Heuristic Optimization of Physical Data Bases: Using a Generic and Abstract Design Model

By: Prashant Palvia

Palvia, P. "Heuristic Optimization of Physical Databases; Using a Generic & Abstract Design Model," Decision Sciences, Summer 1988, Vol. 19, No. 3, pp. 564-579.

Abstract:

Designing efficient physical data bases is a complex activity, involving the consideration of a large number of factors. Mathematical programming-based optimization models for physical design make many simplifying assumptions; thus, their applicability is limited. In this article, we show that heuristic algorithms can be successfully used in the development of very good, physical data base designs. Two heuristic optimization algorithms are proposed in the contest of a genetic and abstract model for physical design. One algorithm is based on generic principles of heuristic optimization. The other is based on capturing and using problem-specific information in the heuristics. The goodness of the algorithms is demonstrated over a wide range of problems and factor values.

Subject Areas: Heuristics, Management Information Systems, and Simulation.

Article:

INTRODUCTION

Data base design is a challenging activity and involves two phases: logical and physical design. Logical design involves the development of a logical data structure (LDS) for the task domain. Physical design is concerned with developing structures for placing data on secondary storage, given a specific LDS. While both phases require significant effort, the physical data base design phase is the focus of this article. The main concern in physical data base design is the efficiency of the physical design, which can be measured in a number of ways, such as storage requirement, response time and total system cost.

Prior works have generally dealt with and developed design/optimization models for specific aspects of physical data base design. These works include index selection [1] [15] [17], file structuring and models of file organization [19] [31], and record segmentation and structuring [16] [18] [24] [25]. Some of these models are reviewed in [30]. Jain states:

> While the attention to individual design problems results in elegant solutions, it is quite plausible that those individual solutions will have to be perturbed when the total data base is put together. [20, p. 217]

Thus, the need for comprehensive design models (which deal with the entire data base design problem rather than parts of it) cannot be overstated. Comprehensive physical design models fall under two categories: (1) those specific to and (2) those generic and independent of any particular logical data model and/or commercial DBMS. The first category offers the advantage of direct implementation on a particular DBMS, but is difficult to convert to another implementation. The second category allows more flexibility at the expense of added conversion requirements for a particular implementation. Optimization of physical design in the first category is reported in [29] for the hierarchical data model and in [12], [13], [20], [20], and [32] for the DBTG/CODASYL environment. For the second category, design models have been proposed in [3], [4], [6], [7], [22], [26], and [33]; optimization algorithms have been presented in [22] and [26].

The optimization algorithms presented in the above references use techniques of mathematical programming such as linear, integer, and goal programming. To use these techniques, the problem formulation must be kept tractable and several assumptions must be made about the problem characteristics. For example, in [20], clustering member records near owner is not considered; in [22], all clustered records are assumed to be on one page and the cost of accessing random files is not considered explicitly; in [26], the problem is formulated only for sequential access and very large buffers; in [32], sequential placement and clustering near the owner are excluded. Such assumptions are necessary to obtain clean formulations. However, the data base solution will be perturbed if these assumptions are relaxed. A solution to this dilemma is the use of heuristic procedures, which may be used efficiently in pursuit of good physical data base designs.

The primary aim of this article is to demonstrate that heuristic procedures can be successfully used to generate good, close to optimal, physical data base designs. The general application of heuristic procedures has been discussed in [11] and [34]; heuristic procedures have also been applied in the file design arena [16] [18]. For demonstration purposes, the physical data base design model chosen here is an abstraction of a generic model; thus, the results are applicable to different logical models and commercial DBMSs. Special features and laborious details have been omitted, but we believe the heuristics presented here can be custom-tailored to incorporate such features.

In the next section, we briefly discuss the factors that determine physical design. One of these is the physical design model itself (the abstract model used in this work is described later). The heuristic algorithms depend on a pairwise analysis of entities at the front end and a simulation program to determine the cost of a physical design. The essential features of the simulation program and pairwise analysis are next summarized and two heuristic algorithms are presented for physical data base design. The next section evaluates the algorithms on various dimensions across several values of the independent variables.

PHYSICAL DATA BASE DESIGN DETERMINANTS

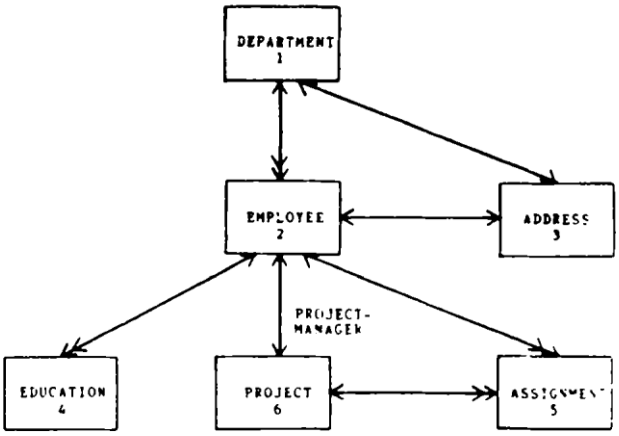
While many goals may be considered when designing the physical data base [20], one that is generally accepted is minimizing the total operational cost of using the data base [1] [6] [13] [16] [18] [25] [29] [32]. In this work we consider two costs: the storage costs of the data and the access costs to satisfy query requests. Access costs are estimated by the total number of page accesses from secondary memory. We concentrate on query requests, although limited kinds of update requests can be handled by the simulation program and the algorithms. The results should be applicable to many data base applications with low levels of data maintenance. Further, we believe the simulation program can be modified and the heuristic algorithms custom-tailored to explicitly consider updates.

'The costs of using the data base are influenced by four major factors: conceptual logical data structure (LDS), queries on the data, computer system characteristics, and physical design considerations. The LDS is based on prior design activity and we assume it to be given to us. The representation of the LDS in this work consists of entities and relationships between entities. The LDS may be directly obtained using a data structure diagram [2] or by simple conversion from an entity-relationship diagram [9]. Figure 1 represents the LDS for a personnel data base, adapted from [6] which we use to describe the heuristics. In Figure 1, there are two relationships between employee and project: the first assigns employees to projects via the assignment entity and the second is the direct, project-manager relationship. Other relationships in this LDS should be self-explanatory from this figure.

The queries on the data base may require selected instances of only one entity (e.g., data about certain employees) or data across several entities and their instances (e.g., data about certain departments and employees who work in these departments). The second type of retrieval is more complex and necessitates "traversing" several entities. The computer characteristics of relevance for this work include page size, pointer size, buffer size, and storage and access costs per unit.

Finally, many physical factors influence the physical data base design such as the access paths available and the data access/navigation strategy. Foremost among physical factors is the physical design model itself which describes the permissible alternative physical designs. Different commercial DBMSs use different physical design models (which are built into them). The next section describes general principles of physical data base design and, based on them, develops a generic and abstract physical data base design model. This abstract model will be used throughout the paper.

Figure 1: Example of a logical data structure (LDS).



PRINCIPLES OF PHYSICAL DATA BASE DESIGN AND AN ABST1RACT MODEL

Generic physical design models for (single) file design have been proposed, in increasing degree of comprehensiveness, in [19], [31], and [33]. The file design model of [33] was extended in [4] for physical data base design. In addition, there are some other models which address part(s) of the physical data base design problem in its own right (that is, without purposely building on the file design models). For example, aggregation/clustering concepts are discussed in [4], [6], and [30]. An abstract model was proposed in [22] which described the concepts of "evaluated," "indexed," "clustered," and "well-placed" records. A recent worthwhile effort is the transformation model (TM) [3], which models storage structures of several commercial DBMSs. (In our opinion, development of a sound and generic physical data base design model is an important issue, worthy of research in the data base area.)

In this paper, we use an abstract and generic physical design model based on some recognized principles that emerge from the current models [3] [4] [22] (24] [29]. These models suggest two fundamental principles for representing a relationship between two entities. The first principle is well known—indicate a relationship between two entities by storing appropriate pointers in the entities' instances. The pointers may be in the form of linked or inverted lists, or a combination of the two. (The pointers may be direct or symbolic.) The second principle is the concept of clustering/aggregation in which all related instances of one entity that are related to an instance of a second entity are clustered with or near the second entity instance.

The two concepts yield substantially different physical designs. The present abstract model captures the spirit of the two concepts; the many variations and details should be incorporated in more comprehensive models. Our initial experience in building the simulation program and developing the heuristics showed that these details do not have significant impact on the physical design. Moreover, the simulation program and the heuristic algorithms can be modified to adapt to the design model. The abstract model allows five ways to represent two entities, X and Y, and the relationship between them:

1. Create two record types, X and Y; X has pointers to Y.

2. Create two record types, X and Y; Y has pointers to X.

3. Create two record types, X and Y; both point to each other.

4. Create one record type X which will aggregate (cluster) Y instances. Aggregating in the abstract model is actualized by making the related Y instances part of the X record.

5. Create one record type Y that will aggregate the X instances. (The pointers above may be all symbolic or all direct.)

This model has strong parallels even in commercial DBMSs. For example, hierarchical and network systems incorporate the concepts of pointers and aggregations. Aggregation is supported in IMS by permitting hierarchical segments in the same data set and in network systems by storing MEMBER record types in OWNER area VIA SET and NEAR OWNER. Relational systems do not allow aggregation at a logical level; however, substantial efficiencies may be achieved by its use at a physical level as reported in [8] and [14]. In fact, many relational systems are now beginning to support clustering, such as SQL- and INGRES-based systems and RBASE.

When the abstract model is used, the total number of physical designs explodes exponentially when the number of entities and relationships in the LDS gets large. For example, with ten entities in the LDS there could be over a million design possibilities; with twenty entities, over a trillion design possibilities. Fortunately, we can curtail many of these options at the front end. In the experiments we conducted with the simulation program, we found that the optimal design is sensitive to the aggregation and pointer options but not much to a specific pointer option [27]. Thus, one of the three pointer options can be preselected for each related pair of entities using some guidelines or analysis. We selected the pointer option by using a pairwise analysis of related entities. The equations developed for pairwise analysis are described in the next section. (This pairwise analysis is also a front end to the heuristics described later.) Even with one pointer option (and two aggregation options) for each entity pair, the problem is still large (a 10-entity LDS may have over 10 thousand physical designs and a 20-entity LDS may have over 100 million designs).

With one pointer option and two aggregation options, a physical design can be fully specified by indicating the aggregations alone. A short-form notation can then be used to represent a physical design. In the short form, only the aggregator or absorber entity (also called physical parent) of each entity is named. A root entity does not have a physical parent; its parent is numbered 0. Table 1 shows some designs for the six-entity LDS of Figure 1.

Table 1: Some physical designs in short-form notation.

Entities	1	2	3	4	5	6	Explanation
Design 1	0	0	0	0	0	0	Unclustered flat-file design. All entities rooted; that is, 6 files.
Design 2	0	1	0	2	2	0	1 clusters 2; 2 in turn clusters 4 and 5. Entities 1, 3, 6 are rooted; that is, 3 files in physical data base.
Design 3	0	0	0	0	6	0	Only 6 clusters 5. All entities except 5 rooted; that is, 5 files.

The first design in Table 1 has all entities stored in independent files (with appropriate pointers to indicate relationships). This is a very common design strategy used by many designers. We call it the flat-file design and will refer to it from time to time.

SIMULATION PROGRAM AND PAIRWISE ANALYSIS

The two heuristic algorithms (described in the next section) use the simulation program to evaluate the costs of a physical design. Part of the pairwise analysis is used at the front end of both heuristics for pointer option determination for each entity pair. The pairwise analysis is also a major contributor to the second algorithm. We present an overview of the simulation program and the pairwise analysis (see (26] for details).

The simulation program written in ANSI FORTRAN V is comprehensive with the following major features:

Problem Specification. The parameters for the physical design problem are specified in four stages. First, the computer system characteristics (page size, direct pointer size, storage cost/character/period, and access cost/page) are defined. Second, the logical data structure is represented using suitable internal data structures. Third, necessary physical attributes are defined, including choice of symbolic or direct pointers, sequential or random access path, and provision for very large buffer sizes. Very large buffer sizes can affect the query processing strategy (discussed later). Finally, the activities (queries and limited updates) on the data base are entered. We assume that a limited update is one which will make twice the number of accesses than that of a query. For each query, the following parameters are specified: the frequency per period, the number of entities addressed by the query, the traversal path over the addressed entities if more than one entity is addressed, and the pre-selection and post-selection proportions of entity instances at each entity in the traversal path. Pre-selection refers to accessing only selected instances of an entity; post-selection refers to selecting among the already accessed instances.

Physical Design Specification. It can be entered either in short form or in long form. In long form, all files, aggregations, and pointers are explicitly specified. If specified in short form, the program will convert the specifications into long form by determining the pointer options. The program can either prompt the user to supply pointer information or invoke the pairwise analysis module to determine pointer options. At this point, the program also checks for feasibility of the physical design. A design may not be feasible because of data base consistency reasons (for example, two entities both cannot cluster each other) or because of user-/designer-supplied reasons.

Storage Cost Determination. For the feasible physical design entered above, the program determines the length of all record types (including pointers and aggregations), the number of records in a file, the file size, and the total storage requirement. This is a relatively straightforward task.

Determination of Page Accesses. The number of page accesses is determined for each query. While the algorithm embedded in the program is quite sophisticated (and complex) to account for different types of queries, we give an intuitive feel for the computation. (A full description is too lengthy for the scope of this paper.) First, the file containing the first entity addressed by the query is found. If this entity is rooted in the file, the number of records needed from this file is simply the number of records in the file multiplied by the pre-selection proportion. If the entity is not rooted, a formula (described in [26]) is applied to determine the required number of records from the file. In any case, for random access of the file, the number of page accesses for a given number of desired records is determined by the formula described in [28]. The required number of records is multiplied by the post-selection proportion to obtain the number of records finally selected. If the query addresses more than one entity (say, two), the related, desired instances of the second entity will be retrieved for each selected record of the first entity. This number is equal to the outdegree of the relationship between the first and second entity times the pre-selection proportion of the second entity. If the second entity is in the same file as the first, no new accesses are required (otherwise, more page accesses will be needed on the second file). The second file will be searched as many times as the selected records from the first file and each search will require a number of page accesses. If there are more than two entities in the traversal path of the query, this process will be repeated. It should be clear from the above description that the number of file searches (and page accesses) multiplies at each step of the traversal path of the query.

The above query processing strategy reflects the current practice (we will call it the normal processing strategy in future references). We implemented another (special) processing strategy in the simulation program; however, that requires very large, main memory buffers. With large buffers, no file is searched more than once. All required records from any given file are obtained in one search (necessitating large buffers). This is done by assembling all pointers (from the selected instances of the first file to the second file) prior to accessing the second file.

Obtaining Total Costs. This is a simple step. The storage requirements and page accesses are converted into costs by multiplying with appropriate cost factors. The total cost is then determined.

We now present essential features of pairwise analysis of entities, referred to in an earlier discussion. The pairwise analysis aids in early selection of pointer options and, more significantly, in the second heuristic algorithm. Pairwise analysis simplifies the complexity (at the risk of sacrificing some optimality) in the LDS. In pairwise analysis, only those entity pairs which are related need to be considered. Thus, when we say an entity pair, we always mean a *related* entity pair.

The main bottleneck to pairwise analysis is the handling of queries. If a query focuses on one or two entities, it fits naturally into the pairwise framework. If not, a query addressing N entities $(N > 2)$ is split into $N - 1$ pairwise queries. The pre- and post-selection proportions remain the same after the split. Thus, for a pair of entities (i,j) in the pairwise framework there may be four types of queries: a query focusing simply on entity i, simply on entity j, traversing from entity i to entity j, or traversing from entity j to entity i. Also, there are five physical designs for each entity pair (per our abstract design model).

Equations are developed in [26] for the storage requirement of each physical design of the (i,j) pair and for the number of page accesses for the different combinations of physical design, type of query, and processing strategy (i.e., 40 combinations). We report a few of these equations below.

Let N_i be the number of instances, l_i the instance length, and p_i the length of primary key of entity i. N_i, l_j, and p_j are the same for entity j. R_{ij} is the outdegree from entity i to entity j and R_{ji} is the outdegree from j to i. P is the direct pointer size, and PG is the page size. Further, let PR_i and PS_i be the pre- and post-selection proportions of a query focusing on entity i alone. Let PR_j and PS_j be the proportions if the query traverses to entity j as well.

Some storage requirement equations are:

A. Entities i, j stored in files i, j with pointers from i to j:
Length of file i record, $LI = lj$;
Length of file j record, $LJ = li + P \times R_{ij}$, with direct pointers
$= li + pj \times R_{ij}$, with symbolic pointers;
Total storage $= (N_i \times LI) + (N_j \times LJ)$;
Pages in file $i = (N_i \times L_i)/PG$; pages in file $j = (N_j \times L_j)/PG$.

B. Entity i aggregates entity j in file i:
Length of file i record, $LI = l_i + (l_j \times R_{ij})$;
Total storage $= (NI \times LI)$; pages in file $i = (N_i \times L_i)/PG$.

Some page access equations (for normal processing strategy, random access, and direct pointers) are:

A. Entities i, j stored in files i, j with pointers from i to j:
(1) Query on entity i alone:
The number of page accesses to retrieve k records from a file with n records and m pages is a function $F(n,m,k)$, where $F(n,m,k)$ is estimated as $m(1-(1-k/n)^{m/n})$ [27].
Page accesses $= F(N_i$, pages in file i, $N_i \times PR_i)$.
(2) Query from entity i to entity j:
Page accesses $= F(N_i$, pages in file i, $N_i \times PR_i)$
$+ N_i \times PR_i \times PS_i \times F(N_j$, pages in file j, $R_{ij})$.

B. Entity i aggregates entity j in file j:
(1) Query on entity i alone:
Page accesses $= F(N_i$, pages in file i, $N_i \times PR_i)$.
(2) Query from entity i to entity j:
Page accesses $= F(N_i$, pages in file i, $N_i \times PR_i)$.

In all of the above page access equations, the number of pages in a file are determined by the storage equations described earlier. Storage equations and access equations were developed in a similar manner for all combinations. By applying the equations, the cost of each of the five physical design options for each entity pair is obtained. Of the three pointer options, the least costly option can be selected at this time for each entity pair.

There are three costs associated with each entity pair (i, j): CP_{ij}, the cost of the best pointer option; CA_{ij}, the cost of entity i aggregating entity j; and CA_{ji}, the cost of entity j aggregating entity i.

We are now ready to present the heuristic algorithms.

THE HEURISTIC ALGORITHMS

While the author developed and evaluated many heuristic algorithms, two of the algorithms are presented that outperformed the others in terms of optimality and computational requirements. These are the generic greedy, foward inclusion algorithm (FWI) and the pairwise greedy algorithm (PWG).

The Generic Greedy Algorithm FWI

A greedy heuristic procedure for optimization is to maximize the solution improvement at each stage of the heuristic [11]. Typically, it works like this: select an arbitrary solution as the starting incumbent solution. Look at solutions near the incumbent and make the solution with the most improvement the new incumbent. Reiterate the procedure until the incumbent solution can no longer be improved. Such a heuristic was developed in [16] for the segmentation of flat files. The FWI algorithm presented below successively includes entities for aggregation based on the greedy principle. The algorithm has two types of references to the simulation program described earlier: FEASIBLE (to determine the feasibility of a physical design) and SIMCOST (to determine the cost of the physical design).

According to the design model, a physical design for an N-entity LDS is represented by an N-tuple: $(A_1, A_2, \ldots, A_n)$, where $A_i = 0$ if entity i is rooted and $A_i = j (j < > i)$ if entity i is clustered near entity j. Define a set U containing doubles (R_{ij}, U_{ij}), where R_{ij} is a relationship in the LDS and U_{ij} is a 0-1 binary variable. $U_{ij} = 1$ indicates a clustering has occurred along the relationship R_{ij}. Also, define a set PCR of triples (P, CIP, R_{ij}), where P itself is an N-tuple describing a physical design and CIP *is* cost improvement. The algorithm appears as follows:

```
A.  Preselect pointer options for all related entity pairs using pairwise analysis.
B.  Starting incumbent physical design is flat-file:
    PI=(0, 0, ..., 0), that is, A_i=0, for i=1, ..., N.
C.  Incumbent physical design cost PCI=SIMCOST(PI).
D.  Initially, U_ij=0, for all U_ij in set U.
E.  Define best cost improvement: BCI=+∞.
F.  DO WHILE (BCI>0 AND any U_ij in U=0)
    Make set PCR=null set.
    F.1       FOR each U_ij=0 in set U, DO
    F.1.1     Make P=PI with A_i=j.
    F.1.1.1   IF (FEASIBLE (P)=TRUE) THEN
                PC=SIMCOST(P)
                CIP=PCI-PC
                PCR=PCR U(P,CIP,R_ij)...(U meaning UNION)
              ENDIF
    F.1.2     Make Q=PI with A_j=i.
    F.1.2.1   IF (FEASIBLE(Q)=TRUE) THEN
                QC=SIMCOST(Q)
                CIQ=PCI-QC
                PCR=PCR U(Q,CIQ,R_ij)
              ENDIF
              ENDFOR
    F.2       Search PCR(BEST) triple in the PCR set so that
              CI of PCR(BEST)=Max[of all CI in PCR triples].
    F.3       BCI=CI of PCR(BEST)
    F.4       IF (BCI>0) THEN
                Make U_ij in U corresponding to R_ij of PCR(BEST)=1.
                PI=P of PCR(BEST)
                PCI=SIMCOST(PI)
              ENDIF
              ENDWHILE
G.  HEUR_BEST_SOLUTION=PI
    HEUR_SOLN_COST=PCI
H.  END
```

The Pairwise Greedy Algorithm (PWG)

The FWI heuristic, while a good one, requires computations of $O(R^2)$, R being the number of relationships in the LDS. This can be prohibitive for very large data base problems (say, with over 50 entities in the LDS). Less computationally intensive heuristics are derived by breaking down complexity and considering pairs of related entities. The pairwise information is then used in a greedy manner to develop an efficient algorithm. A similar strategy was used in [18] for the segmentation of flat files.

Recall that the pairwise analysis described earlier resulted in three costs for each related entity pair: CP_{ij} for the best pointer option, CA_{ij} for i aggregating j and CA_{ji}, for j aggregating i. This information is utilized by computing two clustering benefits for each entity pair (i,j): $CB_{ij} = CP_{ij} - CA_{ij}$, and $CB_{ji} = CP_{ij} - CA_{ji}$. Only positive clustering benefits are retained in a set SCB. The set SCB contains triples (C_{ij}, CB, U_{ij}), where C_{ij} indicates the clustering of entity j near entity i, corresponding to a positive clustering benefit CB. U_{ij} is a 0-1 variable, where $U_{ij} = 1$ indicates that entity j is clustered near entity i in the final physical design. The set SCB may have *up to* two triples for an entity pair (i, j): one for C_{ij} and one for C_{ji}.

As before, a physical design is represented by an N-tuple, $(A_1, A_2, ..., A_n)$. Another set PCC is defined with triples (P, CIP, C_{ij}), where P is the N-tuple for a physical design, CIP shows cost improvement, and C_{ij} shows the clustering considered. The algorithm is:

A. Preselect pointer options for all related entity pairs using pairwise analysis.
B. Starting incumbent physical design is flat-file:
   $PI=(0, 0, . .0)$, that is, $A_i = 0$, for $i = 1, ..., N$.
C. Incumbent physical design cost $PCI=SIMCOST(P/)$.
D. Make $U_{ij} = 0$ in all tuples in *set* SCB (i.e., no initial clustering).
E. Define best cost improvement = $+ \infty$.
F. DO WHILE $(BCI > 0$ AND any U_{ij} in SCB $=0)$
   Make set PCC= null set.
   F. 1 FOR each triple in SCB with $U_{ij} = 0$, DO
   Make $P=PI$ with $A_j = i$
   F.1 .1 IF (FEASIBLE(P)) THEN
   PC= SIMCOST(P)
   $CIP=PCI - PC$
   $PCC=PCC\ U(P, CIP, C_{ij})$
   ENDIF
   ENDFOR
F. 2 Search PCC(BEST) triple in PCC set, so that
   CI of PCC(BEST)= Max (over C/ in all PCC triples)
F. 3 $BCI = CI$ of PCC(BEST)
F. 4 IF $(BCI > 0)$ THEN
   Make U_{ij} of SCB set corresponding to C_{ij} of PCC(BEST)= I.
   $PI=P$ of PCC(BEST)
   PCI=SIMCOST(PI)
   ENDIF
   ENDWHILE
G. HEUR_BEST_SOLN $=PI$
   HEUR_SOLN_COST=PCI
H. END

HEURISTICS EVALUATION

As we remarked in the introduction, the optimal physical design depends on independent factors related to the logical data structure, the computer system, the queries, and the physical design model. A heuristic is good only if it can perform consistently well over a wide range of factor values. In order to evaluate the heuristics, a total

of twelve factors were developed (shown in Table 2). Also shown in Table 2 are the number of factor levels and value of each level.

Some factors need explanation. The proportion of entity instances addressed by a query depends on the operating environment. For example, a production/batch environment requires a large proportion of entity instances while an executive environment needs only a few selected instances. The distribution of queries refers to the degree of concentration on a few entities in the LDS. In addition, we developed a factor, conflict between queries, which may be one of the main reasons the design problem is difficult. There is a conflict between two queries when one traverses in one direction in the LDS and the other traverses in exactly the opposite direction. We do not separately include the frequency of queries as a factor; its effect is manifested in the other query factors. Finally, the large buffers allow the special query processing strategy described earlier.

An experimental design is needed to evaluate the effects of the above factors. Since a full-factorial design is impractical, a reasonable base case was defined by setting each quantitative factor at the middle value; each qualitative factor was set to reflect the current practice in data base design. The evaluation of the heuristic algorithms and the sensitivity analysis of the physical design were conducted by ranging each factor individually around the base case. If a factor was extremely sensitive, another base case was created at the new value of the factor and the process repeated. Two base cases were created for the two levels of buffer sizes and twenty problems were generated for each base case, a total of forty cases. The factor values for the base cases are shown in Table 2.

The algorithms' performance must be evaluated based on two criteria: their computational requirements and their ability to produce good, physical data base designs. The second criterion is measured by comparing the heuristic design to the optimal design and to commonly used designs.

Table 2: Experimental factors and parameters of the base case.

Factor and Factor Levels	Number of Levels	Base Case Value
Number of entitites in the LDS (4, 6, 8, and 10 entities)	4	6 entities
Number of queries (3, 6, and 9 queries)	3	6 queries
Average number of entities traversed per query (low, medium, and high)	3	Medium (3.17 entities)
Proportion of entity instances addressed by queries (low, medium, and high)	3	Medium
Distribution of queries on LDS (low, medium, and high concentrations)	3	Low concentration
Degree of conflict in queries (low, medium, and high)	3	Medium
Access/storage costs (access, storage and combination of both)	3	Access costs
Page size (2000 and 4000 characters)	2	2000 characters
Buffer size (normal, very high)	2	Normal for one base case, very large for second base case
Access path (sequential and random)	2	Random
Pointer type (symbolic/direct)	2	Direct
Mandatory vs. nonmandatory two-way pointer	2	Mandatory

COMPUTATIONAL REQUIREMENTS

The major computation made by each algorithm is when it invokes the simulation program to determine the cost of a physical design. Generally, the number of queries and their complexity is of the order of the LDS size (i.e., the number of entities *(N)* and relationships *(R)* in the LDS). Thus, it can be argued that the computation of the simulator is O(*R*). Since any algorithm will invoke the simulator, we will use the number of physical designs evaluated using the simulator as a measure of the computational requirements of the algorithm. (The computational burden for exhaustive enumeration is exponential.)

Lemma 1: The number of physical designs evaluated by the FWI algorithm is O(R^2).

FWI first evaluates the flat-file design. For each of the *R* relationships, it generates two designs by clustering each of the two entities ill the relationships, generating 2*R* designs. Next, FWI repeats the process for the remaining *R-1* relationships. In the worst case, the process continues to the end; usually, it terminates earlier. Thus, worst case designs generated =1+2*R* + 2(*R*— 1) + ... +2(1) = O(R). The average will be about half of that, so FWI =O(R^2).

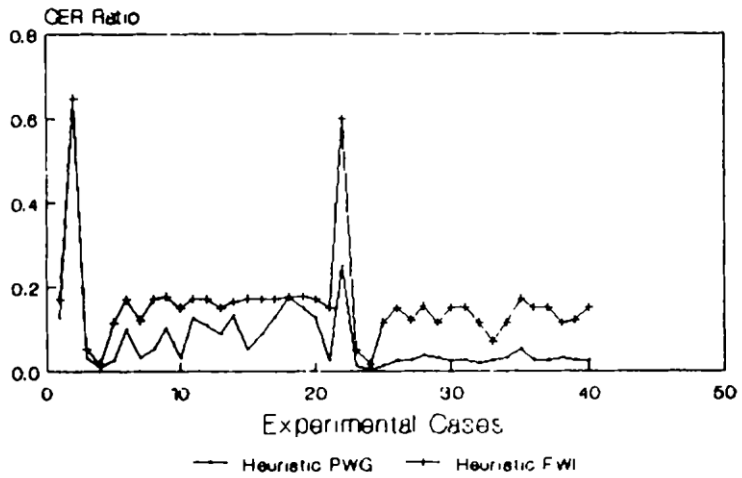
Lemma 2: The number of physical designs evaluated by the PWG heuristic is O(R^2) in the worst case and O(*R*) on the average.

The key to designs evaluated by PWG is the number of elements in the clustering benefits set SCB. At worst there are 2*R* elements in SCB, two for each relationship. However, at least half of these will conflict with the other half; thus, useful elements in SCB are at the most *R.* Empirically, the author has evidence that the useful elements average about O($R^{.5}$). If there are *K* entries in set SCB, the first iteration evaluates *K* designs in search of the best clustering. The next step evaluates *K*-1 designs. In the worst case:

$$\text{Number of designs evaluated} = K + K-1 + K-2 +, \ldots, +1 = O(K^2).$$

In the worst case, *K* = O*(R)* and PWG= O(R^2); generally, K = O($R^{.5}$) and PWG = O(*R*). In any case, the magnitude of computation in PWG is far less than in FW I. The computation effort of the two algorithms for the forty experimental cases is shown in Figure 2 by the CER ratio, which is the ratio of the number of designs evaluated using the algorithm to the total number of feasible designs. The first twenty cases use the normal processing strategy and the last twenty use the special processing strategy (with very large buffers). Cases 2 and 22 are for 4-entity LDS, cases 3 and 23 are for 8-entity LDS, cases 4 and 24 are for 10-entity LDS, and the remaining cases are for 6-entity LDS. Some observations drawn from Figure 2 are worth mentioning. Both algorithms examined are computationally efficient, with PWG outperforming FWI. The relative computational effort decreases with increasing LDS size. Finally, it appears that slightly more computation effort is required for the normal processing strategy.

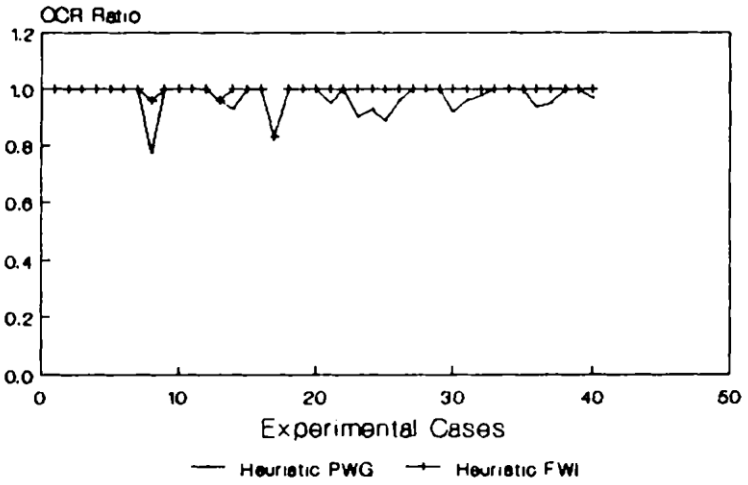
Figure 2: Computational effort of heuristics.



COMPARING WITH OPTIMAL SOLUTIONS

Optimal physical design can be obtained by exhaustive enumeration of feasible designs and cost-determination by the simulator. Because of a very large computational burden, this approach may be used only for small problems. (We could barely do this for a 10-entity LDS on a CDC Cyber-175; beyond that, it took hours.) The ratios OCR of optimal design cost to the heuristic design cost are reported in Figure 3. While both algorithms do very well, FWI is more consistent in producing more cost-effective designs. For the first 20 cases of normal processing strategy, the FWI algorithm produced the optimal design in 17 cases. In the other 3 cases, the cost differences from the optimal designs were only 4, 4, and 17 percent. The PWG algorithm obtained the optimal design in 16 of the 20 cases; the other 4 cases differed from the optimal design cost by 4, 7, 17, and 23 percent.

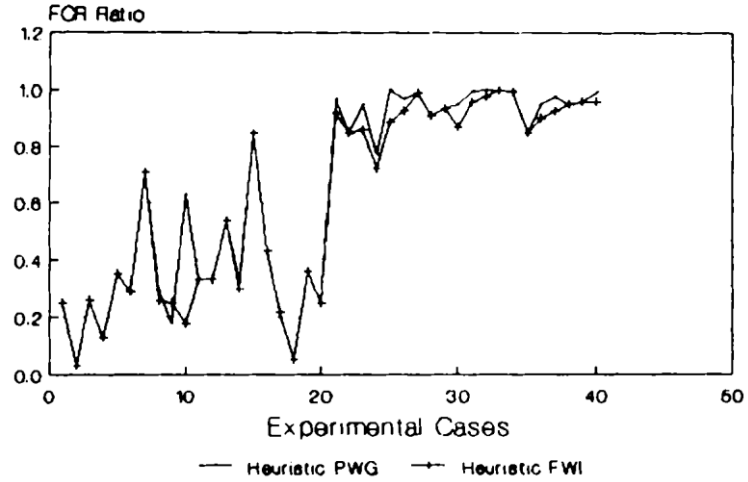
Figure 3: Optimization achieved by heuristics.



COMPARING WITH OTHER DESIGNS

Another manner of evaluating the heuristic designs is by comparing them with those designs used in the real world and developed by other designers/researchers. Many designers and DBMSs do not use any explicit clustering (or repeating groups) in their physical data base designs, selecting instead the flat-file design. Therefore, the ratio FCR of the heuristic design cost to the flat-file cost is shown in Figure 4. It is apparent from the plots that the flat-file is a poor design choice for the normal processing strategy and the heuristic designs offer substantial improvement. (The flat-file is reasonably good for the special processing strategy, but that happens rarely in practice.) With FWI the average cost improvement in the first 20 cases was 68 percent and 66 percent with PWG.

Figure 4: Flat-files versus heuristic designs.



Comparing our results with prior research efforts is difficult due to the lack of compatability of the various models and their assumptions. A limited comparison can be made to the work reported in [6], where guidelines were made for physical data base design based on LDS characteristics. These guidelines suggest the same set of designs (based on LDS guidelines) for all experimental cases. However, we found that different designs will be cost-effective, depending on the query characteristics. In several of the first 20 cases, different designs were more cost-effective and identified by both algorithms.

We have demonstrated that both generic and pairwise information-based heuristics can be used successfully to design the physical data base for a wide variety of problem types. The pairwise information-based algorithm requires less computational effort, at the expense of slightly reduced optimality. In the present context, we recommend using the FWI heuristic for small to medium problems (with less than 30 entities) and use the PWG heuristic for very large problems (greater than 30 entities, and especially in the range of 50 and over).

SUMMARY

Physical data base design can be a very complex and demanding activity. Mathematical programming approaches have had only limited success in this area. This paper, therefore, emphasizes the need for efficient heuristic algorithms to aid in the physical data base design process. Two heuristic algorithms are described in the contact of a generic and abstract design model: one is based on general principles of heuristic optimization and the other on problem-specific information. These algorithms provide good performance over a wide range of problems.

The physical data base designs generated using the abstract design model and the heuristic algorithms may be implemented on many commercial DBMSs. Hierarchical systems and network (DBTG) systems generally include the concepts of pointers and clustering, as reported in detail in [5], [10], and [23]. Examples of implementing the physical designs (generated using this work) on hierarchical and network systems are contained in [26]. Finally, the methods of this paper may be used in the initial creation of the physical data base, as well as its reorganization.

REFERENCES

[1] Anderson, H. D.. & Berra, P. B. Minimum cosi selection of secondary indexes for formatted files. *ACM Transactions on Database Systems,* 1977, *2,* 68-90.
[2] Bachman, C. W. Data structure diagrams. *Data Base,* 1969, 1, 4-10.
[3] Batory, D. S. Modeling the storage architectures of commercial database systems. *ACM Transactions on Database Systems,* 1985, *10,* 463-528.
[4] Batory, D. S., & Gotlieb, C. C. A unifying model of physical databases. *ACM Transactions on Database Systems,* 1982, 7. 509-539.
[5] BCS/CODASYL DDLC Data Base Administration Working Group. Draft specification of data storage description language Appendix. New York: ACM, 19'78.
[6] Carlis, J. V. *Investigation in the modeling and design of large, logically complex, multi-user database.* Unpublished doctoral dissertation, University of Minnesota, 1980.
[7] Carlis, J. V., & March. S. T. Computer-aided physical database design methodology. *Computer Performance,* 1983, *4,* 198-214.
[8] Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., Lorie, R. A., Mehl, G. W., Price, D. G., Putzolu. F, Schkolnik, M., Selinger, P. G., Stutz, D. R., Raiger, I. L., Wade, B. W., & Yost, R. A. A history and evaluation of system R. *Communications of the ACM,* 1981, *24,* 632-646.
[9] Chen, P. P. S. The entity-relationship model—towards a unified view of data. *ACM Transactions on Database Systems,* 1976, 1, 9-36.
[10] Date, C. J. *An introduction to database systems* (4th ed., Vol. 1). Reading, MA: Addison-Wesley, 1985.
[11] Evans, J. R. “Toward a framework for heuristic optimization. In *Proceedings of the 1979 AIM Conference.* Norcross, GA: AIIE, 1979.
[12] Gambino, T. J., & Gerritsen, R. A data base design decision support system. In *Proceedings of the 3rri International Conference on Very Large Databases.* New York: ACM, 1977.
[13] Gerritsen. R. 7bols for the automation of database design. In *Proceedings of the New York Symposium on Database Design.* New York: Springer-Verlag, 1978.
[14] Guttman, A., & Stonebroker, M. Using a relational database management system for computer aided design data. *Quarterly Bulletin of the IEEE Computer Society Technical Committee on Database Engineering,* 1982, 5, 21-28.
[15] Hsunmer, M., & Chan, A. Index selection in a self-adaptive data base management system. In *Proceedings of the ACM Special Interest Group on Management of Data.* New York: ACM, 1976.
[16] Hammer. M., & Niamir, B. A heuristic approach to attribute partitioning. In *Proceedings of the ACM Special Interest Group on Management of Data.* New York: ACM, 1979.
[17] Hoffer, J. A., & Kovacevic, A. Optimal performance of inverted files. *Operations Research,* 1982, *30,* 336-354.

[18] Hoffer, J. A., & Severance, D. G. The use of cluster analysis in physical database design. In Pro*ceedings of the 1st International Conference on Very Large Databases.* New York: ACM, 1975. [19) Hsiao, D., & Harary, F. A formal system for information retrieval from files. *Communications of the ACM,* 1973, *13,* 67-73.
[20] Jain, H. K. A comprehensive model for the storage structure design of CODASYL databases. *Information Systems,* 1984, 9, 217-230.
[21] Jain, H. K., & Krobock, J. R. Computer-aided system for the database storage structure design. *Information Management,* 1983, *6,* 337-349.
[22] Katz, R. H., & Wong, E. Resolving conflicts in global storage design through replication. *ACM Transactions on Database Systems,* 1983, *8,* 110-135.
[23) Kroenke, D. M. *Database processing* (2nd ed.). Chicago, IL: Science Research Associates, 1983.
[24] March, S. T Techniques for record structuring. *ACM Computing Surveys,* 1983, *15,* 45-80.
[25) March, S. T., & Severance, D. G. The determination of efficient record segmentation and blocking factors for shared data files. *ACM Transactions on Database Systems,* 1977, *2,* 279-296.
[26] Palvia, P. *Art analytical investigation into record structuring and physical database design.* Unpublished doctoral dissertation, University of Minnesota, 1984.
[27] Palvia, P. How sensitive is the physical database design?—Results of experimental investigation. In *Proceedings of the AFIPS National Computer Conference.* Reston, VA: AFIPS Press, 1987.
[28] Palvia, P., & March, S. T. Approximating block accesses in database organizations. *Information Processing Letters,* 1984, *19,* 75-79.
[29] Schkolnick, M. A clustering algorithm for hierarchical structures. *ACM Transactions on Database Systems,* 1977, *2,* 27-44.
[30] Schkolnick, M. A survey of physical database design methodology and techniques. In *Proceedings of the 4th International Conference on Very Large Databases.* New York: ACM, 1978.
[31] Severance, D. G. A parametric model of alternative file structures. *Information Systems,* 1975, 1, 51-55.
[32] Whang, K. Y., Weiderhold, G., & Sagalowicz, D. Physical design of network databases using the property of separability. In *Proceedings of the 8th International Conference on Very Large Databases.* New York: ACM, 1982.
[33]Yao, S. B. An attribute based model for database access cost analysis. *ACM Transactions on Database Systems,* 1977, *2, 45-67.*
[34) Zanakis, S. H., Evans, J. R. Heuristic "optimization": Why, when, and how to use it. *Interfaces,* 1981, *ii,* 84-91.