

On the selection of secondary indices in relational databases

Sunil Choenni^{a,*}, Henk M. Blanken^a and Thiel Chang^b

^aUniversity of Twente, Department of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands

^bAdministration Office: G.A.K., Department of Research and Development, P.O. Box 8300, 1005 CA Amsterdam, The Netherlands

Received 11 November 1992

Revised 21 June 1993

Abstract

An important problem in the physical design of databases is the selection of secondary indices. In general, this problem cannot be solved in an optimal way due to the complexity of the selection process. Often use is made of heuristics such as the well-known ADD and DROP algorithms. In this paper it will be shown that frequently used cost functions can be classified as super- or submodular functions. For these functions several mathematical properties have been derived which reduce the complexity of the index selection problem. These properties will be used to develop a tool for physical database design and also give a mathematical foundation for the success of the before-mentioned ADD and DROP algorithms.

Keywords. Physical database design; secondary index selection; ADD and DROP algorithms; supermodular functions; submodular functions.

1. Introduction

Physical database design is an important step in designing databases and aims to generate efficient storage structures for the data. One of the choices that has to be made is that of selecting efficient access paths. Most relational database systems provide indices as one of their main access paths. Indices can be considered as auxiliary files that allow to retrieve tuples satisfying certain selection predicates without having to examine the whole relation. On the other hand, updating the database causes an index to be updated to remain consistent with the new database state. So, an index speeds up retrieval and slows down maintenance.

In general two types of indices can be distinguished, namely primary and secondary indices [8]. In the case of a primary index, the tuples in the relation are ordered on the indexed attribute. This is not the case for a secondary index; from now on we concentrate on secondary indices.

A relational database consists of many stored relations and each stored relation can have many secondary indices. The index set of a (relational) database is the set of indices that are selected for the database. A *cost function* estimates the cost of processing a workload for a database with a given index set. In this cost function the benefits and the drawbacks of

* Corresponding author.

indices are taken into account. The problem of secondary index selection, further denoted as the SIS problem, is NP-complete as demonstrated in [10]. The problem for a single relation can be solved by examining 2^n possible index sets, in which n represents the number of candidate indices in the database. The aim is of course to find the optimal (or a near-optimal) index set without having to examine all sets [1]. Much research has been devoted to the SIS problem using analytical approaches [6, 16, 21], heuristic approaches [5, 11, 13, 24] or combinations of both approaches [2, 3]. The cost function in the approaches differs on two main points. First, some of the approaches use at most one index per relation to process an operation [5, 6, 11] while others use all usable indices [2, 3, 13, 16, 24]. Second, some approaches concern the SIS problem for a single relation [13, 16] while others concern the SIS problem for more than one relation [3, 5, 6, 11, 21].

Whang et al. [21, 23] proposes the so-called separability approach. In this approach, which can be applied if the DBMS satisfies certain conditions, the SIS problem for the whole database is split up into many, smaller problems, namely one SIS problem per relation. After determining the optimal set of indices for each relation, the union of these sets will be the solution for the whole database. With this approach we have to inspect $\sum_{i=1}^r 2^{n_i}$, in which r is the number of relations in the database and n_i is the number of candidate indices for the i th relation. So, the complexity has been reduced from $\prod_{i=1}^r 2^{n_i}$ to $\sum_{i=1}^r 2^{n_i}$. From now on we concentrate on the SIS problem for a relation. It is important to note that in general even for this restricted case, exhaustive search is not a feasible solution.

An important part of a cost function is the so-called *retrieval function* which estimates the number of page accesses needed to fetch the tuples addressed by an ordered list of pointers. Schkolnick [16] uses a simple but unrealistic retrieval function in his cost function leading to the nice property of 'regularity'. On the basis of experiments Schkolnick estimates that the regularity property decreases the complexity to $2^{\sqrt{n}} * \log n$ on the average, in which n is again the number of candidate indices.

Barucci et al. [2, 3] adopt a more realistic retrieval function and show that their cost function also satisfies by approximation the regularity property. They solve the SIS problem by utilizing regularity combined with the ADD and DROP algorithms. The ADD and DROP algorithms use heuristics and are described in [24].

Until now we discussed approaches that consider the total load on the database as a whole. An optimal index set for the whole load is searched for. Another approach is followed in [15]. There an ideal storage structure is determined for each load operation separately. Such a storage structure includes an index set. Then these storage structures are combined to get a storage structure which will be profitable for the total load on the database.

In this paper we describe super- and submodular functions and their mathematical properties. Supermodularity appears to be equivalent to the before-mentioned regularity. We consider some generally accepted retrieval functions and show that the corresponding cost functions are a combination of *super- and submodular functions*. Based on mathematical properties we develop an algorithm that reduces the complexity in optimizing the cost function. In a rough sense this algorithm is a partly generalization of [15]. Instead of single operations, we consider groups of operations and this reduces the complexity of combination steps. It is certainly not claimed that the algorithm is sufficient to solve the SIS problem in general, but it is a step in the direction of the optimal solution. In this paper also attention is paid to the ADD and DROP algorithms. It appears that the mathematical properties of submodular functions give these algorithms a sound, mathematical foundation. So we show that justifying these algorithms by considering supermodular functions, as is done by Barucci et al., is mathematically not correct.

The organization of the paper is as follows. In Section 2 we present some assumptions

related to the SIS problem and derive a general cost function. In the cost function, a retrieval function plays a part. In Section 3 we describe the behaviour of the retrieval function under influence of an increasing number of indices. This behaviour introduces the concepts of super- and submodularity which are defined more precisely in Section 4. We derive relevant, mathematical properties for these functions. Then we establish the relationship between these functions and the SIS problem and show how the properties can be applied in optimizing pure super- or submodular functions. The general cost function introduced in Section 2 gives rise to three specific cost functions depending on the chosen retrieval function. In Section 5 it will be proved that the specific cost functions are not pure super- or submodular but behave sometimes as a super- and sometimes as a submodular function. Section 6 describes an algorithm to utilize the derived properties of super- and submodular functions in solving the SIS problem. In Section 7 it is shown that submodular cost functions provide a foundation for the DROP and ADD algorithms. Section 8 concludes the paper and gives directions for further research.

2. General cost function

In this paper we will deal with relational databases. A relation R is defined over some attributes, such as, $\alpha_1, \alpha_2, \dots, \alpha_n$, and is a subset of the Cartesian product $dom(\alpha_1) \times dom(\alpha_2) \times \dots \times dom(\alpha_n)$. $Dom(\alpha_j)$ is the set of values assumed by the attribute α_j . Databases are stored on an external paged memory. Many relational database management systems (DBMS) store only tuples of one relation on a page. Examples of such DBMSes are Ingres and System R [20]. From now on we assume that pages contain only tuples of one relation. For a given value $a \in dom(\alpha_j)$, use of an index produces a list of tuple identifiers (TIDs). These TIDs allow direct retrieval of the stored tuples possessing the value a for attribute α_j . For example, if an index is organized as a B^+ -tree, leaf pages consist of (value, TID-list) pairs for every unique value of the indexed attribute.

The costs of processing a workload depends on many factors, such as storage costs, number of page accesses, processor time, etc. In this paper we consider the number of page accesses as the only cost factor.

Finally, we assume that the cardinality of the relations remains constant. To be more precise, the frequency of tuple insertions and tuple deletions is such that the total number of tuples of each relation remains constant in two consecutive choices of index sets.

Workload on a relation

We distinguish four possible operations in the workload on a database; queries, updates, insertions and deletions. Each of these operations include one or more steps.

- *Query*
 - (1) Select the relevant tuples from the data pages
 - (2) Output the relevant tuples to user
- *Update of tuples*
 - (1) Select the relevant tuples from the data pages
 - (2) Update the specified attributes and rewrite the data pages
 - (3) Update the relevant indices
- *Deletion of tuples*
 - (1) Select the relevant tuples from the data pages
 - (2) Remove the tuples and rewrite the data pages
 - (3) Update the relevant indices

- *Insertion of tuples*

- (1) Select the location(s) where the tuples will be stored
- (2) Insert the tuples and rewrite the data pages
- (3) Update the relevant indices

We concentrate on steps that influence index selection. The first step of an operation of the workload is always the selection of the relevant tuple(s). The execution of this step clearly depends on the available set of indexes, so it has to be taken into account. The second step is never influenced by the availability of indexes, so it can be ignored, while the third step, if present, depends only on the presence of indexes. This means that only the first and third step of an operation are taken into account from now on.

Let us elaborate on the first step of an operation w of the workload W . Consider the selection of all tuples satisfying a condition that is formulated in conjunctive normal form. We assume that if a usable index is available, it will be used. The following actions are distinguished:

- (1) Access all indices corresponding to attributes specified in the query which are usable search arguments; this gives a list of tuple identifiers (TIDs) per index.
- (2) Intersect the lists in order to determine the TIDs of the tuples that satisfy the conjunction of the conditions on the attributes of an index exists for.
- (3) Retrieve the tuples according to the result of the previous action.
- (4) Discard the tuples not satisfying the condition on the attributes without an index (false drops).

The total cost of selecting the relevant tuples is the sum of the costs resulting from each action. The cost of actions 2 and 4 can be neglected because they take place in main memory. So the costs of selecting tuples for an operation w with an index set A (denoted as $C_{sel}(A, w)$) is the sum of the cost of action 1 ($C_{ind}(A, w)$) and the cost of action 3 ($C_{ret}(T_w(A))$), in which $T_w(A)$ represents the number of tuples to be retrieved). For the used abbreviations, see Table 1.

Updating an index on attribute α_h requires roughly reading and writing of the index page with the old attribute value and doing the same for the new value. The insertion and deletion of an index requires a reading and writing of the relevant index page. The costs to keep an index up to date will be defined as the maintenance cost ($C_{main}(\{\alpha_h\})$) of an index. For a more accurate cost estimation for updates of indices we refer to [18].

Taking into account a workload W consisting of a set of operations (queries, updates, insertions, deletions) and the frequency of each operation $w \in W$ (f_w) the general cost function for the SIS problem is of the form:

Table 1
List of used symbols

α_h, α'_h	= attributes of a relation R
A, A'	= index sets
W	= workload
w	= an operation in the workload
p	= number of pages in file in which a relation is stored
n_R	= number of tuples in relation R
SF_α	= selectivity factor of α (reciprocal of the number of different values)
$T_w(A)$	= number of tuples to be retrieved in processing w with index set A
$C_{ind}(A, w)$	= costs to form the ordered TID-list for w using index set A
$C_{ret}(T_w(A))$	= costs for retrieving $T_w(A)$ tuples given an ordered TID-list
$C_{sel}(A, w)$	= $C_{ind}(A, w) + C_{ret}(T_w(A))$
$C_{main}(A)$	= maintenance cost of an index set A

$$CF(A) = \sum_{w \in W} f_w * C_{sel}(A, w) + \sum_{\alpha_h \in A} C_{main}(\{\alpha_h\}),$$

in which

A = an index set,

f_w = the frequency of the operation w contained in the workload W ,

$C_{sel}(A, w) = C_{ind}(A, w) + C_{ret}(T_w(A))$,

$C_{main}(\{\alpha_h\})$ = the maintenance costs of the index set $\{\alpha_h\}$.

Note, the maintenance cost of an index depends on the frequencies of the insertions, deletions and updates. Furthermore, each of these operations entails different cost as will be discussed in Section 5.

The SIS problem can now be formulated as ‘minimize the function CF ’. In other words, find an index set A in an efficient way such that the costs are minimal in processing a workload W . To solve this problem it is necessary to characterize the behaviour of the function C_{ret} . This is the subject of the next section.

3. Behavior of the retrieval function C_{ret}

In this section we will discuss the influence of index sets on retrieval cost. The number of page accesses needed to retrieve tuples indicated by an ordered TID-list is given by C_{ret} . Given the condition that the tuples must satisfy and given the index set A , the length of the TID-list can be estimated. To estimate the length it is assumed that attributes are mutually independent and that the attributes values are uniformly distributed, see further on. There are for a given TID-list still several ways to estimate the number of page accesses, see Section 5. A general remark about the function C_{ret} can, however, be made.

It can easily be shown that C_{ret} is a *monotonous non-increasing function* of the index set A . If the index set is empty, the file has to be scanned sequentially and this amounts to p page accesses, where p is the number of pages in the file in which the relation is stored. If an index can be used to retrieve the tuples, an ordered list of TIDs will be generated. This TID-list will be used to fetch the relevant tuples and we assume the list to be ordered on page number so that the retrieval requires at most p page accesses. Additional, usable indexes cause the generated list of TIDs to become shorter, so the number of page accesses does not increase.

Let us consider the effect of index sets on C_{ret} by means of an example.

Example 3.1. Consider the relation *Person*(*id.nr*, *name*, *salary*, *age*, *gender*, *education*) consisting of 400.000 tuples and the following query:

```
SELECT  name
FROM    Person
WHERE   education = 'secondary school' AND
        salary = 2000 AND
        age = 21
```

Suppose that the selectivity factors SF of the attributes *education*, *salary* and *age* are $SF_{education} = \frac{1}{10}$, $SF_{salary} = \frac{1}{40}$ respectively $SF_{age} = \frac{1}{50}$. Furthermore, a page contains 20 tuples.

Let us compare the addition of an index on *education* to three index sets $A = \emptyset$, $A' = \{salary\}$ and $A'' = \{salary, age\}$. Solving the query with the index set A requires

$400.000/20 = 20.000$ page accesses; when an index on *education* is added to the index set A , $400.000/10 = 40.000$ tuple retrievals are required, which implies again a cost of 20.000 page accesses.

Solving the query with A' requires 10.000 tuple retrievals; this means at most 10.000 page accesses as retrieving cost. The addition of *education* to the index set A' leads to at most 1000 page accesses as retrieval cost.

Using the index sets A'' respectively $A'' \cup \{education\}$ leads to at most 200 respectively 20 page accesses.

It is clear that adding an index on *education* to the empty index set does not help much, while the addition of the same index to the set A' causes a big gain. Moreover, the addition of the index on *education* to A'' gives, although less, still a substantial gain. These observations are the basis to characterize the function C_{ret} .

In the example above, it is shown that addition of an index to a bigger index set can be sometimes more profitable than the addition of the same index to a smaller index set. For instance, adding an index on *education* to A' is more profitable than adding the index to A . But sometimes the reverse also holds. The addition of the index on *education* to A' brings much more than the addition of the index to A'' . Taking these observations into account and considering the fact that C_{ret} is a monotonous non-increasing function, C_{ret} may be characterized according to Fig. 1.¹

The index sets in Fig. 1 are organized as $\emptyset \subset A_1 \subset A_2 \subset \dots \subset A_n$. Function 1 represents the retrieval cost as function of the index sets A_i , while Function 2 represents the retrieval cost with the index sets to which index α_h is added. The index sets in the area marked by I satisfy the observation that the addition of index α_h to a bigger index set is more profitable than the addition of the same index to a smaller set, i.e. $|\Delta C_{ret}(A_j, \alpha_h)|$ becomes bigger with the increase of the index sets. In the following we will refer to this observation as *Observation 1*. The index sets in the area marked by II of Fig. 1 satisfy the observation that the addition of an index to a smaller index set is more profitable than the addition of the

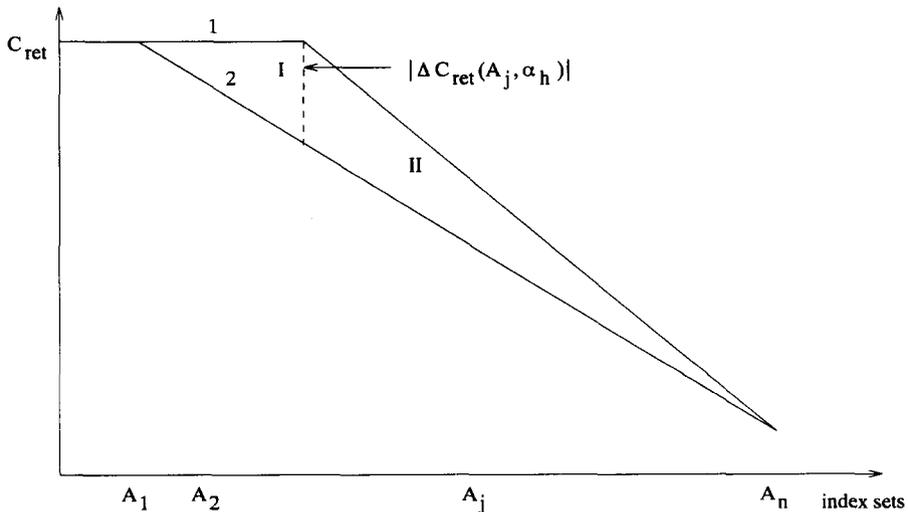


Fig. 1. Characterization of the function C_{ret} .

¹ For convenience's sake we have drawn a continuous function which has to be of course a discrete function.

index to a bigger set, i.e. $|\Delta C_{ret}(A_j, \alpha_h)|$ becomes smaller with the increase of the index sets. We shall refer to this as *Observation 2*.

Assuming that A and A' are in the same area, the observations can be written in formula form as:

$$\begin{aligned} & |C_{ret}(T_w(A' \cup \{\alpha'_h\})) - C_{ret}(T_w(A'))| \geq \\ & |C_{ret}(T_w(A \cup \{\alpha'_h\})) - C_{ret}(T_w(A))| \text{ for any } A \subseteq A', \alpha'_h \notin A' \end{aligned} \quad (1)$$

or

$$\begin{aligned} & |C_{ret}(T_w(A' \cup \{\alpha'_h\})) - C_{ret}(T_w(A'))| < \\ & |C_{ret}(T_w(A \cup \{\alpha'_h\})) - C_{ret}(T_w(A))| \text{ for any } A \subseteq A', \alpha'_h \notin A'. \end{aligned} \quad (2)$$

The following proposition shows that the behaviour of the general cost function CF is totally determined by the two observations.

Proposition 3.1. *The behaviour of the cost function*

$$CF(A) = \sum_{w \in W} f_w * C_{sel}(A, w) + \sum_{\alpha_h \in A} C_{main}(\{\alpha_h\}),$$

is determined by the function C_{ret} .

Proof. Because the maintenance cost of an index and the access cost to an index is constant this means that the cost function can be expressed as the sum of a constant function and a retrieval function C_{ret} . So, the form of the cost function is determined by C_{ret} and C_{ret} on his turn is determined by Observations 1 and 2. \square

In the next section sub- and supermodular functions are defined. The relationship between these functions and the observations described in this section are exposed. Finally, the way in which sub- and supermodular functions can contribute to the solution of the SIS problem, is described.

4. Super- and submodular functions

In the previous section two observations has been made. In this section it will be shown that the first observation leads to cost functions that are submodular, while the second observation causes supermodular cost functions. These functions will be characterized and it will be shown how totally submodular or totally supermodular cost functions can be optimized. In Section 5 we will show that often cost functions are not totally submodular or totally supermodular. For these cost functions we will combine the optimizing techniques which are valid for cost functions considering only one of the observations. This will be the subject of Section 6.

We start this section with a little variant on the general definition of supermodular functions with regard to sets, as presented in [25, 19]. Then we will manipulate the supermodular functions, such that it will be suitable to derive some properties which can be used in optimizing these kinds of functions. Finally we will generalize these results to submodular functions.

Definition 4.1. Let N be a finite set and G be a function defined as $G: \mathbb{P}(N) \rightarrow \mathbb{R}$, in which $\mathbb{P}(N)$ is the power set of N . We say that G is supermodular if:

$$G(X) + G(Y) \leq G(X \cup Y) + G(X \cap Y) \quad \forall X, Y \subseteq N. \quad (3)$$

Because we will often make use of the incremental value of adding an element e'_h to a set E we describe the class of super- and submodular functions in a more proper form with regard to our purpose.

Proposition 4.1. Let G be a supermodular function defined on a finite set $\mathbb{P}(N)$ then definition 4.1 is equivalent with the following statement:

$$G(E' \cup \{e'_h\}) - G(E') \geq G(E \cup \{e'_h\}) - G(E) \text{ for any } E \subseteq E' \subseteq N \text{ and } e'_h \notin E'. \quad (4)$$

A shorter notation for (4) is:

$$\Delta G(E' \cup e'_h) \geq \Delta G(E \cup e'_h) \text{ for any } E \subseteq E' \subseteq N \text{ and } e'_h \notin E'.$$

Proof (3) \Rightarrow (4). Take $E \subseteq E'$, $X = E \cup \{e'_h\}$ and $Y = E'$ in (3). This yields

$$\begin{aligned} G(E \cup \{e'_h\}) + G(E') &\leq G((E \cup \{e'_h\}) \cup E') + G((E \cup \{e'_h\}) \cap E') \Leftrightarrow \\ G(E \cup \{e'_h\}) + G(E') &\leq G(E' \cup \{e'_h\}) + G(E) \Leftrightarrow \\ G(E' \cup \{e'_h\}) - G(E') &\geq G(E \cup \{e'_h\}) - G(E). \end{aligned}$$

(4) \Rightarrow (3). Let $\{e_1, \dots, e_r\} = X \setminus Y$. Note that $X \setminus Y$ represents the set $\{e_i \mid e_i \in X \wedge e_i \notin Y\}$. From (4) we obtain

$$\begin{aligned} G(Y \cup \{e_1\}) - G(Y) &\geq G(X \cap Y \cup \{e_1\}) - G(X \cap Y) \\ G(Y \cup \{e_1, e_2\}) - G(Y \cup \{e_1\}) &\geq G(X \cap Y \cup \{e_1, e_2\}) - G(X \cap Y \cup \{e_1\}) \\ &\vdots \\ G(Y \cup \{e_1, e_2, \dots, e_r\}) - G(Y \cup \{e_1, e_2, \dots, e_{r-1}\}) &\geq \\ G(X \cap Y \cup \{e_1, e_2, \dots, e_r\}) - G(X \cap Y \cup \{e_1, e_2, \dots, e_{r-1}\}). \end{aligned}$$

Summing all these inequalities yields

$$\begin{aligned} G(Y \cup X) - G(Y) &\geq G(X) - G(X \cap Y) \Leftrightarrow \\ G(X) + G(Y) &\leq G(X \cup Y) + G(X \cap Y). \quad \square \end{aligned}$$

Suppose we want to minimize the supermodular function G defined on a finite set $\mathbb{P}(N)$, in other words to find the set $MIN \subseteq N$ such that $G(MIN) \leq G(E)$ for any $E \subseteq N$. The Propositions 4.2 and 4.3 defined below, provide us techniques to avoid exhaustive searching for a supermodular function. The intuitive ideas of these techniques are to divide the elements of N into three groups.

- (1) A group of elements which will never belong the the set MIN . These elements are achieved by applying Proposition 4.2, which is defined below.
- (2) A group of elements which will belong always to the set MIN . These elements are determined by Proposition 4.3, which is also defined below.

(3) A group of elements that require further exploration. This group contains all the elements not belonging to the previous groups.

The following proposition provides a technique to detect elements which will never belong to the set MIN . Suppose element e'_h is added to a set E . If the value of G does not decrease then it will also not decrease if e'_h is added to a set E' containing E . This means that e'_h will not be a member of MIN . The proof of the proposition follows directly from (4).

Proposition 4.2. *Let G be a supermodular function defined on the finite set $\mathbb{P}(N)$. Then the following holds:*

$$\text{If } G(E \cup \{e'_h\}) \geq G(E) \text{ then } G(E' \cup \{e'_h\}) \geq G(E') \text{ for any } E \subseteq E' \subseteq N \\ \text{and } e'_h \notin E' .$$

The following proposition identifies the elements that are certainly a member of the set MIN . Suppose that element e'_h is eliminated from a set E . If the value of G does not reduce by this elimination, it will not reduce if e'_h is eliminated from a set E' contained in E . So, e'_h must be added to the solution MIN .

Proposition 4.3. *Let G be a supermodular function defined on the finite set $\mathbb{P}(N)$. Then the following holds:*

$$\text{If } G(E \setminus \{e'_h\}) > G(E) \text{ then } G(E' \setminus \{e'_h\}) > G(E') \text{ for any } E' \subseteq E \text{ and } e'_h \in E' .$$

Proof. The dual form of Proposition 4.1 is:

$$G(E' \setminus \{e'_h\}) - G(E') \geq G(E \setminus \{e'_h\}) - G(E) \text{ for any } E' \subseteq E \text{ and } e'_h \in E' .$$

So, $\Delta G(E' \setminus e'_h) > \Delta G(E \setminus e'_h)$. \square

The relationship between these optimizing techniques and the SIS problem will be clarified. Suppose that only the second observation plays a role in the SIS problem. Then according to Proposition 3.1 and Proposition 4.1 the cost function CF (see Section 3) will behave like a supermodular function. So, Propositions 4.2 and 4.3 can be used. Note, the Propositions 4.1, 4.2 and 4.3 can easily be translated to the context of the SIS problem by replacing the supermodular function G by a supermodular cost function CF , N by the set of all attributes of a relation, the sets E , E' by the index sets A , A' and the element e'_h by the attribute α'_h . The SIS problem is solved if an index set MIN is found such that the cost of processing a workload with MIN will be minimal. One way to do this is by an ordered exhaustive search of all subsets represented by a full graph, in which each node represents a subset of NA , the set of all attributes of the relation, and each link between two nodes indicates that a node can be reached from the other by adding or discarding an element to or from the other node. The root of the graph is the empty set and the bottom of the graph is the full index set, the set consisting of all relevant attributes. An example of a full graph is given by Fig. 2.

Propositions 4.2 and 4.3 can help us to avoid an exhaustive search if the cost function behaves like a supermodular function. Applying these propositions means that we can construct a smaller graph without losing the optimal set of indices. Proposition 4.2 justifies the cutting of nodes and links starting from the root of the search tree while Proposition 4.3 can be applied to cut nodes and links starting from the bottom of the tree. Proposition 4.2 states that, if the addition of an index α'_h to an existing index set A causes an increase in costs

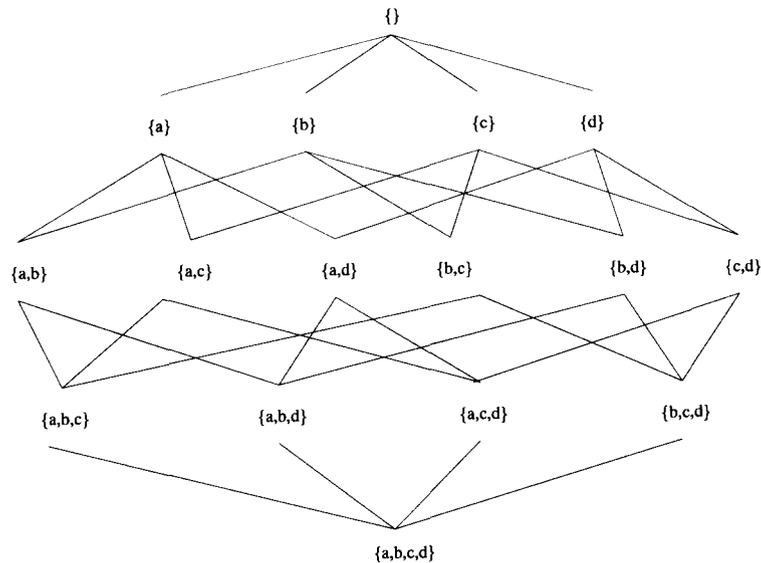


Fig. 2. Example of a full graph.

then the same will happen for all index sets A' containing A . So, α'_h is not a good index candidate. Proposition 4.3 asserts that, if the elimination of an index α'_h to an existing index set A causes an increase of the costs then the same will happen for each A' subset of A . So, α_h is a good index candidate. The following example illustrates the power of the propositions.

Example 4.1. Suppose we have a relation $R = \{a, b, c, d\}$ on which a workload is defined. Furthermore, the cost function which estimates the processing cost of the workload behaves like a supermodular function. Without applying the supermodular optimizing propositions all the nodes of Fig. 2 have to be evaluated to find the optimal set. Applying the propositions leads to the graph of Fig. 3. The numbers below the index sets are the processing cost for the workload, e.g. processing the workload with $\{\}$ costs 40 page accesses. We start with the empty set as root and the set $\{a, b, c, d\}$ as bottom. Applying Proposition 4.2, by taking $A = \{\}$ and $\alpha'_h = b$, to the root of the graph discards b as index candidate because the addition of b increases the cost. Applying Proposition 4.3, by taking $A = \{a, b, c, d\}$ and $\alpha'_h = c$, to the bottom of the graph adds c to the optimal set, because the drop of c causes an increase in cost. The consequence is that we have to consider from now on only subsets which do not contain b and must contain c .

So, only 12 sets have to be evaluated now to achieve the optimal set of secondary indices instead of 16.

Note that the cutting procedure starts always by evaluating $2n + 2$ sets. Then the Propositions 4.2 and 4.3 are applied. This process is repeated one level lower and is continued until it is no longer possible to add or to discard attributes as indices.

The reduction of the complexity of course depends on how often Propositions 4.2 and 4.3 can be applied successfully.

Now the relationship between the SIS problem in which only the second observation plays a role and the theory of supermodular function has been established we will introduce briefly submodular functions. The relationship between submodular functions and the SIS problem in which only the first observation plays a role is then easy to establish. Cost functions in which only the first observation plays a role will lead to a submodular cost function.

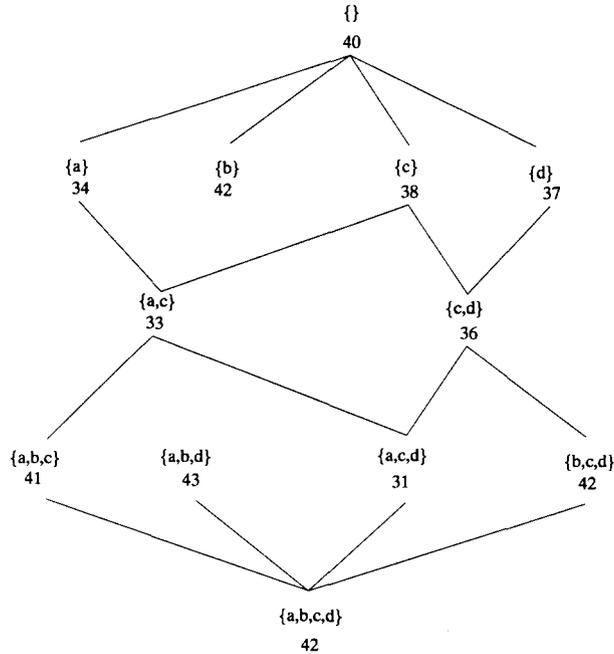


Fig. 3. Example of the reduced graph.

Definition 4.2. Let N be a finite set and G be a function defined as $G: \mathbb{P}(N) \rightarrow \mathbb{R}$, in which $\mathbb{P}(N)$ is the power set of N .

We say that G is submodular if:

$$G(X) + G(Y) \geq G(X \cup Y) + G(X \cap Y) \quad \forall X, Y \subseteq N. \tag{5}$$

Proposition 4.4. Let G be a submodular function defined on a finite set $\mathbb{P}(N)$ then Definition 4.2 is equivalent with the following statement:

$$G(E' \cup \{e'_h\}) - G(E') \leq G(E \cup \{e'_h\}) - G(E) \text{ for any } E \subseteq E' \subseteq N \text{ and } e'_h \notin E'. \tag{6}$$

A shorter notation for (6) is:

$$\Delta G(E' \cup e'_h) \leq \Delta G(E \cup e'_h) \text{ for any } E \subseteq E' \subseteq N \text{ and } e'_h \notin E'.$$

The proof of this proposition can be constructed in exactly the same way as the proof of Proposition 4.1.

For submodular functions we can derive similar propositions as for supermodular functions which may contribute in the reduction of the optimization of these functions. These propositions can be applied in the same way as done for supermodular functions. We will give now the propositions for submodular functions in terms of the SIS problem, i.e E, E' are considered as index sets and e'_h is considered as an index.

The following proposition holds for submodular functions and is analogous to Proposition 4.3. So, it can be used to select advantageous indices. It says that if an index e'_h is advantageous for an index set E , it will be advantageous also for each index set E' which contains E . So, e'_h must be added to the optimal set of indices.

Proposition 4.5. *Let G be a submodular function defined on the finite set $\mathbb{P}(N)$. Then the following holds:*

If $G(E \cup \{e'_h\}) < G(E)$ then $G(E' \cup \{e'_h\}) < G(E')$ for any $E \subseteq E' \subseteq N$ and $e'_h \notin E'$.

The proof of this proposition follows directly from Proposition 4.4.

The following proposition identifies the elements that can be discarded and is thus the equivalent of Proposition 4.2 for supermodular functions. It says that an index e'_h that reduces the costs when eliminated from a set E will also reduce the costs if e'_h is eliminated from any E' contained in E .

Proposition 4.6. *Let G be a submodular function defined on the finite set $\mathbb{P}(N)$. Then the following holds:*

If $G(E \setminus \{e'_h\}) \leq G(E)$ then $G(E' \setminus \{e'_h\}) \leq G(E')$ for $E' \subseteq E$ and $e'_h \in E'$.

The proof of this proposition can be constructed in the same way as the proof of Proposition 4.3.

Until now we have shown how to optimize cost functions which are either super- or submodular. How to decide whether a cost function is sub- or supermodular? The following proposition helps us.²

Proposition 4.7. *Let N be a finite set, G and M functions defined as $G: \mathbb{P}(N) \rightarrow \mathbb{R}$ and $M: N \rightarrow \mathbb{R}$. If $\Delta G(E \cup e'_h)$ can be written as $\Delta G(E \cup e'_h) = M(e'_h) - S(E \cup e'_h)$, in which $S(\cdot)$ is a function, $E \subseteq N$ and $e'_h \in N$ then G is a supermodular function if S is monotonous non-increasing and G is submodular if S is monotonous non-decreasing.*

Proof. Take $E \subseteq E' \subseteq N$ and $e'_h \notin E'$. We obtain:

$$\begin{aligned} \Delta G(E' \cup e'_h) - \Delta G(E \cup e'_h) &= (M(e'_h) - S(E' \cup e'_h)) - (M(e'_h) - S(E \cup e'_h)) \\ &= S(E \cup e'_h) - S(E' \cup e'_h). \end{aligned}$$

If S is a monotonous non-increasing function, then $S(E' \cup e'_h) \leq S(E \cup e'_h)$. Therefore, $\Delta G(E' \cup e'_h) \geq \Delta G(E \cup e'_h)$. If S is a monotonous non-decreasing function then the following holds: $S(E' \cup e'_h) \geq S(E \cup e'_h)$. Therefore, $\Delta G(E' \cup e'_h) \leq \Delta G(E \cup e'_h)$. \square

Note, the cost function CF of Section 2 can be written according to Proposition 4.7 as:

$$\Delta CF(A \cup \alpha'_h) = C_{main}(\{\alpha'_h\}) - \sum_{w \in W} f_w(C_{sel}(A, w) - C_{sel}((A \cup \{\alpha'_h\}), w)).$$

The first part of the function is the maintenance cost of the added index α'_h , while the second part indicates the gain achieved by adding the index.

To prove when and under which condition a cost function which has the form of CF is super- or submodular, we have to determine when and under which condition the second part of the function, thus $C_{sel}(A, w) - C_{sel}((A \cup \{\alpha'_h\}), w)$, is monotonous non-increasing or monotonous non-decreasing.

² This can be reached of course also by proving Proposition 4.1 or 4.4 but is not very practical.

We have presented some techniques to achieve complexity reduction for supermodular as well as for submodular functions. In general we expect that a cost function will be a combination of super- and submodular functions. How to apply the propositions then will be discussed in Section 6.

5. Specific cost functions

In this section we shall take a closer look at and analyse different cost functions. These functions are based on the general cost function of Section 2 and differ only in the retrieval function. Table 2 gives the specific symbols of this section which are used to construct the specific cost functions.

The cost to derive an ordered TID-list, $C_{ind}(A, w)$, will be expressed in C_{sind} , which represents the access cost to a single index. The blocking factor (BF) of an index is the average number of {value, TID-list} pairs per page and will among others be used in the formula for C_{sind} . In the particular case when indices are organized as a B^+ -tree $C_{sind}(\alpha_h) = \lceil \log_{BF_{\alpha_h}} n_R \rceil + \lceil SF_{\alpha_h} n_R / BF_{\alpha_h} \rceil - 1$. The first term is the height of the index tree and the second term is the number of leaf level blocks which are accessed.

Recall from Section 2 that the cost function CF consists of two parts, the maintenance cost of indices (C_{main}) and the selection cost of relevant tuples (C_{sel}). These costs will be the subject of the following subsections.

5.1 Maintenance cost of indices

The maintenance cost due to insertions or deletions of tuples requires an adjustment of all the indices. To perform this task we have to access the right index pages for all indices of the index set and to rewrite the index page. Let C_{ins} be the cost to place a pair {key, TID} after the last pair that has the same key in an index file and C_{del} the deletion cost of a pair {key, TID} or insertion cost of the same pair with any key.³ Then the maintenance cost due an insertion is expressed by:

$$\sum_{\alpha_h \in A} C_{ins}(\alpha_h).$$

Then maintenance cost due to deletion is expressed by:

$$\sum_{\alpha_h \in A} C_{del}(\alpha_h).$$

Table 2
Additional notations

NA	= full index set, i.e., contains all attributes of relation R
SNA_w	= set of attributes involved in the WHERE part of an operation w
SNA_u	= set of attributes which has to be updated
$C_{main}(A)$	= maintenance cost of an index set A
$C_{mind}(A)$	= $C_{main}(A) + \sum_{w \in W} f_w C_{ind}(A, w)$
$C_{sind}(\alpha_h)$	= costs to access to an index α_h
BF_{α_h}	= the blocking factor of an index on α_h

³ The formulae for C_{ins} and C_{del} can be found in [24] in the case indices are organized as B^+ -trees.

The maintenance cost due to tuple updates require deletions of the pairs {value, TID} with the old values and the insertions of the pairs with the new values. An estimation for the cost is:

$$\sum_{\alpha_h \in A \cap SNA_u} 2C_{del}(\alpha_h),$$

in which SNA_u is the set of attributes which has to be updated for update u .

Let I be the set of tuple insertions, D the set of tuple deletions and U the set of tuple updates. If we take the frequencies of each insertion f_i , deletion f_d and update f_u into account, then the maintenance cost of an index set A can be written as:

$$C_{main}(A) = \sum_{i \in I} f_i \sum_{\alpha_h \in A} C_{ins}(\alpha_h) + \sum_{d \in D} f_d \sum_{\alpha_h \in A} C_{del}(\alpha_h) + \sum_{u \in U} f_u \sum_{\alpha_h \in A \cap SNA_u} 2C_{del}(\alpha_h).$$

5.2 Selection costs of relevant tuples

Recall that the selection costs (C_{sel}) of relevant tuples in processing an operation w with an index set A consists of two parts. The first part is the cost required to form the TIDs lists of tuples which can be resolved with index set A (C_{ind}). The second part is the cost required to retrieve the tuples which remain after the intersection of the TIDs lists (C_{rel}).

The expected number of page accesses to form the lists of TIDS when an operation w requires a selection on relation R is:

$$C_{ind}(A, w) = \sum_{\alpha_h \in A \cap SNA_w} C_{sind}(\alpha_h),$$

in which SNA_w is the set of attributes involved in the WHERE part of w .

The number of tuples $T_w(A)$ that has to be retrieved for w after intersecting the TIDs lists is:

$$T_w(A) = n_R \prod_{\alpha_h \in A \cap SNA_w} SF_{\alpha_h}.$$

There are several functions available to retrieve the tuples given an ordered TID-list. Some of them will be discussed in the next subsection.

5.3 Retrieval functions

The simplest retrieval function one can think of, assumes that each TID results in one page access. Although this assumption is not very realistic, especially not when the blocking factor is large, it is used in several cost functions [13, 16]. The first retrieval function we consider is a variant of this simple function. Suppose T is the number of tuples that has to be retrieved and p the number of pages on which the tuples of a relation are stored. Then the expected number of page accesses (NPA) is the minimum of $\{p, T\}$, further denoted as $\min\{p, T\}$.

In the second function the retrieval of T tuples is modelled as drawing T tuples from p pages with replacement of the tuples, i.e. a tuple can be selected more than once at a time. This leads to the following formula $NPA_{Cardenas} = p(1 - (1 - 1/p)^T)$. The justification of the formula can be found in [7, 26].

The third retrieval function deals with the retrieving strategy based on drawing T tuples from p pages without replacement, i.e., a tuple cannot be selected more than once at a time. This strategy leads to the following formula $NPA_{Yao} = p(1 - \prod_{i=1}^T ((nd - i + 1)/(n - i + 1)))$,

in which n is the total number of tuples and $d = 1 - 1/p$. The proof of this formula can be found in [26, 12]. For practical reasons several researchers approximate this function [4, 14, 22]. A simple and good approximation of the function given by Bernstein et al. [4] is:

$$NPA_{Bernstein} = \begin{cases} T & \text{for } T < \frac{1}{2}p \\ \frac{1}{3}(T + p) & \text{for } \frac{1}{2}p < T < 2p \\ p & \text{for } 2p < T. \end{cases}$$

It is this approximation that will be used in the third retrieval function.

5.4 Three cost functions

We will present three different cost functions CF_i , $i = 1, 2, 3$, based on the earlier discussed retrieval functions. (Variants of) these cost functions are used by several researchers, see among others [2, 3, 6, 13, 16, 18, 24]. In the first cost function CF_1 we use the retrieval formula $C_{ret}(T_w(A)) = \min\{p, T_w(A)\}$ to retrieve $T_w(A)$ tuples. So, the following holds for processing a workload W :

$$CF_1(A) = C_{main}(A) + \sum_{w \in W} f_w(C_{ind}(A, w) + \min\{p, T_w(A)\}).$$

Because the maintenance cost of an index and the access cost to an index is constant we will denote them as C_{mind} . Thus,

$$CF_1(A) = C_{mind}(A) + \sum_{w \in W} f_w \min\{p, T_w(A)\},$$

in which $C_{mind}(A) = C_{main}(A) + \sum_{w \in W} f_w C_{ind}(A, w)$.

The following proposition describes the behaviour of CF_1 .

Proposition 5.1. *The cost function $CF_1(A)$ is supermodular for a workload W on the domain of index sets A for which holds: $\forall w \in W: T_w(A) \leq p$ and submodular on the domain of index sets A for which holds: $\forall w \in W: T_w(A) > p$.*

For the proof of the proposition we refer to appendix A. To illustrate the use of the proposition we give the following example.

Example 5.1. Consider a workload consisting of one operation w , a relation R with the set of attributes $\{a, b, c, d, e\}$ and the function CF_1 . The task is to select an optimal set of secondary indices. According to Proposition 5.1 each index set belongs to either the supermodular part or the submodular part. Consider two initial index sets the empty index set $\{\}$ and the full index set NA . Let us assume that processing the operation with NA leads to less than p retrievals, i.e. $T_w(NA) < p$, and processing the operation with $\{\}$ leads to $T_w(\{\}) \geq p$. So, $\{\}$ belongs to the submodular part of CF_1 and NA belongs to the supermodular part. Suppose that the processing cost for w with $\{\}$ is 40 and the processing cost with NA is 30. Assume further that we have computed the processing cost for w with the index sets given in Table 3.

Because the index set $\{\}$ belongs to the submodular part we can apply Proposition 4.5. This proposition says that an index is desired on the attributes a , c and e for all index sets belonging to the submodular part (because all these index sets includes the empty set index set). Applying Proposition 4.3 for the set NA , which belongs to the supermodular part

Table 3
Processing cost of q with several index sets

α'_h	$\{\} \cup \{\alpha'_h\}$	CF_1	α'_h	$NA \setminus \{\alpha'_h\}$	CF_1
a	{a}	39	a	{b, c, d, e}	32
b	{b}	41	b	{a, c, d, e}	29
c	{c}	38	c	{a, b, d, e}	38
d	{d}	42	d	{a, b, c, e}	35
e	{e}	38	e	{a, b, c, d}	28

results in the conclusion that an index on the attribute a , c and d is desired for all the index sets belonging to the supermodular part (because all these index sets are a subset of NA).

Then, it is easy to see that an index on the attributes a and c is desired for all index sets. So, a and c will always belong to the optimal set. The gain achieved by applying the Propositions 4.5 and 4.3 is that instead of evaluating 2^5 sets we have now to evaluate 20 sets to find the optimal set if the remaining index sets will be explored by an exhaustive search.

In Section 6 this idea will be generalized for workloads consisting more than one operation.

In the second cost function CF_2 we use the formula of Cardenas to retrieve $T_w(A)$ tuples. So, the following formula can be derived for CF_2 :

$$CF_2(A) = C_{mind}(A) + \sum_{w \in W} p(1 - (1 - 1/p)^{T_w(A)}).$$

The following proposition describes the behaviour of CF_2 .

Proposition 5.2. *The cost function CF_2 is supermodular, under the addition of an index α_h , for a workload W on the domain of index sets A for which holds $\forall w \in W: T_w(A) \leq 1/(SF_{\alpha'_i} - 1) \log_{p-1/p}(1/SF_{\alpha'_i})$ and submodular on the domain of index sets for which holds: $\forall w \in W: T_w(A) > 1/(SF_{\alpha'_i} - 1) \log_{p-1/p}(1/SF_{\alpha'_i})$.*

For a proof of the proposition we refer to the appendix. This result was earlier achieved by [2, 3]. In these papers it is asserted that it is reasonable to approximate CF_2 as a pure supermodular function. This means that Propositions 4.2 and 4.3 can be applied to optimize CF_2 .

In the third cost function CF_3 we use the retrieval function of Bernstein resulting in the following formula for CF_3 :

$$CF_3(A) = C_{mind}(A) + \sum_{w \in W} \begin{cases} T_w(A) & \text{for } T_w(A) \leq \frac{1}{2}p \\ \frac{1}{3}(T_w(A) + p) & \text{for } \frac{1}{2}p < T_w(A) \leq 2p \\ p & \text{for } T_w(A) > 2p. \end{cases}$$

The following proposition describes the behaviour of CF_3 . For the proof we refer to the appendix.

Proposition 5.3. *The cost function $CF_3(A)$ is supermodular for a workload W on the domain of index sets A for which holds: $\forall w \in W: T_w(A) \leq 2p$ and submodular on the domain of index sets A for which holds: $\forall w \in W: T_w(A) > 2p$.*

The fact that the maximal number of page accesses will be reached in a finite number of tuple retrievals for the cost functions CF_1 and CF_3 entails the practical significance of submodularity for these functions.

If we had assumed in CF_1 that each retrieval causes one page access than the CF_1 would be a supermodular function whatever the number of tuple retrievals would be [16]. For CF_3 it is also valid that the ceiling of the function $NPA_{Bernstein}$ makes submodularity of practical importance. The fact that in the formula of Cardenas, $NPA_{Cardenas} = p(1 - (1 - 1/p)^T)$ used in CF_2 , the maximal number of page accesses will be reached if the number of retrievals will be infinite, declares why submodularity hardly plays a role in this cost function.

5.5 Index sets

In this section we derive some properties for index sets in view of super- and submodular functions. Two index sets that play an important role in the optimizing techniques of sub- and supermodular functions are the empty index set $\{\}$ and the full index set NA . The following proposition summarizes some properties for those index sets and for index sets in general.

Proposition 5.4. *Let W be a workload and CF be a cost function of the following form*

$$CF(A) = \begin{cases} \text{supermodular on the index sets } A \text{ if } \forall w \in W: T_w(A) \leq kp, k = 1, 2 \\ \text{submodular on the index sets } A \text{ if } \forall w \in W: T_w(A) > kp. \end{cases}$$

Then the following holds:

- (1) *The empty set belongs to the submodular part of CF , i.e. $\forall w \in W: T_w(\{\}) > kp$.*
- (2) *If $\forall w \in W: T_w(NA) \geq kp$ then the optimal set of CF is $\{\}$.*
- (3) *Let A_{sup} be a fixed index set for which holds $\forall w \in W: T_w(A_{sup}) \leq kp$ then CF is supermodular for all index sets A satisfying $A_{sup} \subseteq A \subseteq NA$.*
- (4) *Let A_{sub} be a fixed index set for which holds $\forall w \in W: T_w(A_{sub}) \geq kp$ then CF is submodular for all index sets A satisfying $A \subseteq A_{sub} \subseteq NA$.*

Proof. Processing operations with the empty set implies a sequential scan for the operations, thus, $T_w(\{\}) = n_R$. Because the blocking factor (BF) is in general bigger than 2, the situation $n_R \leq kp = kn_R/BF$ can never occur. So, (1) holds.

If $\forall w \in W: T_w(NA) \geq kp$ then this will also hold for each index set because each index set is a subset of the full index set and T_w is monotonous non-increasing. So, for all index sets the retrieval cost for each query will be p page accesses. Each index set except the empty index set may entail maintenance cost. So, the empty index set will always be an optimal index set. So, (2) holds.

If $\forall w \in W: T_w(A_{sup}) < kp$ then this will also hold for all index sets $A \supseteq A_{sup}$ because T_w is monotonous non-increasing. Because each index set is a subset of the full index set (3) holds.

The proof for (4) follows also from the fact that T_w is monotonous non-increasing. \square

Corollary. *Let W_{red} be the set of operations w of a workload W such that $T_w(NA) < kp$. Then W_{red} is relevant for optimizing CF .*

Proof. The workload can be split in two groups, W_{red} and $W \setminus W_{red}$. For the group $W \setminus W_{red}$ the optimal set is the empty set according item 2 of Proposition 5.4. \square

Note, that the corollary contributes to the reduction of the complexity. It says that

operations which requires at least kp tuple retrievals using the full index set can be discarded from the optimizing process. Instead of evaluating the cost function with the whole workload we may evaluate the cost function now with a smaller workload.

Summarizing, optimizing a cost function of the form given in Proposition 5.4 for a workload W , which a corresponding W_{red} , starts with the conditions:

- (1) $\forall w \in W_{red}: T_w(\{\}) \geq kp$
- (2) $\forall w \in W_{red}: T_w(NA) < kp$.

6. A framework of an algorithm for the SIS problem

This section concerns the utilization of the theory developed in the previous sections in designing algorithms for the SIS problem. From this theory we know that a workload W can be replaced by a subset W_{red} without any adverse effect. Before considering a workload with an arbitrary W_{red} we shall first consider a W_{red} which consists of exactly one operation (Section 6.1). We will show how the achieved results of Section 5 may be applied to this specific W_{red} . Section 6.2 will generalize these techniques for a workload with an arbitrary W_{red} .

6.1 Index selection for a single operation

For the cost functions CF_1 and CF_3 it is clear from Proposition 5.1 and 5.3 respectively that processing an operation, each conceivable index set can be classified as belonging to either the supermodular or submodular part of the cost functions. From the nature of the retrieval functions used in CF_1 and CF_3 we know that the retrieval cost in processing the operation will be p page accesses if at least p respectively $2p$ tuple retrievals are required. So, processing the operation with index sets which require more than p tuples is not interesting if CF_1 respectively $2p$ if CF_2 is used as cost function because the retrieval cost will be maximal in these cases. This implies that for these cost functions the optimal set will be in the submodular part or it will be the empty set. Once the optimal set of the supermodular part is known it is easy to decide if this is the optimal set of the function or the empty set.

A technique to walk through the supermodular space will be given. We start by applying Proposition 4.3 to the full index set because this set belongs to the submodular part and contains all the subsets in this space. Suppose, this results in the conclusion that the set $\{\alpha_j, \dots, \alpha_n\}$ belongs to the optimal set, we call this the supported set A_{spt} . Then we evaluate $T_w(A_{spt})$. If $T_w(A_{spt}) \leq kp$, $k=1$ for CF_1 and $k=2$ for CF_3 , then we can apply repeatedly the optimizing properties of supermodular function as illustrated in Example 4.1 with as root the set $\{\alpha_j, \dots, \alpha_n\}$ and as bottom the full index set. This is justified by Property 3 of Proposition 5.4. If $T_w(A_{spt}) > kp$ then we do not have any exact techniques to reduce the complexity further. A heuristic is to enlarge the set A_{spt} with a minimal extra number of proper indices such that the number of tuple retrievals with the enlarged set becomes less than kp . In practice database designers can often rely on rules of thumb to produce such indices for a single operation [15]. Then the optimizing techniques for supermodular functions may be applied. The eventually solutions depend then on the extra added indices, the better they will be the better the solution.

6.2 Index selection for an arbitrary workload

The input of the algorithm consists of the cost function CF_i , the workload $W_{red} \subseteq W$, the attributes which have to be indexed according to the database designers and a good index set for each operation proposed by the database administrator.

The output of the algorithm will be a set of advantageous indices and a set of dis-

advantageous indices for groups of operations which are subsets of W_{red} . By an advantageous set we mean a set which will belong certainly to the optimal index set and by a disadvantageous set we mean a set which will never belong to the optimal index set. The union of the groups is W_{red} .

Before giving the body of the algorithm we shall take a closer look at the conditions of super- and submodularity for cost functions which have the form as in Proposition 5.4. This clarifies the techniques used in the body of the algorithm. As already noticed we can classify each index set either as sub- or supermodular with regard to a single operation. Considering the set W_{red} it is theoretically possible to generate the sub- and supermodular index sets for each $w \in W_{red}$. Table 4 gives a hypothetical example for $W_{red} = \{w_1, \dots, w_6\}$ and index sets A_1, \dots, A_5 . For the index sets in the table holds $A_1 \subset A_2 \subset \dots \subset A_5$.

Starting the optimizing process by applying Proposition 4.3 and taking the full index set as bottom this step is legal for some operations with regard to some index sets and for some not. For example, considering Table 3 and assuming that A_5 is the full index set then Proposition 4.3 would be applied legally for the situations under the solid line in the table. The application of Proposition 4.3 will generally lead to a statement that a set of attributes will belong to the optimal set, we call this set the supported set A_{spt1} . As already said this statement will be not true for all operations. The operations which do not support A_{spt1} can be detected easily by evaluating $T_w(A_{spt1})$.

The evaluations of $T_w(A_{spt1})$ for all $w \in W_{red}$ will generally lead to two groups, one group G_1 containing the operations for which hold $T_w(A_{spt1}) \leq kp$ and the second group G_2 containing the operations for which hold $T_w(A_{spt1}) > kp$. For group G_1 the proposition is applied legally and for G_2 not.

The optimizing process for G_1 can be continued by applying the supermodular properties, taking the full index set as bottom and A_{spt1} as root, finally leading to an advantageous set of indices $A_{ad1} \supseteq A_{spt1}$ and a disadvantageous set A_{dis1} .

There are several ways to deal with the group G_2 . We will discuss one possibility which will be used in the algorithm. We divide G_2 in a minimal number of subgroups G_{2i} such that it will be possible to find an index set A_{spt2i} for each subgroup G_{2i} for which holds $\forall w \in G_{2i}: T_w(A_{spt2i}) \leq kp$. Then it is clear that we can apply the optimizing properties of a supermodular function for each subgroup G_{2i} by taking A_{spt2i} as root and the full index set as bottom. Finally for each subgroup G_{2i} we will find an advantageous set of indices $A_{ad2i} \supseteq A_{spt2i}$ and a disadvantageous set A_{dis2i} .

The dividing process in subgroups will be done on basis of the proposed index sets for the operations by the database administrator. Suppose we have three operations w_1, w_2 and w_3 which belong to group G_2 for which the database administrator propose the index sets $\{a, b, d\}, \{a, c, d\}$ respectively $\{a, b, c, d, e\}$. Then we may choose $A_{spt2i} = \{a, d\}$ because a and c are important candidates for all the sets. Suppose the evaluation of the number of tuple retrievals with A_{spt2i} results in $T_{w_1}(A_{spt2i}) < kp, T_{w_2}(A_{spt2i}) < kp$ and $T_{w_3}(A_{spt2i}) > kp$

Table 4
Sub- and supermodular index sets for several operations

	w_1	w_2	w_3	w_4	w_5	w_6
A_1	sub	sub	sub	sub	sub	sub
A_2	sub	sub	sub	sub	sup	sub
A_3	sub	sup	sub	sup	sup	sub
A_4	sup	sup	sub	sup	sup	sub
A_5	sup	sup	sup	sup	sup	sup

then w_1 and w_2 will form a subgroup and w_3 another group. Note, in the worst case we will have subgroups consisting of only one operation.

We will give now an informal description of the algorithm. The first step in the algorithm is to check the cost function. If the cost function is CF_2 then the optimizing properties of supermodular functions can be applied. In the other cases we take the full index set minus the attributes which have to be indexed (NA_{red}) according to the user and apply Proposition 4.3. If this step leads to the support of a set of indices A_{spt} then we evaluate $\forall w \in W_{red}: T_w(A_{spt})$. On the basis of the outcome we discern two groups G_1 containing all w 's for which holds $T_w(A_{spt}) < kp$ and G_2 contains the remaining w 's.

For group G_1 we can apply repeatedly the supermodular optimizing propositions taking A_{spt} as root and NA_{red} as bottom finally leading to an advantageous set of indices A_{ad1} and the disadvantageous set A_{dis1} .

Group G_2 will be divided in several subgroups G_{2i} such that the supermodular properties can be applied leading for each subgroup G_{2i} an advantageous set of indices A_{ad2i} and a disadvantageous set of indices A_{dis2i} .

Note, the algorithm does not propose an index set for the workload but it generates for groups of operations the advantageous and disadvantageous indices. This information may be useful in the selection of indices in some approaches. We discuss briefly one alternative to use this information.

We combine all the advantageous and disadvantageous sets of all the subgroups leading to an advantageous set A_{ad} and a disadvantageous A_{dis} for the whole workload. Then the original SIS problem is replaced by a more 'simple' one since we do not have to consider the attributes in A_{ad} and A_{dis} . Then, the simple SIS problem may be solved by an exhaustive search.

To get the advantageous set A_{ad2} and the disadvantageous set A_{dis2} for group G_2 we have to combine all the sets A_{ad2i} respectively A_{dis2i} . Then we have to combine A_{ad2} with the advantageous index set of G_1 , A_{ad1} , resulting in A_{ad} . A similar procedure holds for achieving A_{dis} . For the combinations of the sets one can use simple or more advanced techniques [15]. A simple technique may be the rule 'if an index is supported by $x\%$ of the operations and it is not a disadvantageous index for the other operations then it belongs to A_{ad} '. To obtain the set A_{dis} a similar rule can be used; 'if an index is disadvantageous for $y\%$ of the operations and it is not an advantageous index for the other operations then it belongs to A_{dis} '. If $(x, y) = (100, 100)$ then A_{ad} contains the indices which is advantageous for each group and A_{dis} the indices which is disadvantageous for all groups. The value for x and y may be derived from experiments.

7. Foundation for ADD and DROP algorithms

Whang's DROP algorithm [24] starts by considering the set made up of all possible indices and, at each step, drops the index which would cause the highest decrease in cost. When the function value cannot be further lowered by leaving out any of the indices left, the algorithm attempts to cut out two indices at a time, then three indices, and so on, until no further reductions are possible. The cost function on which this algorithm is applied is quite similar to CF_3 . The technique to drop one index at a time is just an application of Proposition 4.6 and it is a property of *submodular* functions. As we can see from Proposition 5.3, CF_3 does not behave completely as a submodular function. Later on in this section we shall discuss the consequence of applying the submodular property on this cost function.

Whang compares his algorithm which the ADD type algorithm with regard to the same cost function CF_3 . An ADD type algorithm starts from an empty set and, at each step, it adds the index most capable of reducing the cost and it stops when there are no more

cost-reducing indices. This rule is just an application of Proposition 4.5 which is also a property of *submodular* functions. So, the ADD and the DROP algorithms are not competitive but are supplementary in the case of a submodular function.

We discuss the consequence of applying the submodular properties to CF_3 . Applying the ADD algorithm to this cost function starting with an empty index set is a legal step (Proposition 5.4). Suppose this step results in an advantageous index set. In the ADD algorithm one of the elements of this set will be chosen, let us call this set A_{ad} . Then it is not necessary that A_{ad} will belong to the submodular part for all operations. In general, for some operations A_{ad} will be in the submodular part, while for others in the supermodular part. Applying again the ADD algorithm means that this is applied illegally to the groups of operations for which hold that A_{ad} was in the supermodular part. At each addition of an index to A_{ad} we may expect in general that the number of operations for which the ADD algorithm is applied illegally will grow.

Another point is that once an attribute is added as an index to A_{ad} , it will always belong to the set. So, if on a moment a disadvantageous index is added to A_{ad} , which may be the case when most of the operations behave like supermodular, then this will be always in the index set.

Applying the DROP algorithm on CF_3 will be in general a bad start from a theoretical point of view because the set belongs to the supermodular part. But the DROP algorithm will never eliminate an advantageous index (compare Proposition 4.6 with Proposition 4.3 which we apply). The DROP algorithm eliminates the indices which would be a candidate for further exploration. This is probably the reason why the optimal index set may be missed. The success of the DROP algorithm is that it provides all advantageous indices.

It is noteworthy that when the ADD and DROP algorithms are allowed to apply this can be performed in one step instead of the several steps described above. This is justified by Propositions 4.5 and 4.6.

8. Concluding remarks and further research

We have described the problem of secondary index selection, the so-called SIS problem. Adding an index to a set of selected indexes can cause, among others depending on the cost function, two different results. The first result is that ‘the addition of an index to a bigger index set is more profitable than the addition of the same index to a smaller index set’. This is called Observation 1. The opposite observation (Observation 2) implies that ‘the larger an index set the less the gain of an index addition will be’. It has been shown that in reality both observations can occur.

Situations in which the first observation holds can be captured by submodular functions; the second observation is covered by supermodular functions. Mathematical properties of sub- and supermodular functions have been derived. It appears that SIS problems that can be described completely by submodular (or supermodular) functions can be solved easily, that means with a low complexity.

We have analysed three representative cost functions and we exposed that two of the cost functions cannot be classified as a pure super- or submodular function. Depending on the workload, a sub- or a supermodular function is obtained. To deal with this situation we developed an algorithm that forms groups of operations of the workload. It then determines disadvantageous and advantageous indices for each group. In a rough sense this can be considered as a partly generalization of [15]. On basis of a conceptual schema and each query separately an ideal storage structure (so among others secondary indices) for the relations is determined. Then all the ideal storage structures are combined to determine the eventually storage structure. In this paper, instead of considering each query separately we presented a technique to consider *groups of queries*.

The analysis of the cost functions in terms of super- and submodular functions gives also a better understanding of the ADD and DROP algorithms. They have been described as being successful [24] but the mathematical foundation behind the success has not been given. It appears that these algorithms are based on the mathematical properties of submodular functions.

It should be noticed that in [2, 3, 16] it was already pointed out how to handle supermodular functions. But the lack of a complete mathematical foundation of super- and submodular functions has led to confusions. For example, Barcucci et al. [3] have proved that a variant of cost function CF_2 could be considered as supermodular and then they applied the optimizing properties of these functions. After this process was finished they applied the heuristic DROP and ADD algorithms to reduce the complexity further. It is clear that this step cannot be justified because the ADD and DROP algorithms are based on properties of submodular functions.

A topic for future research is the elaboration and implementation of the algorithm described in this paper in a tool for physical database design [9]. The application of the algorithm to real problems and the evaluation of the results have to be done in the future. A last research topic concerns the extension of the theory to operations that concern more than one relation, that is to joins.

Appendix A. Derivation of the behaviour of specific cost functions

In this appendix the proofs of the Propositions 5.1, 5.2 and 5.3 are given.

Proposition 5.1. *The cost function $CF_1(A)$ is supermodular for a workload W on the domain of index sets A for which holds: $\forall w \in W: T_w(A) \leq p$ and submodular on the domain of index sets A for which holds: $\forall w \in W: T_w(A) > p$.*

Proof. Suppose that $\alpha'_h \notin A$. Then the following holds:

$$\begin{aligned} \Delta CF_1(A \cup \alpha'_h) &= CF_1(A \cup \{\alpha'_h\}) - CF_1(A) = C_{mind}(A \cup \{\alpha'_h\}) \\ &\quad + \sum_{w \in W} f_w \min\{p, T_w(A \cup \{\alpha'_h\})\} \\ &\quad - \left(C_{mind}(A) + \sum_{w \in W} f_w \min\{p, T_w(A)\} \right) \\ &= C_{mind}(\{\alpha'_h\}) + \sum_{w \in W} f_w (\min\{p, T_w(A \cup \{\alpha'_h\})\} - \min\{p, T_w(A)\}) \\ &= C_{mind}(\{\alpha'_h\}) - \left(\sum_{w \in W} f_w (\min\{p, T_w(A)\} - \min\{p, SF_{\alpha'_h} T_w(A)\}) \right). \end{aligned}$$

It is obvious from Fig. 4 that

$$\begin{aligned} &\min\{p, T_w(A)\} - \min\{p, SF_{\alpha'_h} T_w(A)\} \\ &= \begin{cases} T_w(A)(1 - SF_{\alpha'_h}) & \text{for } T_w(A) < p \\ p - SF_{\alpha'_h} T_w(A) & \text{for } p \leq T_w(A) \leq \frac{p}{SF_{\alpha'_h}} \\ 0 & \text{for } T_w(A) > \frac{p}{SF_{\alpha'_h}}. \end{cases} \end{aligned}$$

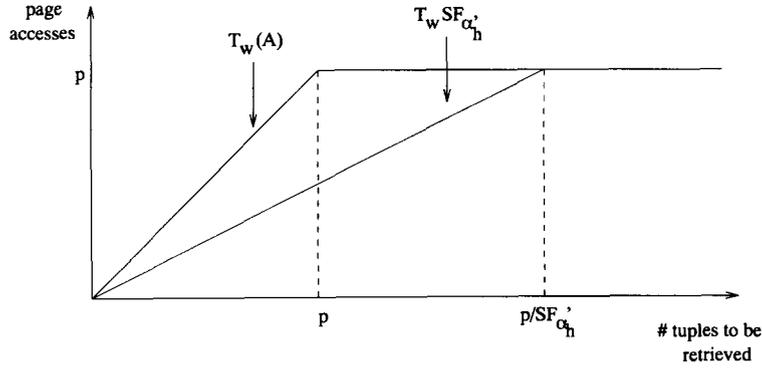


Fig. 4. Expected number of page accesses using the retrieval function $\min\{p, T_w(A)\}$.

Because T_w is a monotonous non-increasing function, $-T_w$ is monotonous non-decreasing. So, we conclude that $\min\{p, T_w(A)\} - \min\{p, SF_{\alpha'_h} T_w(A)\}$ is monotonous non-increasing for $T_w(A) < p$ and monotonous non-decreasing for $T_w(A) \geq p$. Because the sum of monotonous non-decreasing functions is a monotonous non-decreasing function and the sum of non-increasing functions is a non-increasing function the following holds:

- (1) If $\forall w \in W: T_w(A) < p$ then $\sum_{w \in W} (\min\{p, T_w(A)\} - \min\{p, SF_{\alpha'_h} T_w(A)\})$ will be monotonous non-increasing.
- (2) If $\forall w \in W: T_w(A) \geq p$ then $\sum_{w \in W} (\min\{p, T_w(A)\} - \min\{p, SF_{\alpha'_h} T_w(A)\})$ will be monotonous non-decreasing.

Therefore, according to Proposition 4.7 $CF_1(A)$ is a supermodular function if $\forall w \in W: T_w(A) < p$ and submodular if $\forall w \in W: T_w(A) \geq p$. \square

Proposition 5.2. *The cost function CF_2 is supermodular, under the addition of an index α_h , for a workload W on the domain of index sets A for which holds $\forall w \in W: T_w(A) \leq 1/(SF_{\alpha'_h} - 1) \log_{p-1/p}(1/SF_{\alpha'_h})$ and submodular on the domain of index sets for which holds: $\forall w \in W: T_w(A) > 1/(SF_{\alpha'_h} - 1) \log_{p-1/p}(1/SF_{\alpha'_h})$.*

Proof. Suppose that $\alpha'_h \notin A$. Then the following holds:

$$\begin{aligned}
 \Delta CF_2(A \cup \alpha'_h) &= CF_2(A \cup \{\alpha'_h\}) - CF_2(A) \\
 &= C_{mind}(A \cup \{\alpha'_h\}) + \sum_{w \in W} f_w p (1 - (1 - 1/p)^{T_w(A \cup \{\alpha'_h\})}) \\
 &\quad - \left(C_{mind}(A) + \sum_{w \in W} f_w p (1 - (1 - 1/p)^{T_w(A)}) \right) \\
 &= C_{mind}(\{\alpha'_h\}) + \sum_{w \in W} f_w (p (1 - (1 - 1/p)^{T_w(A \cup \{\alpha'_h\})}) \\
 &\quad - p (1 - (1 - 1/p)^{T_w(A)})) \\
 &= C_{mind}(\{\alpha'_h\}) - \left(\sum_{w \in W} f_w (p (1 - (1 - 1/p)^{T_w(A)}) \right. \\
 &\quad \left. - p (1 - (1 - 1/p)^{T_w(A) SF_{\alpha'_h}})) \right).
 \end{aligned}$$

The derivative of $p(1 - (1 - 1/p)^{T_w(A)}) - p(1 - (1 - 1/p)^{T_w(A)SF_{\alpha'_h}})$ with respect to $T_w(A)$, $d'(T_w(A))$ is:

$$d'(T_w(A)) = pSF_{\alpha'_h}T'_w(A)(1 - 1/p)^{SF_{\alpha'_h}T_w(A)} \log\left(1 - \frac{1}{p}\right) - pT'_w(A)(1 - 1/p)^{T_w(A)} \log\left(1 - \frac{1}{p}\right).$$

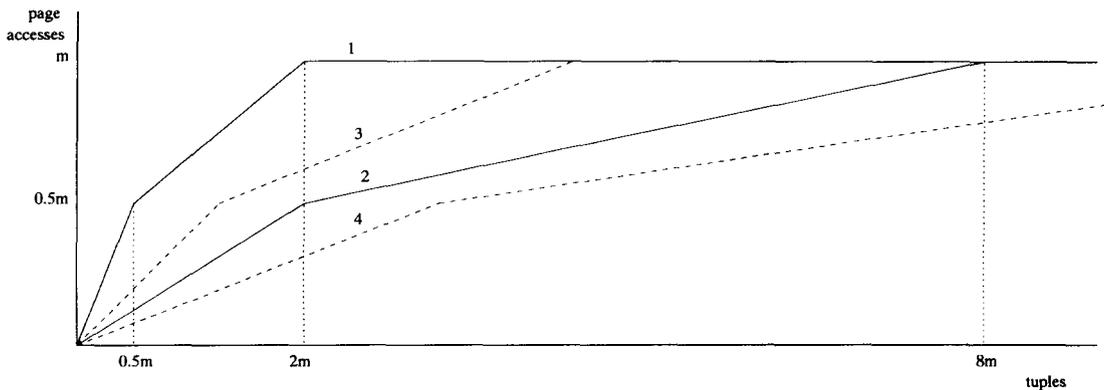
Considering $d'(T_w(A))$ we see that $d'(T_w(A)) \leq 0$ for $T_w(A) \leq (1/SF_{\alpha'_h} - 1) \log_{p-1/p}(1/SF_{\alpha'_h})$ and $d'(T_w(A)) > 0$ otherwise. Therefore, according to Proposition 4.7 $CF_2(A)$ is supermodular if $\forall w \in W: T_w(A) \leq (1/SF_{\alpha'_h} - 1) \log_{p-1/p}(1/SF_{\alpha'_h})$ and submodular if $\forall w \in W: T_w(A) > (1/SF_{\alpha'_h} - 1) \log_{p-1/p}(1/SF_{\alpha'_h})$. \square

Proposition 5.3. *The cost function $CF_3(A)$ is supermodular for a workload W on the domain of index sets A for which holds: $\forall w \in W: T_w(A) \leq 2p$ and submodular on the domain of index sets A for which holds: $\forall w \in W: T_w(A) > 2p$.*

Proof. Suppose that $\alpha'_h \notin A$. Then the following holds for $CF_3(A \cup \{\alpha'_h\})$:

$$CF_3(A \cup \{\alpha'_h\}) = C_{\min}(A \cup \{\alpha'_h\}) + \sum_{w \in W} \begin{cases} T_w(A)SF_{\alpha'_h} & \text{for } T_w(A) \leq \frac{p}{2SF_{\alpha'_h}} \\ \frac{1}{3}(T_w(A)SF_{\alpha'_h} + p) & \text{for } \frac{p}{2SF_{\alpha'_h}} < T_w(A) \leq \frac{2p}{SF_{\alpha'_h}} \\ p & \text{for } T_w(A) > \frac{2p}{SF_{\alpha'_h}}. \end{cases}$$

It is obvious from Fig. 5 that $\Delta CF_3(A \cup \{\alpha'_h\})$ depends on the selectivity factor $SF_{\alpha'_h}$. For $SF_{\alpha'_h} \geq \frac{1}{4}$ the following holds:



Function 1 represents the number of tuples ($T_w(A)$) that have to be retrieved considering an index set A while the other functions represent the number of tuples that have to be retrieved considering an index set $(A \cup \{\alpha'_h\})$ with different selectivity factors $SF_{\alpha'_h}$ for α'_h . In function 2, $SF_{\alpha'_h} = \frac{1}{4}$, in function 3, $SF_{\alpha'_h} > \frac{1}{4}$ and in function 4, $SF_{\alpha'_h} < \frac{1}{4}$.

Fig. 5. Expected number of page accesses using the retrieval function $NPA_{Bernstein}$.

$$\Delta CF_3(A \cup \alpha'_h) = C_{\min}(\{\alpha'_h\})$$

$$- \sum_{w \in W} \begin{cases} T_w(A)(1 - SF_{\alpha'_h}) & \text{for } T_w(A) < \frac{1}{2} p \\ T_w(A)\left(\frac{1}{3} - SF_{\alpha'_h}\right) + \frac{1}{3} p & \text{for } \frac{p}{2} \leq T_w(A) < \frac{p}{2SF_{\alpha'_h}} \\ \frac{1}{3} T_w(A)(1 - SF_{\alpha'_h}) & \text{for } \frac{p}{2SF_{\alpha'_h}} \leq T_w(A) < 2p \\ \frac{2}{3} - \frac{1}{3} T_w(A)SF_{\alpha'_h} & \text{for } 2p \leq T_w(A) < \frac{2p}{SF_{\alpha'_h}} \\ 0 & \text{for } T_w(A) \geq \frac{2p}{SF_{\alpha'_h}} \end{cases}$$

For $SF_{\alpha'_h} < \frac{1}{4}$, $\Delta CF_3(A \cup \alpha'_h)$ becomes:

$$\Delta CF_3(A \cup \alpha'_h) = C_{\min}(\{\alpha'_h\})$$

$$- \sum_{w \in W} \begin{cases} T_w(A)(1 - SF_{\alpha'_h}) & \text{for } T_w(A) < \frac{1}{2} p \\ T_w(A)\left(\frac{1}{3} - SF_{\alpha'_h}\right) + \frac{1}{3} p & \text{for } \frac{p}{2} \leq T_w(A) < 2p \\ -T_w(A)SF_{\alpha'_h} + p & \text{for } 2p \leq T_w(A) < \frac{p}{2SF_{\alpha'_h}} \\ -\frac{1}{3} T_w(A)SF_{\alpha'_h} + \frac{2}{3} p & \text{for } \frac{p}{2SF_{\alpha'_h}} < T_w(A) \leq \frac{2p}{SF_{\alpha'_h}} \\ 0 & \text{for } T_w(A) \geq \frac{2p}{SF_{\alpha'_h}} \end{cases}$$

In the case of $SF_{\alpha'_h} \geq \frac{1}{4}$ we can distinct two situations. It can easily be verified for $SF_{\alpha'_h} > \frac{1}{3}$ that $T_w(A)\left(\frac{1}{3} - SF_{\alpha'_h}\right) + \frac{1}{3}p$ is non-decreasing on $p/2 \leq T_w(A) < p/2SF_{\alpha'_h}$, while it is non-increasing on this area for $\frac{1}{4} \leq T_w(A) \leq \frac{1}{3}$.

Because T_w is a monotonous non-increasing and $-T_w$ monotonous non-decreasing the following holds:

$$\frac{1}{4} \leq T_w(A) \leq \frac{1}{3}: T_w(A)(1 - SF_{\alpha'_h}) \text{ for } T_w(A) < \frac{1}{2} p \text{ is non-increasing,}$$

$$T_w(A)\left(\frac{1}{3} - SF_{\alpha'_h}\right) + \frac{1}{3} p \text{ for } \frac{p}{2} \leq T_w(A) < \frac{p}{2SF_{\alpha'_h}}$$

is non-increasing,

$$\frac{1}{3} T_w(A)(1 - SF_{\alpha'_h}) \text{ for } \frac{p}{2SF_{\alpha'_h}} \leq T_w(A) < 2p \text{ is non-increasing,}$$

$$\frac{2}{3} - \frac{1}{3} T_w(A)SF_{\alpha'_h} \text{ for } 2p \leq T_w(A) < \frac{2p}{SF_{\alpha'_h}} \text{ is non-decreasing,}$$

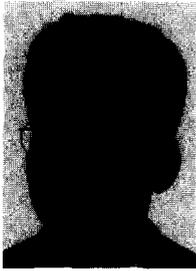
$$\begin{aligned}
SF_{\alpha_h} < \frac{1}{4}: T_w(A)(1 - SF_{\alpha_h}) \text{ for } T_w(A) < \frac{1}{2} p \text{ is non-increasing,} \\
T_w(A)\left(\frac{1}{3} - SF_{\alpha_h}\right) + \frac{1}{3} p \text{ for } \frac{p}{2} \leq T_w(A) < \frac{p}{2SF_{\alpha_h}} \text{ is non-increasing,} \\
-T_w(A)SF_{\alpha_h} + p \text{ for } 2p \leq T_w(A) < \frac{p}{2SF_{\alpha_h}} \text{ is non-decreasing,} \\
-\frac{1}{3} T_w(A)SF_{\alpha_h} + \frac{2}{3} p \text{ for } \frac{p}{2SF_{\alpha_h}} < T_w(A) \leq \frac{2p}{SF_{\alpha_h}} \text{ is non-decreasing.}
\end{aligned}$$

Keeping in mind that the sum of non-increasing respectively non-decreasing functions is a non-increasing respectively non-decreasing function we conclude that when all $SF_{\alpha_h} \leq \frac{1}{3}$, $CF_3(A)$ is supermodular if $\forall w \in W: T_w(A) < 2p$ and submodular if $\forall w \in W: T_w(A) \geq 2p$. \square

References

- [1] H.D. Anderson and P.B. Berra, Minimum cost selection of secondary indexes for formatted files, *ACM Trans. Database Syst.* 2 (1977) 68–90.
- [2] E. Barucci, A. Chiuderi, R. Pinzani and M.C. Verri, Index selection in relational databases, *Proc. Mathematical Fundamentals of Database Syst.* 89 (1989) 24–36.
- [3] E. Barucci, R. Pinzani and R. Sprugnoli, Optimal selection of secondary indexes, *IEEE Trans. Software Engrg.* 16 (1) (1990) 32–38.
- [4] P.A. Bernstein, N. Goodman, E. Wong, C.L. Reeve and J.B. Rothnie, Query processing in a system for distributed databases (SDD-1), *ACM Trans. Database Syst.* 6 (4) (1981) 602–625.
- [5] R. Bonanno, D. Maio and P. Tiberio, An approximation algorithm for secondary index selection in relational database physical design, *Comput. J.* 28 (4) (1985) 398–405.
- [6] 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. (North-Holland, Amsterdam, 1983) 149–160.
- [7] A.F. Cardenas, Analysis and performance of inverted database structures, *Commun. ACM* 18 (5) (1975) 253–263.
- [8] R. Choenni, H.M. Blanken and S.C. Chang, Index selection in relational databases, to appear in *Proc. Internat. Conf. on Computing and Information* (1993).
- [9] R. Choenni, H.M. Blanken and S.C. Chang, On the automation of physical database design, *Proc. ACM Symp. Applied Computing* (1993) 358–368.
- [10] D. Comer, The difficulty of optimum index selection, *ACM Trans. Database Syst.* 3 (4) (1978) 440–445.
- [11] S. Finkelstein, M. Schkolnick and P. Tiberio, Physical database design for relational databases, *ACM Trans. Database Syst.* 13 (1) (1988) 91–128.
- [12] A.H. Haitzma, A combinatorial problem related to the selection of indices for a file, Memorandum Nr. 98, Department of Applied Mathematics, Twente University of Technology, Enschede, 6p., 1975.
- [13] M.Y.L. Ip, L.V. Saxton and V.V. Raghavan, On the selection of an optimal set of indexes, *IEEE Trans. Software Engrg.* SE-9 (2) (1983) 135–143.
- [14] W.S. Luk, On estimating block accesses in database organizations, *Commun. ACM* 26 (11) (1983) 945–948.
- [15] S. Rozen and D. Shasha, A framework for automating database design, *Proc. Internat. Conf. on Very Large Databases* (1991) 401–411.
- [16] M. Schkolnick, The optimal selection of secondary indices for files, *Informat. Syst.* 1 (1975) 141–146.
- [17] M. Schkolnick, A survey of physical database design methodology and techniques, *Proc. Internat. Conf. on Very Large Databases* (1978) 474–487.
- [18] M. Schkolnick and P. Tiberio, Estimating the cost of updates in a relational database, *ACM Trans. Database Syst.* 10 (2) (1985) 163–179.
- [19] D.M. Topkis, Minimizing a submodular function on a lattice, *Operat. Res.* 26 (2) (1978) 304–321.
- [20] J.D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. 1* (Computer Science Press, Rockville, MD, 1988).
- [21] K. Whang, G. Wiederhold and D. Sagalowicz, Separability – An approach to physical database design, *Proc. Internat. Conf. on Very Large Data Bases* (1981) 487–500.

- [22] K. Whang, G. Wiederhold and D. Sagalowicz, Estimating block accesses in database organizations: a closed noniterative formula, *Commun. ACM* 26 (11) (1983) 940–944.
- [23] K. Whang, Property of separability in physical design of networkmodel databases, *Informat. Syst.* 10 (1) (1984) 57–63.
- [24] K. Whang, Index selection in relational databases in: S. Ghosh, Y. Kambayashi and K. Tanaka (eds), *Proc. Foundations of Data Organization*, (1987) 487–500.
- [25] L.A. Wolsey, Maximising real-valued submodular functions: Primal and dual heuristics for location problems, *Math Operat. Res.* 7 (3) (1982) 410–425.
- [26] S.B. Yao, Approximating block accesses in database organizations, *Commun. ACM* 20 (4) (1977) 260–261.



Sunil Choenni holds a master degree in Theoretical Computer Science from the Delft University of Technology. Currently, he is a Ph.D. candidate at the Computer Science department of the University of Twente. During November '92 until February '93 he was a visiting researcher at the University of Genoa (Italy). His research topics are physical design of relational and object-oriented databases.



Henk M. Blanken joined, after receiving his master degree in Mathematics, in 1966 Philips Computer Industry (Apeldoorn, The Netherlands). In 1971 he joined the University of Twente as an associate professor of Computer Science. From this university he received a Ph.D. degree in Computer Science in 1984. During a one-year visit to the IBM Scientific Center at

Heidelberg he contributed to the AIM/II project. His current research interests are non-standard database applications and storage structures.



Thiel Chang was born in Leiden, The Netherlands, in 1942. He received his degree in Mathematics from the University of Leiden. He joined the computer science arena in 1978. During the last 12 years he has held many senior management positions at large Software and Maintenance departments. Currently, he heads the Research & Development department of the

GAK. He has given invited talks on his research at conferences and working groups in Europe and served in program committees on conferences. He is a member of the ACM. His current research interests are software engineering, knowledge-based reusable software design, workflow management, expert systems, object-oriented databases and software development based on specification languages.