, and autonomous systems. He is an IEEE member.

Information about The IEEE Computer Society and its services is available by calling our Customer Service Department at **+1-714-82-8380** or by e-mail: **cs.books@computer.org**. The society maintains its home page on the World Wide Web at **http://www.computer.org**

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE • 1946-1996