An International Journal
computers & mathematics with applications

PERGAMON Computers and Mathematics with Applications 37 (1999) 111-120

Near Optimal Multiple Choice Index Selection for Relational Databases

T. I. GÜNDEM
Computer Engineering Department, Boğaziçi University
80815 Bebek, Istanbul, Turkey
gundem@boun.edu.tr

(Received September 1997; accepted October 1997)

Abstract—Index selection for relational databases is an important issue which has been researched quite extensively [1-5]. In the literature, in index selection algorithms for relational databases, at most one index is considered as a candidate for each attribute of a relation. However, it is possible that more than one different type of indexes with different storage space requirements may be present as candidates for an attribute. Also, it may not be possible to eliminate locally all but one of the candidate indexes for an attribute due to different benefits and storage space requirements associated with the candidates. Thus, the algorithms available in the literature for optimal index selection may not be used when there are multiple candidates for each attribute and there is a need for a global optimization algorithm in which at most one index can be selected from a set of candidate indexes for an attribute. The problem of index selection in the presence of multiple candidate indexes for each attribute (which we call the multiple choice index selection problem) has not been addressed in the literature. In this paper, we present the multiple choice index selection problem, show that it is NP-hard, and present an algorithm which gives an approximately optimal solution within a user specified error bound in a logarithmic time order. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords—Database systems, Knapsack problem, Discrete optimization.

NOMENCLATURE

U	the set of relations in the database
R_i	a relation name
F	the set of all files for U
f_i	a file in F
$g(R_i)$	a function giving us the file associated with a relation R_i
A	the set of all attributes in U
a_i	an attribute
$\alpha(R_i)$	a function giving us the set of attributes of a relation R_i
UP	the set of frequently used updates
u_i	an update operation
S	the set of frequently used selections
s_i	a selection operation
$\mathrm{fr}(o_i)$	a function giving us the expected frequency of occurrence of a selection or an update
$r(o_i)$	a function giving us the relation associated with a selection or an update
$a(o_i)$	a function giving us the set of attributes associated with a selection or an update
X	the set of all candidate indexes
x_i	an index
b_i	the benefit of an index x_i
st_i	the storage space requirements of an index x_i

I would like to thank M. Karayel for beneficial discussions on combinatorial optimizations.

PII: S0898-1221(98)00256-9

$\mathrm{at}(x_i)$	a function giving us the set of attributes associated with x_i
$\mathrm{idx}(a_j)$	the equivalence class of candidate indexes associated with an attribute a_j
$\mathrm{AS}(x_i)$	the set of selections associated with x_i
$\mathrm{AU}(x_i)$	the set of updates associated with x_i
$\mathrm{sb}_{k,i}$	the benefit contributed by s_k to x_i
$\mathrm{ub}_{k,i}$	the benefit contributed by u_k to x_i
M	the maximum storage space reserved for the indexes for U

1. INTRODUCTION

The efficiency of a relational database depends on the physical level of the database. Creating indexes for the attributes of the relations may significantly contribute to the efficiency of the physical level of a relational database. Although an index on an attribute A of a relation R expedites the processing of selection operations on attribute A, it slows down the processing of modification operations associated with attribute A and insertion and deletion operations on relation R. Thus, it must be determined whether or not the advantages of an index overweigh its disadvantages. Also, even if the advantages overweigh the disadvantages for all the attributes, it may not be possible to create an index for every attribute of a relation because of the maximum storage space constraint of the system. Thus, there is a need for a global optimization in selecting the set of indexes for a relational database that will make the selection and update operations on the database as efficient as possible while satisfying the maximum storage space constraint. The problem of selecting an optimal set of indexes has been shown to be NP-hard and studied by many researchers [1–5]. However, there is an important dimension to this problem which occurs in real life database design but has been overlooked in the studies that appear in the literature.

What has been overlooked is the possibility of having more than one index with different storage space requirements as candidates for an attribute of a relation. For example, we may have the following alternatives for an attribute:

(i) a B-tree [6],
(ii) an ordinary index [6],
(iii) a set of partial indexes [7–9],
(iv) both a set of partial indexes and a regular index.

Other possible alternative index types may also be considered. Since each alternative may have a different benefit and storage space requirement, it may not be possible to choose one of them locally as the "best" and use an index selection algorithm available in the literature. There is a need for a global optimization in which more than one alternative are considered for an attribute. We call this the multiple choice index selection problem.

In this paper, we show that the multiple choice index selection problem is NP-hard and present an algorithm which gives an approximate solution within a user specified error tolerance in a logarithmic time order. The methodology we present in this paper chooses the set of indexes I_x (from a given set of candidates) that minimizes (within a given error tolerance) the cost of processing the given set of selection and update operations without violating the specified maximum storage space constraint. In I_x, there may be at most one index for an attribute. In the methodology presented, we assume that the alternatives for each attribute and the frequently used selections and updates (i.e., insertion, deletion, and modification operations) are given. For the problem of choosing alternative indexes for the attributes of a given database from a set of usage patterns of the database, one may refer to [10].

In the next section, we present the basic concepts. In Section 3, we give the cost functions. In Section 4, the multiple choice index selection problem is formally presented and an approximate solution to it is given. The last section contains the conclusions.

2. PRELIMINARY CONCEPTS

Let us assume we have a database with a set of relations $U = \{R_1, R_2, \ldots, R_r\}$ and a set of storage structures $F = \{f_1, f_2, \ldots, f_y\}$ where the relations in U are stored. Let $g(R_i)$ give us the file where the tuples of R_i are stored. Let $A = \{a_1, a_2, \dots, a_t\}$ be the set of all attributes associated with the relations in U. Let $at(R_i) = \{a_{i1}, a_{i2}, \dots, a_{ik}\}$, where $k \geq 1$ and each $a_{il} \in A$, give us the attributes associated with R_i.

Let the set of frequently used updates (i.e., insertions, deletions, and modifications) be specified by the set $\mathrm{UP} = \{u_1, u_2, \dots, u_p\}$. We also have a set of frequently used selections $S = \{s_1, s_2, \dots, s_k\}$. Selection operation is one of the most frequently used operations in query processing. We assume that each selection operation is associated with just one attribute of a base relation. A selection query referring to n attributes of a relation may be represented by n selection operations, each of which refers to just one attribute. Each update operation is associated with a number of attributes, whereas each selection operation is associated with just one attribute. Each insertion or deletion operation on relation R_i is associated with all the attributes of relation R_i since the indexes on all the attributes of R_i have to be updated when an insertion or deletion is applied to R_i. A modification operation is associated with only the attributes whose values it modifies.

Let o_i represent an s_i or u_i. Associated with each o_i, we have the following properties.

(i) $\mathrm{fr}(o_i)$ gives us the expected frequency of occurrence of o_i.

(ii) $r(o_i)$ gives us the relation associated with a selection or an update.

(iii) $a(o_i)$ gives us the set of attributes associated with a selection or an update. In case o_i represents a selection, $a(o_i)$ contains just one attribute.

We assume that the set of frequently incurred updates UP and selections S are provided by the database administrator.

Let $X = \{x_1, x_2, \dots, x_{xt}\}$ be the set of all indexes. Let $\mathrm{at}(x_i)$ give us the attribute associated with index x_i. For each attribute a_j of each relation R_i, let $\mathrm{idx}(a_j) = \{x_{j1}, x_{j2}, \dots, x_{jk}\}$, where each $x_{ji} \in X$, give us the set of candidate indexes for attribute a_j. Let us call $\mathrm{idx}(a_j)$ the *equivalence class* associated with a_j. The set of all equivalence classes partitions the set of all indexes X for the attributes in A. Each candidate index x_k may be an index such as a B-tree, an ordinary index, a set of partial indexes, or any combination of these. For example, x_k may be a set of partial indexes and an ordinary index on an attribute. The set of candidate access structures $\mathrm{idx}(a_i)$ (i.e., the equivalence class) for each a_i has to be provided by the database administrator. In obtaining different sorts of candidate indexes of any complexity, one may make use of the work done by Gündem [10]. Each index structure x_i has a benefit b_i and a secondary storage space requirement st_i. The benefit of an index is related to the cost of processing all the associated selections and updates using the index and is going to be defined in the next section.

3. COMPUTATION OF BENEFITS AND COSTS

METHOD 1. The benefit b_i associated with an index structure x_i is computed as follows.

(i.a) For x_i, find the set of associated selections. $\mathrm{AS}(x_i) = \{s_k \mid \mathrm{at}(x_i) \in a(s_k)\}$. For all the indexes in equivalence class $\mathrm{idx}((\mathrm{at}(x_i))$, we have the same AS. That is to say, $\mathrm{AS}(x_i) = \mathrm{AS}(x_j)$, if $\mathrm{at}(x_i) = \mathrm{at}(x_j)$, where $i \neq j$.

(i.b) For x_i, find the set of associated updates. $\mathrm{AU}(x_i) = \{u_i \mid \mathrm{at}(x_i) \in a(u_i)\}$. For all the indexes in equivalence class $\mathrm{idx}(\mathrm{at}(x_i))$, we have the same AU.

(ii.a) For each $s_k \in \mathrm{AS}(x_i)$, compute benefit $\mathrm{sb}_{k,i}$ contributed by s_k to x_i. For s_k, the benefit gives us the gain in the cost of processing s_k due to the presence of index x_i:

$$\mathrm{sb}_{k,i} = \mathrm{fr}(s_k) * (\mathrm{sfc}_k - \mathrm{sic}_{k,i}),$$

where $\mathrm{fr}(s_k)$ is the frequency of occurrence of s_k, $\mathrm{sic}_{k,i}$ is the cost of processing s_k using using the index x_i, and sfc_k is the cost of processing s_k in the absence of index x_i (i.e., just using the file $g(r(s_i))$ where the tuples of the relation $r(s_i)$ are stored).

(ii.b) For each $u_k \in \mathrm{AU}(x_i)$, compute benefit $\mathrm{ub}_{k,i}$ contributed by u_k to x_i. For u_k, the benefit gives us the gain in the cost of processing u_k due to the presence of index x_i. The benefit

$\mathrm{ub}_{k,i}$ is usually negative and represents the burden in processing u_k due to the presence of the index x_i. This is due to the fact that in updating the relation, the index x_i has to be updated too:

$$\mathrm{ub}_{k,i} = \mathrm{fr}(u_k) * (\mathrm{ufc}_k - \mathrm{uic}_{k,i}),$$

where $\mathrm{fr}(u_k)$ is the frequency of occurrence of u_k, $\mathrm{uic}_{k,i}$ is the cost of processing u_k using the index x_i, and ufc_k is the cost of processing u_k without using the index x_i (i.e., just using the file $g(r(u_k))$ where the tuples of the relation $r(u_k)$ are stored).

(iii) Compute b_i using formula (1) given in the following (in Definition 1). ∎

For a selection s_k, usually $\mathrm{sfc}_k > \mathrm{sic}_{k,i}$ and for an update u_k, usually $\mathrm{ufc}_k < \mathrm{uic}_{k,i}$. In computing ufc_k (sfc_k) or $\mathrm{uic}_{k,i}$ ($\mathrm{sic}_{k,i}$), the number of pages accessed in processing u_k (s_k) are computed. In a database management system, there is an algorithm alg_u (alg_s) that is used to process an update (selection) operation in the presence of an index. The number of pages accessed during the execution of algorithm alg_u (alg_s) consitutes $\mathrm{uic}_{k,i}$ ($\mathrm{sic}_{k,i}$). There is another algorithm alg'_u (alg'_s) that is used to process an update (selection) operation in the absence of an index. The number of pages accessed during the execution of algorithm alg'_u (alg'_s) constitutes ufc_k (sfc_k). The methodology that we present is independent of any specific database management system or specific algorithms alg_u, alg_s, alg'_u, and alg'_s. Thus, we do not and cannot give detailed formulas for the computation of the costs $\mathrm{uic}_{k,i}$, $\mathrm{sic}_{k,i}$, ufc_k, and sfc_k. In computing these costs, one can make use of the profusion of work done in the literature such as [11,12].

DEFINITION 1. *The total benefit b_i of an index x_i is given by*

$$b_i = \left(\sum_{k\in \mathrm{uf}_i} \mathrm{ub}_{k,i}\right) + \left(\sum_{k\in \mathrm{sf}_i} \mathrm{sb}_{k,i}\right) - \mathrm{cb}_i, \tag{1}$$

where cb_i is the cost of building the index x_i, $\mathrm{uf}_i = \{k \mid \mathrm{at}(x_i) \in a(u_k)\}$ gives us the identifiers of update operations associated with x_i, and $\mathrm{sf}_i = \{k \mid \mathrm{at}(x_i) \in a(s_k)\}$ gives us the identifiers of selection operations associated with x_i. ∎

DEFINITION 2. *A disjoint set of indexes I_x for the relations in U is a set of indexes such that for any pair of indexes x_i and x_j in I_x, where $i \neq j$, $\mathrm{at}(x_i) \neq \mathrm{at}(x_j)$ and $I_x \subseteq X$.* ∎

LEMMA 1. *In a disjoint set of indexes I_x for the relations in U, there can be at most one index from each equivalence class.*

PROOF. Associated with each equivalence class $\mathrm{idx}(a_i)$, there is an attribute a_i. For the equivalence $\mathrm{idx}(a_i)$, if $x_i \in \mathrm{idx}(a_i)$ and $x_j \in \mathrm{idx}(a_i)$, then $\mathrm{at}(x_i) = \mathrm{at}(x_j)$. Thus, both x_i and x_j cannot be in I_x from Definition 2. ∎

ELIMINATION 1. Eliminate all x_j in a disjoint set of indexes I_x if $b_j \leq 0$.

THEOREM 1. *Given a disjoint set of indexes I_x (to which Elimination 1 is applied) for a relational database of relations U, the total cost of processing all the updates in* UP *and all the selections in S in the presence of the indexes in I_x plus the total cost of building all the indexes in I_x is given by T, defined as follows. Let*

$$\mathrm{FC} = \mathrm{UFC} + \mathrm{SFC},$$

where

$$\mathrm{UFC} = \sum_{i\in\{i \mid (\exists u_i)(u_i \in \mathrm{UP})\}} \mathrm{fr}(u_i) * \mathrm{ufc}_i$$

and

$$\mathrm{SFC} = \sum_{i\in\{i \mid (\exists s_i)(s_i \in S)\}} \mathrm{fr}(s_i) * \mathrm{sfc}_i. \tag{2}$$

FC *gives us the total cost of processing all the updates and selections in the absence of any index. Let*

$$B = \sum_{i \in \mathrm{LL} = \{i \mid (\exists x_i)(x_i \in I_x)\}} b_i. \tag{3}$$

B gives us the total benefits of all the indexes in I_x.

$$T = \mathrm{FC} - B. \tag{4}$$

PROOF. The proof is constructive. We are going to show that the total cost of processing (i) all the selections in S and that of (ii) all the updates in UP in the presence of the indexes in I_x plus the total cost of (iii) building all the indexes in I_x are included in T and nothing else is.

(i) Selections. We are going to show that the total cost of processing all the selections in S are included in T.

Let $\mathrm{ATL} = \{\mathrm{atr} \mid \mathrm{atr} \in A \wedge (\exists x_i)(x_i \in I_x \wedge \mathrm{at}(x_i) = \mathrm{atr})\}$. ATL gives the set of attributes on which there are indexes from the set I_x.

CASE 1. For any selection s_k, if $a(s_k) \cap \mathrm{ATL} = \emptyset$, then the cost of processing s_k is $\mathrm{fr}(s_k) * \mathrm{sfc}_k$ and is included in SFC which can be seen from the definition of SFC, formula (2).

CASE 2. For any selection whose associated attribute $a(s_k)$ is the same as that of an index in I_x (i.e., $a(s_k) \cap \mathrm{ATL} \neq \emptyset$), consider the following. We know that for a selection s_k, $a(s_k)$ has one attribute. Let $a(s_k) = \{a_h\}$. Since $a(s_k) \cap \mathrm{ATL} \neq \emptyset$, there must be an index, say $x_l \in I_x$, such that $\mathrm{at}(x_l) = a_h$. The cost of processing s_k is $\mathrm{fr}(s_k) * \mathrm{sic}_{k,l} = (\mathrm{fr}(s_k) * \mathrm{sfc}_k) - \mathrm{sb}_{k,l}$. The cost $\mathrm{fr}(s_k) * \mathrm{sfc}_k$ is included in FC since s_k is in S. The cost $\mathrm{sb}_{k,l}$ is included in b_l due to formula (1). B includes b_l because $x_i \in I_x$. In formula (4), we have $-B$.

(ii) Updates. We are going to show that the total cost of processing all the updates in UP in the presence of the indexes in I_x are included in T.

Let uc_k designate the total cost of processing an update u_k. $\mathrm{uc}_k = \mathrm{uc1}_k - \mathrm{uc2}_k$, where

$$\mathrm{uc1}_k = \mathrm{fr}(u_k) * \mathrm{ufc}_k \qquad \text{and}$$
$$\mathrm{uc2}_k = \sum_{j \in L} \mathrm{ub}_{k,j},$$

where $L = \{j \mid (\exists x_j)(x_j \in I_x) \wedge \mathrm{at}(x_j) \in a(u_k)\}$.

The cost $\mathrm{uc1}_k$ represents the cost of processing u_k using the file $g(r(u_k))$ where the relation associated with u_k is stored. It is included in UFC in T. The cost $\mathrm{uc2}_k$ represents the effect of the indexes in I_x whose attributes are included in the set of attributes associated with u_k. This effect is usually negative. Thus, the presence of indexes usually adds to the cost of processing updates. Each $\mathrm{ub}_{k,j}$ in $\mathrm{uc2}_k$ is included in b_j of formula (1) since $\mathrm{at}(x_j)$ is in $a(u_k)$. And each b_j such that j is in L is included in B because L is a subset of LL in formula (3). Thus, $\mathrm{uc2}_k$ is included in T. We conclude that the cost of processing u_k (i.e., uc_k) is included in T.

(iii) There is a b_i associated with each x_i in I_x. A b_i includes the cost of building the access structure x_i as can be seen from formula (1). The b_i associated with each x_i in I_x is included in B. Thus, the cost of building all the indexes are included in T.

Only the costs specified above in (i), (ii), and (iii) are included in T and nothing else. To see this, let us examine the costs in UFC, SFC, and B which comprise T.

(a) UFC: Let us assume that u_h is an update not in UP whose ufc_h is included in UFC. Then by the definition of UFC, h has to be in $\{i \mid (\exists u_i)(u_i \in \mathrm{UP})\}$ which means that u_h is in UP. This is a contradiction. Thus, UFC does not include ufc_h, if u_h is not in UP.

(b) SFC: Let us assume that s_h is an update not in S but its cost of processing, sfc_h, is included in SF. Then by the definition of SFC, h has to be in $\{i \mid (\exists s_i)(s_i \in S)\}$ which means that s_h is in S. This is a contradiction. Thus, SFC does not include sfc_h, if s_h is not in S.

(c) B: By the definition of the formula for B (formula (3)), B includes b_i only if x_i is in I_x. Consider the following for each b_i.

   (i) Let us assume b_i includes $\mathrm{sb}_{k,i}$ for a selection s_k whose attribute $a(s_k)$ is different from the associated attribute of x_i. But we can see from formula (1) that if b_i includes $\mathrm{sb}_{k,i}$, then $\mathrm{at}(x_i) \in a(s_k)$, which is a contradiction. Thus, we can conclude that b_i does not include $\mathrm{sb}_{k,i}$ for a selection s_k, if $\mathrm{at}(x_i) \notin a(s_k)$.

   (ii) Let us assume that b_i includes $\mathrm{ub}_{k,i}$ for an update u_k whose associated attribute set $a(u_k)$ does not include the associated attribute of x_i. But since b_i includes $\mathrm{ub}_{k,i}$, then from formula (1) we see that $\mathrm{at}(x_i) \in a(u_k)$, which is a contradiction. Thus, we can conclude that b_i does not include $\mathrm{ub}_{k,i}$ for an update u_k, if $\mathrm{at}(x_i) \notin a(u_k)$.

This completes the proof of Theorem 1. ∎

4. OPTIMIZATION PROBLEM FOR OBTAINING THE OPTIMAL DISJOINT SET OF INDEXES

In index selection problems, there is a storage space constraint. The total storage space requirements of all the indexes for a database cannot be greater than a constant, M. The multiple choice index selection problem can be stated as follows. Given a set of equivalence classes of indexes associated with the attributes of the relations in U, find a disjoint set of indexes (over all possible disjoint sets of indexes) that minimizes the total cost of processing all the updates and selections in UP and S, respectively, and satisfies the storage space constraint. Formally, the problem is given in Problem 1.

PROBLEM 1. Given a set of equivalence classes of indexes (at most one equivalence class, $\mathrm{idx}(a_i)$, for each attribute a_i) and the amount of maximum storage space M reserved for the indexes in a database, find a disjoint set of indexes I_x over all possible disjoint sets of indexes, such that

(i) $T = \mathrm{FC} - B$ (as given by formula (4)) has its minimum value for the set of all possible disjoint sets of indexes (i.e., the total cost of processing all the selections and updates in S and U, respectively, is minimized from Theorem 1), and

(ii) $\sum_{i\in\{i\,|\,x_i\in I_x\}} \mathrm{st}_i \le M$,

(i.e., the total storage space requirements of the indexes in I_x is less than M). ∎

LEMMA 2. *In Problem 1, T has its minimum value when B in formula (4) has its maximum value.*

PROOF. It is straightforward as can be seen from formula (4). In formula (2), FC, the cost of processing selections and updates due to the file structures, is fixed. (We do not change the file structures associated with the relations in U during the methodology.) In formula (4), B is present due to indexes. Depending on the indexes in a disjoint set of indexes, B changes, as can be seen from formula (3). Thus, when we find the disjoint set of indexes that makes B have its maximum value, then T has the minimum possible value for any disjoint set of indexes for the problem. ∎

PROBLEM 2. It is the same as Problem 1 with the condition (i) replaced by the following:

(i) $\sum_{i\in\{i\,|\,x_i\in I_x\}} b_i$ has its maximum value for the set of all possible disjoint sets of indexes. ∎

THEOREM 3. *A solution to Problem 1 is also a solution to Problem 2 and vice versa.*

PROOF. It follows from Lemma 2. ∎

DEFINITION 3. *The multiple choice 0-1 knapsack optimization problem is defined as follows. Given a set of n objects $XX = \{xx_1, xx_2, \dots, xx_n\}$, where each object xx_i has a benefit b_i and a* *weight w_i; a maximum weight capacity MM and m equivalence classes where each equivalence class e_i has a set of objects,*

$$\text{maximize} \qquad \sum_{i \in \{i \mid xx_i \in XX\}} b_i * a_i, \tag{5}$$

subject to the constraints

$$\text{(i)} \qquad \sum_{j \in \{j \mid xx_j \in XX\}} w_j * a_j \leq M,$$

$$\text{(ii)} \qquad \sum_{j \in \{j \mid xx_j \in e_i\}} a_j \leq 1, \qquad i = 1, 2, \ldots, m,$$

$$\text{(iii)} \qquad a_j \in \{0, 1\}, \qquad j = 1, 2, \ldots, n.$$

It is known that the multiple choice 0-1 knapsack optimization problem is NP-hard, since its subset, the 0-1 knapsack optimization problem, is NP-hard [13]. When there is at most one object in each equivalence class, the multiple choice 0-1 knapsack problem becomes the same as the 0-1 knapsack optimization problem.

THEOREM 4. *Problem 2 is NP-hard.*

PROOF. In the proof, we will obtain in polynomial time an instance, IP2, of Problem 2 from an instance, IKS, of the multiple choice 0-1 knapsack problem such that from the solution of IP2, we can determine in polynomial time the solution to IKS.

Let IKS be the instance specified in Definition 3. We obtain IP2 by the following conversions: an object xx_i is converted into in index x_i; benefit b_i of an object xx_i is converted into the benefit b_i of an index x_i; weight w_i of an object xx_i is converted into the storage space requirement st_i of an index x_i; maximum weight capacity MM is converted into maximum storage space reserved for indexes M; and an equivalence class e_i of objects is converted into an equivalence class $\mathrm{idx}(a_i)$ of indexes associated with attribute a_i such that if xx_l is in e_i, then x_l is in $\mathrm{idx}(a_i)$.

Let I_x be the disjoint set of indexes that is a solution of IP2. The solution for IKS is obtained as follows. Go over the set of objects in XX. For each object xx_i in XX, if the corresponding x_i is in I_x, then $a_i = 1$, otherwise $a_i = 0$. Now we show that this is a solution to IKS.

The constraint (i) in Definition 3 is satisfied because the condition (ii) in Problem 2 is satisfied. The constraint (ii) in Definition 3 is satisfied as one can see from Lemma 1 which states that in a disjoint set of indexes there is at most one index from each equivalence class. We see that formula (5) is maximized because the condition (i) in Problem 2 is maximized. ∎

Since Problem 2 is NP-hard, we will give an approximate solution within a user specified error bound. The approximate solution that we will present is based on an approximate solution to the multiple choice 0-1 knapsack optimization problem.

METHOD 2.

1. Convert an instance, IP2, of Problem 2 into an instance, IKS, of the multiple choice 0-1 knapsack optimization problem by the following conversions: an index x_i is converted into an object xx_i; the benefit b_i of an index x_i is converted into the benefit b_i of an object xx_i; the storage space requirement st_i of an index x_i is converted into the weight w_i of object xx_i; the maximum storage space reserved for indexes M is converted into the maximum weight capacity MM, and an equivalence class $\mathrm{idx}(a_i)$ of indexes is converted into an equivalence class e_i of objects such that if x_l is in $\mathrm{idx}(a_i)$, then xx_l is in e_i.
2. Solve IKS using the fully polynomial time approximation algorithm given by Lawler in [14].
3. Using the solution to IKS, obtain a solution to IP2. The solution to IP2 is the disjoint set of indexes I_x which is obtained as follows. If $a_j = 1$ in IKS, then x_j is in I_x. Otherwise, x_j is not in I_x. ∎

For a total of n objects (or indexes) and m equivalence classes, Lawler's algorithm [14] gives an approximate solution to the multiple choice 0-1 knapsack optimization problem in time order $O(n\log(n) + m * n/\varepsilon)$ and space order $O(n + m^2/\varepsilon)$ for a given accuracy $\varepsilon > 0$. That is, if P^* is the optimal solution and P is the solution we obtain for a given ε using the approximation algorithm given by Lawler, then $P^* - P \leq \varepsilon P$. This approximate solution is desirable because of its reasonable time and space requirements. Additionally, since the frequencies of updates and insertions are only expected statistical values, an accuracy of ε is permissible in index selection problems.

THEOREM 5. *Method 2 finds an approximate solution to Problem 2.*

PROOF. Let AKS be the approximate solution to IKS obtained at Step 2 of Method 2. AKS must satisfy the constraints in Definition 3. The fact that AKS satisfies constraints (ii) and (iii) implies that I_x obtained at Step 3 of Method 2 is indeed a disjoint set of indexes as elaborated in the following. Due to constraints (ii) and (iii), any two objects in AKS, say xx_j and xx_k such that $a_j = a_k = 1$, must be from two different equivalence classes, say e_z and e_y, respectively. Thus, the indexes x_z and x_y (corresponding to e_z and e_y, respectively) are associated with two different equivalent classes of indexes $\mathrm{idx}(a_z)$ and $\mathrm{idx}(a_y)$, respectively, as can be seen from the conversions at Step 1 of Method 2. Since x_z and x_y are from two different equivalence classes, $\mathrm{at}(x_j) \neq \mathrm{at}(x_k)$, by the definition of equivalence classes of indexes. By Definition 2, I_x is a disjoint set of indexes.

It is simple to show that since AKS satisfies the constraints (i) and (iii), the condition (ii) in Problem 2 is satisfied by I_x for IP2.

It is also simple to show that since AKS maximizes the formula (5) for an accuracy of ε, the condition (i) in Problem 2 is also maximized with the same degree of accuracy. In fact, for AKS and I_x obtained in Method 2, we have the following equivalence:

$$\sum_{i\in\{i \mid xx_i \in XX\}} b_i * a_i = \sum_{k\in\{i \mid x_i \in I_x\}} b_k. \qquad \blacksquare$$

In some cases, it is beneficial to apply the following elimination to each equivalence class before Method 2. The application of the following elimination may help decrease the number of candidate index structures depending on their benefits and storage space requirements.

ELIMINATION 2. For an equivalence class $\mathrm{idx}(a_i)$, if we have two indexes x_k and x_l in $\mathrm{idx}(a_i)$ such that $\mathrm{st}_k \geq \mathrm{st}_l$ and $b_l \geq b_k$, then eliminate x_k from the equivalence class $\mathrm{idx}(a_i)$.

THEOREM 6. *Indexes eliminated by the application of Elimination 2 to each equivalence class do not prevent an optimal solution from being computed for an instance of Problem 2.*

PROOF. Let IPL be an instance of Problem 2 and I_x be an optimal solution to it. Let x_k be an index eliminated by Elimination 2. Let us consider the following cases.

(a) If an index x_k is eliminated because $\mathrm{st}_k \geq \mathrm{st}_l$ and $b_l > b_k$, then x_k cannot be in an optimal solution to Problem 2. Let us assume that x_k is in optimal solution I_y. Then replace x_k with x_l, and the formula in condition (i) of Problem 2 has a higher value than that for I_y. This means that I_y is not optimal, which is a contradiction. Thus, x_k cannot be in an optimal solution.

(b) If an index x_k is eliminated because $\mathrm{st}_k = \mathrm{st}_l$ and $b_l = b_k$, then x_k may be in an optimal solution to Problem 2. But now instead of x_k, we have x_l in an optimal solution.

(c) If an index x_k is eliminated because $\mathrm{st}_k > \mathrm{st}_l$ and $b_l = b_k$, then x_k may or may not be in an optimal solution to Problem 2. Now we have x_l in an optimal solution instead of x_k. Conditions (i) and (ii) in Problem 2 are still satisfied and an optimal solution is not prevented. $\blacksquare$

We can use the algorithm whose summary is given in the following to apply Elimination 2 to each equivalence class in $(n \log n) + n$ time order for n indexes.

ALGORITHM 1.

1. Sort all of the indexes according to increasing storage space, st_i. Those that have the same storage space are sorted according to increasing benefit, b_i.
2. Store the maximum benefit associated with each equivalence class in a separate data structure. Initially equate them all to 0.
3. Starting with the first index in the sorted list, repeat the following until after the last element in the list is read.

- Read the benefit, b_i, of the next index x_i. Let the maximum benefit so far for the equivalence class $\mathrm{idx}(\mathrm{at}(x_i))$ be $\max(\mathrm{idx}(\mathrm{at}(x_i)))$. Eliminate x_i, if $b_i \leq \max(\mathrm{idx}(\mathrm{at}(x_i)))$. (This elimination is according to Elimination 2.) If not, $\max(\mathrm{idx}(\mathrm{at}(x_i))) = b_i$. ∎

The worst case time complexity of all the methods presented so far may be summarized as follows. Let us assume we have a total n indexes, m equivalence classes, and an error tolerance of ε is specified. $|S|$ and $|U|$ be the number of selections and updates, respectively.

- Method 1 for benefit computations takes $O(n * |S| * |U|)$.
- Elimination 1 takes $O(n)$.
- Method 2 for optimization takes $O(n \log(n) + m * n/\varepsilon)$.
- Algorithm 1 for applying Elimination 2 takes $O(n \log n)$.

The worst case time complexity of the whole methodology is $O((n*|S|*|U|)+(n \log(n)+m*n/\varepsilon))$.

5. CONCLUSIONS

Index selection is an important problem as far as the efficiency of relational databases are considered. In index selection problems in the literature, only one index is considered as candidate for each attribute. However, it is likely that more than one different indexes of various type, storage space requirement, and benefit may be present as candidates for an attribute, and it may not be possible to eliminate locally all but one. Thus, it may not be possible to use the index selection algorithms presented in the literature.

In this paper, we consider the problem of index selection for relational databases in the presence of multiple candidates with different benefits and storage space requirements. We show that the problem is NP-hard. We present a methodology that finds a fully polynomial time approximation to the problem. In the methodology that we present, we first compute the benefits associated with candidates from the given set of commonly used selections and updates on the database. Then we apply the optimization algorithm to find a subset of the candidate indexes that minimizes the cost of processing the selections and the updates within a user given error tolerance subject to the maximum storage space constraint and to the condition that at most one candidate is selected for each attribute. Candidates are determined by the database administrator. A candidate may be a combination of different types of indexes for an attribute (for example, a set of partial indexes and a B-tree on the same attribute). We also present an algorithm to possibly eliminate (without effecting the result of the global optimization) some of the candidates associated with an attribute locally before the global optimization is applied.

Approximately optimal solutions obtained by our methodology are permissible in index selection problems since the given selections and updates on the database are only expected values. However, the user is able to change the error tolerance to suit his/her needs. The implementation of the benefit computations and cost functions for a specific system are given by Sahin [15]. The major contributions of the paper can be summarized as follows. The methodology presented gives a solution to the index selection problem when more than one candidate is present for each attribute. The solution given to the global optimization is fully polynomial time approximation and not a heuristic.

REFERENCES

1. M.Y.L. Ip, L.V. Saxton and V.V. Raghavan, On the selection of an optimal set of indexes, *IEEE Transactions on Software Engineering* **SE-9** (9), 135–143 (March 1983).
2. B. Falkowski, Comments on an optimal set of indexes for a relational database, *IEEE Transactions on Software Engineering* **SE-18** (2), 168–171 (February 1992).
3. R. Bonanna, D. Maio and P. Tiberio, An approximation algorithm for secondary index selection in relational database physical design, *The Computer Journal* **28** (4), 398–404 (1985).
4. E. Barcucci, R. Pinzani and R. Sprugnoli, Optimal selection of secondary indexes, *IEEE Transactions on Software Engineering* **SE-16** (1), 32–38 (January 1990).
5. M. Frank, E.R. Omiecinski and S.B. Navathe, Adaptive and automated index selection in RDBM's, *Proceedings of the 3^{rd} International Conference on Extending Database Technology*, March 1992, Vienna, Austria, pp. 277–292, Springer-Verlag.
6. J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Vol. 1, Computer Science Press, Maryland, (1988).
7. M. Stonebraker, The case for partial indexes, *ACM Sigmod Record* **18** (4), 4–12 (December 1989).
8. P. Seshadri and A. Swami, Generalized partial indexes, *Proceedings of the Eleventh International Conference on Data Engineering*, March 6–10, 1995, Taiwan, pp. 420–427.
9. C. Sartori and M.R. Scalas, Partial indexing for nonuniform data distributions in relational DBMS's, *IEEE Transactions on Knowledge and Data Engineering* **6** (3), 420–429 (June 1994).
10. T.I. Gündem, Fundamentals of the metamorphosis of access specifications into simple and complex access structures and a model of complex access structures, *International Journal of Systems Research and Information Science* **7**, 169–187 (1997).
11. K.Y. Whang, G. Wiederhold and D. Sagalowitz, Estimating block accesses database organizations: A closed non-iterative formula, *Communications of the ACM* **26** (11) (1983).
12. T.Y. Cheung, A statistical model for estimating the number of records in a relational database, *Information Processing Letters* **15** (3) (1982).
13. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Maryland, (1978).
14. E.L. Lawler, Fast approximation algorithms for knapsack problems, *Mathematics of Operations Research* **4** (4), 339–356 (November 1979).
15. S. Sahin, Database optimization for internal storage structures, Project No. R9 92K, Computer Engineering Department, Boğaziçi University, Istanbul, Turkey, (1992).