Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

A TRANSACTION-BASED APPROACH TO VERTICAL PARTITIONING FOR RELATIONAL DATABASES

W. W. Chu
I. T. Ieong

December 1990
CSD-900043

A Transaction-Based Approach to Vertical Partitioning for Relational Databases*

W. W. Chu and I. T. Ieong
Department of Computer Science
University of California, Los Angeles

December 10, 1990

Abstract — A new approach to vertical partitioning is proposed to partition the attributes of a relation according to the set of transactions. The objective of vertical partitioning is to minimize the total number of disk accesses in the system. Since transactions have more semantic meaning than attributes, it allows optimizing the partitioning based on a selected set of important transactions. An optimal binary partitioning algorithm, OBP, with complexity of $O(2^n)$ is presented where n is the number of transactions in the system. A heuristic algorithm BPi with complexity varying from $O(n)$ to $O(2^n)$ is also developed to handle systems with a large number of transactions. Our experimental results show that both the algorithms OBP and BPi yield results comparable with that of the global optimum obtained from exhaustive search.

1 Introduction

In a relational database, system throughput is affected by the rate which the data can be retrieved from the disks. To enhance the performance of a database system, those attributes that are frequently requested by transactions may be grouped together to reduce

*This research is supported by Hughes Aircraft Company contract S8-544866-X21 and Hughes Micro contract 89-005.

the number of pages transferring from disks to the main memory for processing. Vertical partitioning assigns the attributes of a logical relation into fragments which minimizes the total number of disk accesses requested by the transactions. However, the number of sets of fragments that the attributes can be assigned to is too large. It is equal to the mth Bell number for a relation with m attributes, which is roughly on the order of $O(m^m)$[3][6]. It is infeasible to solve the problem optimally by an exhaustive approach, even for a relatively small m.

Hoffer[4] developed a nonlinear, zero-one integer programming technique for determining the vertical fragmentation that minimizes storage, retrieval, and update costs under capacity constraints assigned to each subfile. Hoffer and Severance[5] developed an algorithm by permuting the attributes and grouping the ones with "high affinity" together. Affinity among attributes is a measure of the extent that they are being referenced together. Navathe et al.[6]extended this work and proposed a two phase vertical partitioning algorithm. The first phase sets up an attribute pairwise affinity matrix. A clustering algorithm is used to perform a pairwise permutation of rows and columns to put the matrix in a semiblock diagonal form. The matrix is then divided into two sets of attributes which minimizes the accessing of irrelevant attributes. An affinity graph can also be constructed with the edge value representing the affinity between two attributes from the attribute affinity matrix. A linearly connected spanning tree is then formed. By considering a cycle as a fragment, the algorithm generates all meaningful fragments in one pass[7]. Cornell and Yu[1][2] developed an approach to vertical partitioning based on the physical design of relational databases. An integer linear programming technique was used to develop an optimal binary partitioning algorithm which minimizes the number of disk accesses by assigning attributes to physical segments.

All previous research used an attribute as the basic manipulation unit which lacks semantic information. To remedy this shortcoming, we consider the attributes accessed by a transaction as a unit. Using this approach, an optimal binary vertical partitioning algorithm with the complexity of $O(2^n)$ is developed as compared to the attribute-based algorithm of $O(2^m)$ where n and m are the number of transactions and attributes respectively in the system. Since usually $n < m$, the proposed algorithm yields a reduction of computation complexity of $O(2^{m-n})$. Because transactions have more semantic meanings than attributes, this transaction-based approach allows the selection and the optimization of only the important transactions (e.g., frequently used); thereby, reducing the effective

n. A heuristic algorithm BPi with significantly less complexity (varying from $O(n)$ to $O(2^n)$) is also proposed for systems with a large number of transactions.

In this paper, we shall first present the concept of transaction-based vertical partitioning and then introduce the notion of reasonable and unreasonable cuts and their groupings. Next, we present a theorem stating that an unreasonable cut can be improved by a reasonable cut. Based on this property, two vertical partitioning algorithms are developed: the optimal binary partitioning algorithm OBP and the heuristic partitioning algorithm BPi. Finally, we compare the performance and behavior of the proposed algorithms with the known algorithms as well as the global optimal solution obtained from exhaustive search.

2 Vertical Partitioning

Vertical partitioning depends on attribute access patterns and the number of scans through the relation in each transaction invocation. Usually, there are two types of scans: index and segment scans. For an index scan, the average number of pages retrieved by a transaction depends on the average fraction of tuples satisfying the predicate for the indexed attribute. If the indexed attribute is the clustering attribute, it is called a clustered index scan, otherwise, it is an unclustered index scan. A segment scan retrieves all pages of a relation. Since reading is sequential, several pages of the relation can be prefetched in a single disk access. For a given transaction type, the query optimizer decides the type of scans to be used.

2.1 Objective Function

The cost in terms of the number of disk accesses of a transaction depends on the number of fragments it accesses, its access frequency, and the access method used. The cost can be estimated as follows[2]:

- Unclustered index: Number of accesses = $(cardinality)(selectivity)$
- Clustered index: Number of accesses = $\frac{(cardinality)(selectivity)(length\ \ of\ \ tuple)}{(page\ \ size)}$
- Segment scan: Number of accesses = $\frac{(cardinality)(length\ \ of\ \ tuple)}{(page\ \ size)(prefetch\ \ blocking\ \ factor)}$

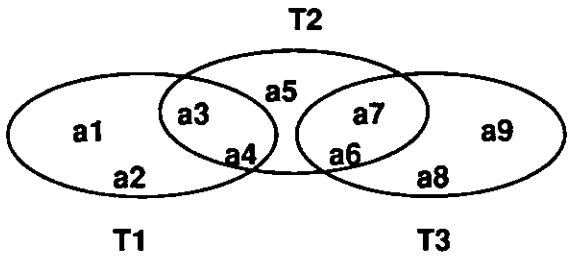
The total number of disk accesses by all the transactions is the sum of the number of disk accesses incurred by each transaction. The objective function is to minimize the total number of disk accesses.

2.2 Properties of Transaction-Based Partitioning

2.2.1 Reasonable and Unreasonable Cuts

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
T_1	1	1	1	1	0	0	0	0	0
T_2	0	0	1	1	1	1	1	0	0
T_3	0	0	0	0	0	1	1	1	1

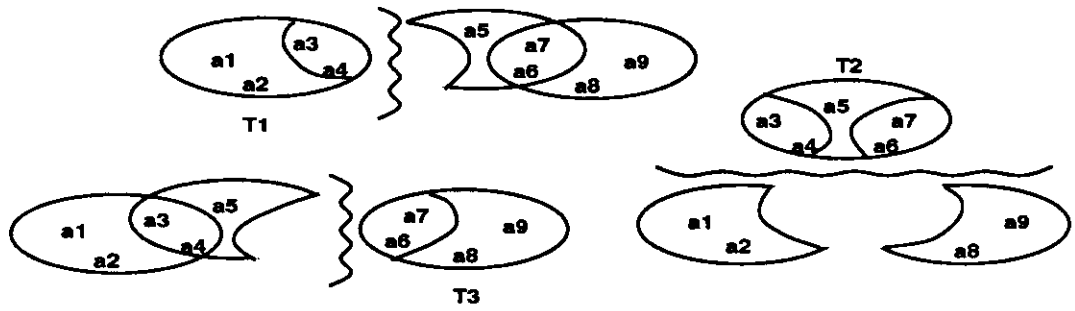
(a) The access pattern matrix for three transactions.



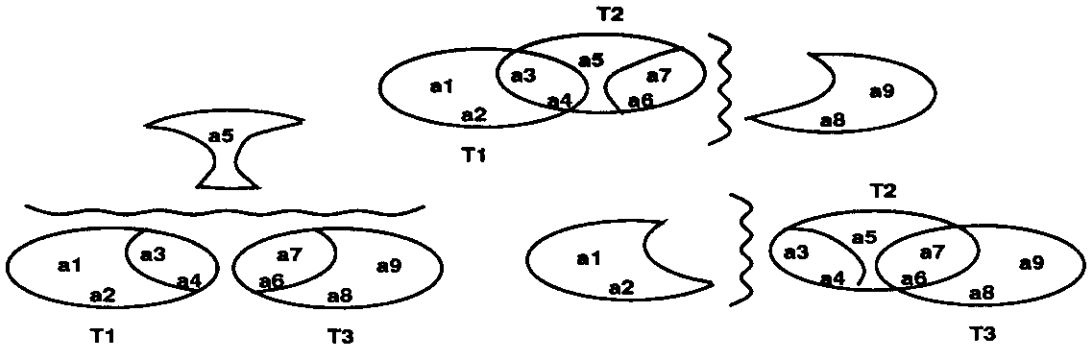
(b) An attribute intersection graph for three transactions.

Figure 1: Access pattern matrix and attribute intersection graph.

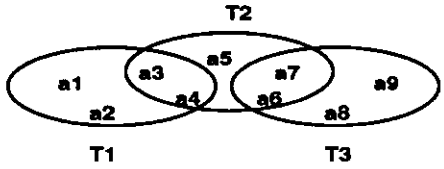
A *self-contained fragment*, T_i, is the set of attributes transaction i accesses. The union of such self-contained fragments is referred to as a *contained fragment*. A binary cut that partitions the attributes into two sets in which at least one of them is a contained fragment is called a *reasonable cut*. For the example in Figure 1, the self-contained fragment T_1 for transaction 1 is composed of attributes a_1, a_2, a_3, and a_4. The union of self-contained fragments T_1 and T_2 is a contained fragment. The reasonable cuts among the binary cuts are shown in Figure 2. We shall define group i reasonable cuts when their contained fragments consist of exactly i self-contained fragments. All binary cuts that are not reasonable cuts are called *unreasonable cuts*.



(a) Group 1 reasonable cuts



(b) Group 2 reasonable cuts



(c) Group 3 reasonable cut

Figure 2: Reasonable cuts for the three transactions in Figure 1

Lemma 1: For a system with n transactions, there are $\begin{pmatrix} n \\ i \end{pmatrix}$ group i reasonable cuts.

Proof: Straightforward.

Lemma 2: There are $2^n - 1$ reasonable cuts to binary partition a relation.

Proof: Since there are $\begin{pmatrix} n \\ i \end{pmatrix}$ group i reasonable cuts, the number of reasonable cuts to binary partition a relation is $\sum_1^n \begin{pmatrix} n \\ i \end{pmatrix} = 2^n - 1$. $Q.E.D.$

For the example with nine attributes and three transactions as shown in Figure 1,

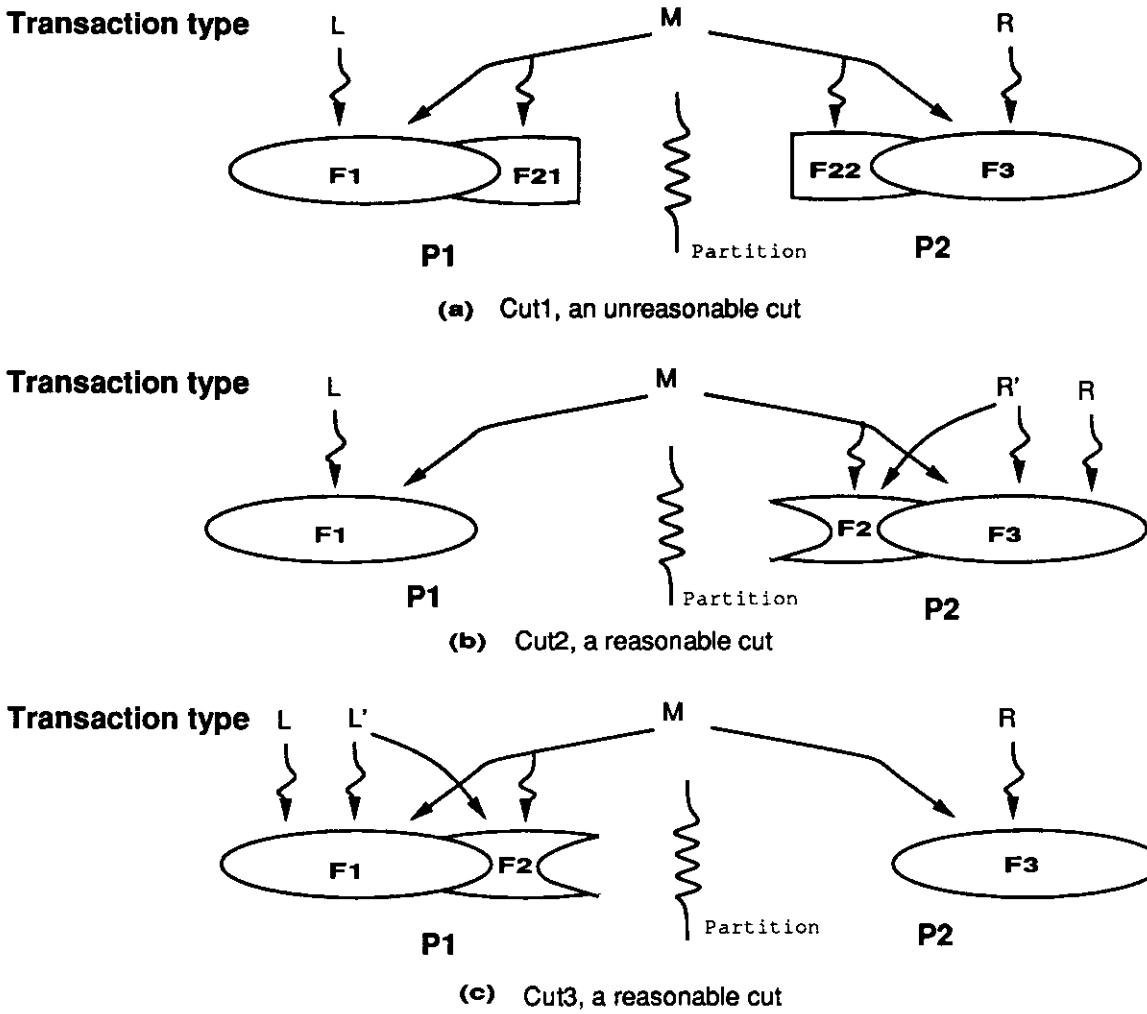


Figure 3: Reasonable and unreasonable cuts of a relation.

there are $2^9 - 1$ possible binary cuts and $2^3 - 1 = 7$ reasonable cuts (Figure 2). Three of them are in group 1, three in group 2, and one in group 3. We shall now show that the optimal binary cut is derived from the reasonable cuts.

Theorem 1: For a given unreasonable cut of a relation, there exists at least one reasonable cut that yields less or equal cost (in terms of the total number of disk accesses) than the unreasonable cut.

Proof: After partitioning a relation by an unreasonable cut as shown in Figure 3a, the attributes are divided into two sets, P_1 and P_2. Let type L (Left) and R (Right) transactions be the ones that access attributes in P_1 and P_2, respectively. F_1 (F_3) is the set

of attributes accessed by type L (R) transactions and F_{21} ($F_{22} = F_2 - F_{21}$) is the set of attributes that are not accessed by type L (R) transactions in P_1 (P_2). Type M (Middle) transactions access attributes both in P_1 and P_2. Let us move the partition to the boundary of F_1. The number of type L transactions remains the same; however, the number of type M transactions may decrease, since some of the attributes are moved from P_1 to P_2. After moving the partition to the boundary of F_1, those type M transactions that only access F_{21} no longer need to access attributes in P_1. We call these transactions type R' transactions, as shown in Figure 3b. Likewise, L' is defined as shown in Figure 3c. Let C_1, C_2, and C_3 be the cost functions for cut_1, cut_2, and cut_3, respectively. Cut_2 is obtained from cut_1 by moving the partition to the boundary of F_1. Cut_3 is obtained by moving the partition to the boundary of F_3. The cuts in Figures 3b and 3c are reasonable cuts.

Let Lc, Lu, and Ls be the set of type L transactions using clustered index, unclustered index, and segment scans, respectively. In the same manner, we shall define Mc, Mu, Ms, Rc, Ru, Rs, Lc', Lu', Ls', Rc', Ru', and Rs' for type M, R, L', and R' transactions. For simplicity in analysis, we use the real number for the number of disk accesses. The cost function for cut_1, C_1 is:

$$C_1 = D_1 \sum_{x \in Lu \cup Ru} f_x s_x + 2D_1 \sum_{x \in Mu} f_x s_x$$

$$+D_2(l(F_1) + l(F_{21})) \sum_{x \in Lc} f_x s_x + D_2(l(F_1) + l(F_2) + l(F_3)) \sum_{x \in Mc} f_x s_x$$

$$+D_2(l(F_{22}) + l(F_3)) \sum_{x \in Rc} f_x s_x + \frac{D_2}{b}(l(F_1) + l(F_{21})) \sum_{x \in Ls} f_x$$

$$+ \frac{D_2}{b}(l(F_1) + l(F_2) + l(F_3)) \sum_{x \in Ms} f_x + \frac{D_2}{b}(l(F_{22}) + l(F3)) \sum_{x \in Rs} f_x \qquad (1)$$

where

$D_1 = cardinality$, number of tuples in the relation

$D_2 = cardinality/pagesize$

$l(F_k)$ = tuple length of fragment F_k

b = prefetch blocking factor

s_i = selectivity for transaction i

r_i = relative execution frequency of transaction i

n_i = number of scans of the relation in each invocation of transaction i

$f_i = r_i n_i$, the relative access frequency of transaction i

C_1 is the sum of the costs incurred by the transactions using different access methods. The first two terms in (1) are the costs due to types L, M, and R transactions using the unclustered index scan. The next three terms are the costs for the clustered index scan and the remaining terms are the costs due to the segment scan. In the same manner, the cost functions for Cut_2, C_2, and Cut_3, C_3, are:

$$C_2 = D_1 \sum_{x \in Lu \cup Ru \cup Ru'} f_x s_x + 2D_1 \sum_{x \in Mu - Ru'} f_x s_x$$
$$+ D_2 l(F_1) \sum_{x \in Lc} f_x s_x + D_2 (l(F_1) + l(F_2) + l(F_3)) \sum_{x \in Mc - Rc'} f_x s_x$$
$$+ D_2 (l(F_2) + l(F_3)) \sum_{x \in Rc \cup Rc'} f_x s_x + \frac{D_2}{b} l(F_1) \sum_{x \in Ls} f_x$$
$$+ \frac{D_2}{b} (l(F_1) + l(F_2) + l(F_3)) \sum_{x \in Ms - Rs'} f_x + \frac{D_2}{b} (l(F_2) + l(F3)) \sum_{x \in Rs \cup Rs'} f_x \quad (2)$$

$$C_3 = D_1 \sum_{x \in Lu \cup Lu' \cup Ru} f_x s_x + 2D_1 \sum_{x \in Mu - Lu'} f_x s_x$$
$$+ D_2 (l(F_1) + l(F_2)) \sum_{x \in Lc \cup Lc'} f_x s_x + D_2 (l(F_1) + l(F_2) + l(F_3)) \sum_{x \in Mc - Lc'} f_x s_x$$
$$+ D_2 l(F_3) \sum_{x \in Rc} f_x s_x + \frac{D_2}{b} (l(F_1) + l(F_2)) \sum_{x \in Ls \cup Ls'} f_x$$
$$+ \frac{D_2}{b} (l(F_1) + l(F_2) + l(F_3)) \sum_{x \in Ms - Ls'} f_x + \frac{D_2}{b} l(F3) \sum_{x \in Rs} f_x \quad (3)$$

Comparing the cost functions between C_1 and C_2, and C_1 and C_3, we have

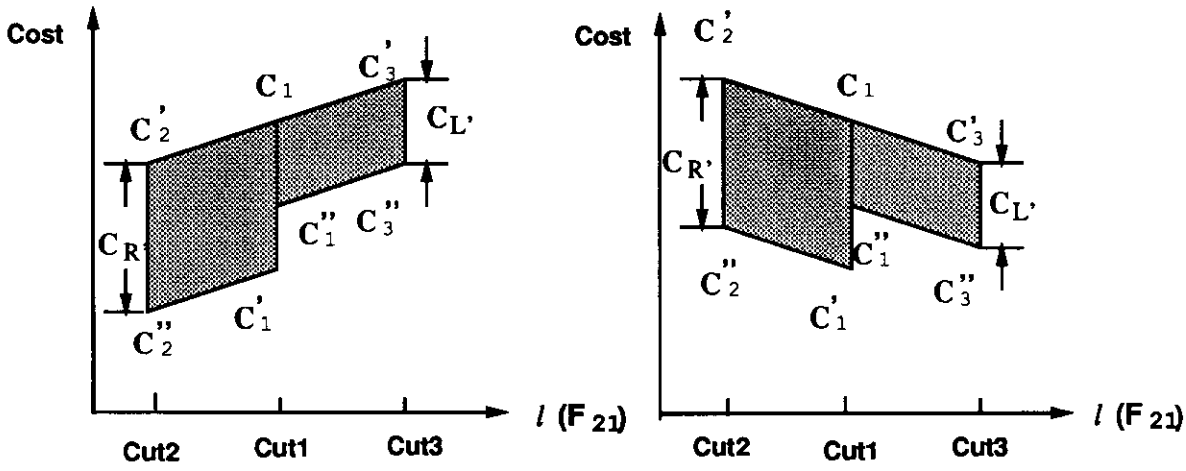
$$C_1 - C_2 = C_{R'} + D_2 \alpha l(F_{21}) \quad (4)$$

$$C_1 - C_3 = C_{L'} - D_2 \alpha l(F_{22}) \quad (5)$$

where

$$C_{R'} = D_1 \sum_{x \in Ru'} f_x s_x + D_2 l(F_1) \sum_{x \in Rc'} f_x s_x + \frac{D_2}{b} l(F_1) \sum_{Rs'} f_x$$

$$C_{L'} = D_1 \sum_{x \in Lu'} f_x s_x + D_2 l(F_3) \sum_{x \in Lc'} f_x s_x + \frac{D_2}{b} l(F_3) \sum_{Ls'} f_x$$



(a) $\alpha \geq 0$ (b) $\alpha < 0$

Figure 4: The cost region of an unreasonable cut, cut_1.

$$\alpha = \sum_{x \in Lc} f_x s_x + \frac{1}{b} \sum_{x \in Ls} f_x - \sum_{x \in Rc} f_x s_x - \frac{1}{b} \sum_{x \in Rs} f_x$$

Since type L' and R' transactions only need to access P_1 or P_2, this reduces access cost. Let $C_{L'}$ and $C_{R'}$ be the corresponding cost reduction. Since $C_{R'}$ and $C_{L'} \geq 0$, if $\alpha \geq 0$, then $C_1 \geq C_2$. Likewise, $C_1 > C_3$, if $\alpha < 0$. This implies that there exits a reasonable cut with equal or less cost than that of the unreasonable cut. *Q.E.D.*

2.2.2 Cost Region of an Unreasonable Cut

Let us now discuss the change of cost function when we move the partition from an unreasonable cut to the reasonable cuts. As we move the partition to the left, some of the type M transactions may become type R' transactions; likewise, as we move the partition to the right, some of the type M transactions may become type L' transactions. The change of cost function as we move the partition to either direction depends on α and the number of type R' and L' transactions. Figure 4 represents the cost region (in terms of total number of disk accesses) when $\alpha \geq 0$ and $\alpha < 0$. For convenience in presentation, the cost is presented as a continuous function. C_2' on the left upper corner represents the case when all the type M transactions that access P_1 need to access both F_1 and F_{21}. Similarly, the point C_3' represents the case when all the type M transactions that access P_2 are required to access F_{22} and F_3. C_2'' represents the case when all the

type M transactions that access P_1 need only access F_{21}, and C_3'' represents the case when all the type M transactions that access P_2 need only access F_{22}. As a result, the difference between C_2' and C_2'' (C_3' and C_3'') is $C_{R'}(C_{L'})$ which equals the amount of cost reduction from type R' (L') transactions.

From (1) and (4), we know that $C_2'C_3'$ in Figure 4a is a linear increase function of $l(F_{21})$ with a slope of $D_2\alpha$ which represents that there are no type L' and R' transactions. As we move the partition to the left, this reduces F_{21} while increasing F_{22}. Since $\alpha \geq 0$, the cost function decreases as F_{21} reduces. For an unreasonable cut that both type L' and R' transactions exist, as we move the partition to the left, the number of R' transactions increases which reduces the cost function. Therefore, the cost function is a monotonic increase function as F_{21} increases inside the region of $C_1C_2'C_2''C_1'$. Note that $C_2''C_1'$ ($C_1''C_3''$) has the same slope as $C_2'C_1$ (C_1C_3'). When we move the partition to the right, the cost function is bounded by the region of $C_1C_3'C_3''C_1''$ and the cost improvement depends on the combined effect of cost decrease due to the change from type M to L' transactions and the cost increase due to $\alpha \geq 0$. The cost region in Figure 4b is similar to that in Figure 4a except $\alpha < 0$.

2.2.3 Fragments with Tuple Identifier

To identify an original tuple in different fragments of a relation, it is necessary to identify each tuple in all of the fragments. This can be accomplished by replicating the primary key or tuple identifier (i.e., system controlled identifier for each tuple of the original relation) for all the fragments of the relation.

Corollary 1: Theorem 1 is true when adding a primary key or tuple identifier to each tuple for all the fragments of a relation.

Proof: Let the length of the primary key or tuple identifier attached to each tuple be the same and denoted by $l(I)$. Let C_1^*, C_2^*, and C_3^* be the costs with primary key or tuple identifier for cut_1, cut_2, and cut_3. Then,

$$C_1^* = C_1 + D_2 l(I) \sum_{x \in Lc} f_x s_x + 2D_2 l(I) \sum_{x \in Mc} f_x s_x + D_2 l(I) \sum_{x \in Rc} f_x s_x$$

$$+\frac{D_2}{b}l(I)\sum_{x\in Ls}f_x+\frac{2D_2}{b}l(I)\sum_{x\in Ms}f_x+\frac{D_2}{b}l(I)\sum_{x\in Rs}f_x \tag{6}$$

$$C_2^*=C_2+D_2l(I)\sum_{x\in Lc}f_xs_x+2D_2l(I)\sum_{x\in Mc-Rc'}f_xs_x+D_2l(I)\sum_{x\in Rc\cup Rc'}f_xs_x$$
$$+\frac{D_2}{b}l(I)\sum_{x\in Ls}f_x+\frac{2D_2}{b}l(I)\sum_{x\in Ms-Rs'}f_x+\frac{D_2}{b}l(I)\sum_{x\in Rs\cup Rs'}f_x \tag{7}$$

$$C_3^*=C_3+D_2l(I)\sum_{x\in Lc\cup Lc'}f_xs_x+2D_2l(I)\sum_{x\in Mc-Lc'}f_xs_x+D_2l(I)\sum_{x\in Rc}f_xs_x$$
$$+\frac{D_2}{b}l(I)\sum_{x\in Ls\cup Ls'}f_x+\frac{2D_2}{b}l(I)\sum_{x\in Ms-Ls'}f_x+\frac{D_2}{b}l(I)\sum_{x\in Rs}f_x \tag{8}$$

$$C_1^*-C_2^*=C_1-C_2+D_2l(I)(\sum_{x\in Rc'}f_xs_x+\frac{1}{b}\sum_{x\in Rs'}f_x) \tag{9}$$

$$C_1^*-C_3^*=C_1-C_3+D_2l(I)(\sum_{x\in Lc'}f_xs_x+\frac{1}{b}\sum_{x\in Ls'}f_x) \tag{10}$$

From (9) and (10), we know that if $\alpha\geq 0$, then $C_1^*\geq C_2^*$; otherwise, $C_1^*>C_3^*$. $Q.E.D.$

Theorem 2: The optimal binary vertical partitioning of a relation has complexity $O(2^n)$.

Proof: Based on Lemma 2 and Theorem 1, an unreasonable cut can be improved by one of the reasonable cuts. Since there are 2^n-1 reasonable cuts for a system with n transactions, the computation complexity of the optimal binary vertical partitioning is $O(2^n)$. $Q.E.D.$

3 The Algorithms

By successive binary partitioning of a relation with reasonable cuts, based on Theorem 2, an optimal binary partitioning algorithm, OBP, with complexity $O(2^n)$ can be developed as will be discussed below.

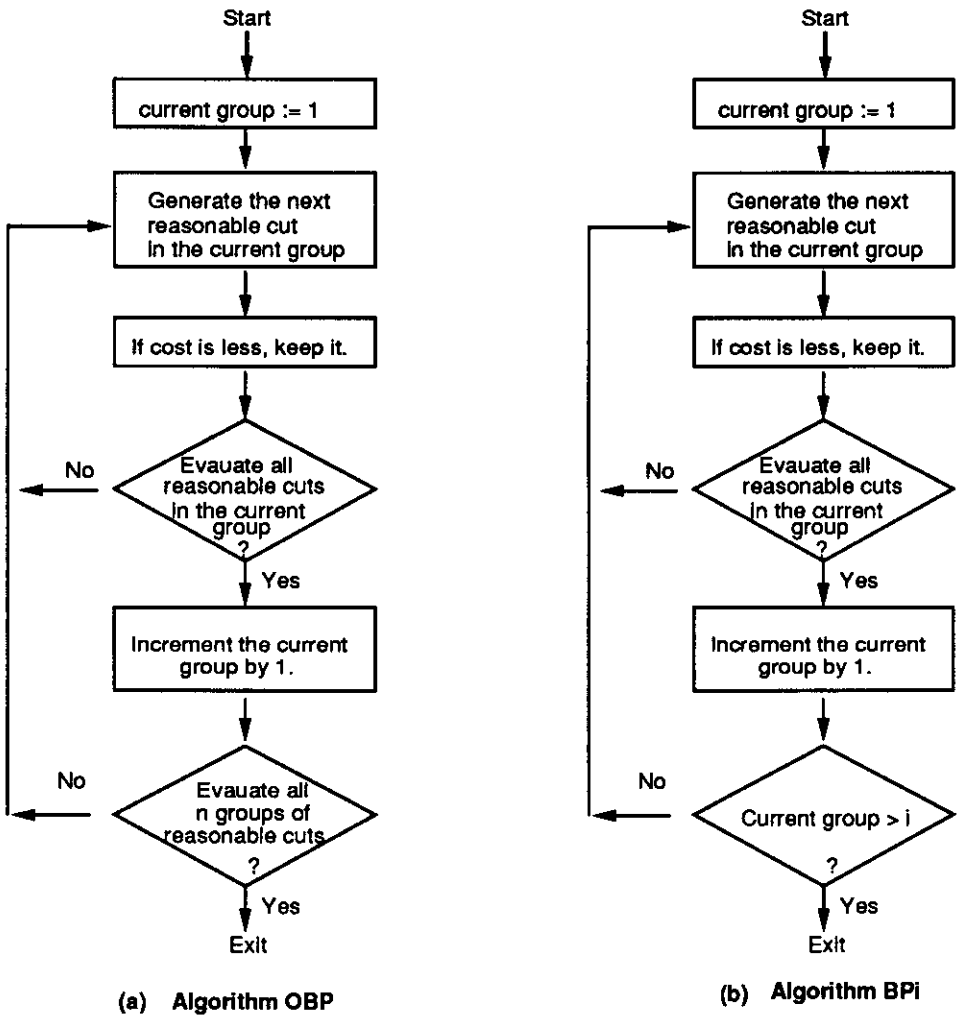


Figure 5: Flowcharts for the partitioning algorithms.

3.1 Optimal Binary Partitioning (OBP) Algorithm

The algorithm first generates all the first group reasonable cuts, keeping the one that yields the lowest cost, and then repeats the same process for the second group of reasonable cuts. This process is repeated until all the n groups of reasonable cuts are evaluated as shown in the flowchart in Figure 5a. Figure 6 shows the corresponding pseudo code of the algorithm.

The procedure `evaluate_cost()` in line 2 evaluates the total number of disk accesses requested by all transactions for fragment F. The procedure `NEXKSB()` [8] in line 7 generates the next j out of n self-contained fragments represented by their transaction number

```
Algorithm OBP(F, F1, F2)

(0)BEGIN
(1)     better := 'no better'; {Improvement indicator set to 'no better'.}
(2)     mincost := evaluate_cost(F); {Evaluate the cost for the fragment F.}
(3)     FOR j := 1 to n DO { Evaluate all n groups of reasonable cuts.}
(4)     BEGIN
(5)       flag := false;
(6)       REPEAT
(7)         NEXKSB(n, j, T, flag);
              {Generate the next j out of n combination and store
               it in the first j elements of vector T. flag is set
               to: false, for the first call; true, if the current
               combination is not the last one; false, if the current
               combination is the last.  See [8] for more details.}
(8)         tempF1 := reasonable_cut(j, T);
              {tempF1 stores all the attributes accessed by
               the j transactions indicated in T.}
(9)         tempF2 := F - tempF1;
              {tempF2 stores the rest of the attributes.}
(10)        currentcost := evaluate_cost(tempF1) + evaluate_cost(tempF2);
              {Evaluate the cost of the current binary partition.}
(11)        IF currentcost < mincost THEN
              {If there is an improvement, keep it.}
(12)        BEGIN
(13)          F1 := tempF1; F2 := tempF2;
(14)          mincost := currentcost;
(15)          better := 'better';
(16)        END;
(17)      UNTIL flag = false;
(18)    END;
(19)    RETURN(better);
(21)END;
```

Figure 6: Pseudo code for algorithm OBP.

```
Algorithm BPi(F, F1, F2)

(0)BEGIN
 .
 .
(3)     FOR j := 1 to i DO { Evaluate up to ith group reasonable cuts.}
 .
 .
 .
(21)END;
```

Figure 7: Pseudo code for algorithm BPi.

and stores them in the first j elements of T. The function `reasonable_cut()` in line 8 separates all the attributes in the contained fragments indicated by T from the relation and assigns them to `tempF1`. Since the computation complexity of algorithm OBP grows exponentially with n, using OBP can be quite time consuming for systems with a large number of transactions. Therefore, we shall propose a heuristic algorithm that greatly reduces the computation complexity for solving the vertical partitioning problem.

3.2 Heuristic Algorithm BPi

To reduce the computation complexity for systems with a large number of transactions, we propose to evaluate up to the group i reasonable cuts (see flowchart in Figure 5b). We shall call this heuristic algorithm BPi, $i = 1, ..., n$. The number of reasonable cuts evaluated is $\sum_{j=1}^{i} \binom{n}{j}$, for $i = 1, ..., n$. The complexity of this algorithm depends on i and varies from $O(n)$ to $O(2^n)$. The search space of BPi is the union of the search space of BP$(i-1)$ and all the group i reasonable cuts. The pseudo code for algorithm BPi is almost identical to that of algorithm OBP (see Figure 7) with the exception of line 3 that evaluates only up to the ith group of reasonable cuts, $1 \leq i \leq n$.

To generate all the fragments, we shall apply our binary partitioning algorithms repetitively as shown in the pseudo code in Figure 8.

```
Algorithm Repetitive_Binary_Partitioning( A_Binary_Partitioning_Algorithm, F)
(0)BEGIN
(1)     IF A_Binary_Partitioning_Algorithm(F, F1, F2) = 'better'
(2)     THEN BEGIN
(3)            Repetitive_Binary_Partitioning( A_Binary_Partitioning_Algorithm, F1);
(4)            Repetitive_Binary_Partitioning( A_Binary_Partitioning_Algorithm, F2);
(5)          END
(6)     ELSE print(F);
(7)END;
```

Figure 8: Binary partitioning algorithm with repetition.

4 Experimental Results

In this section, we shall compare the performance of the proposed algorithms with that of the algorithms based on the attribute affinity matrix. The global optimal solution from exhaustive search is also generated for comparison. To provide stable statistics, a thousand cases are generated randomly. For each case, a relation is constructed with a set of attributes with characteristics chosen uniformly in the range of values as specified in Table 1. A transaction access pattern matrix is then created. In addition, in each case the number of transactions is assumed to be equal or less than that of the attributes. The three access methods are assumed to be equally likely to be used by the transactions. According to the attribute access pattern of the generated transactions, the relation is then partitioned by OBP and BPi.

Parameters (Uniform distribution)	Values
Number of attributes	$5 \leq m \leq 10$
Number of transactions	$5 \leq n \leq 10$
Length of attribute j	$1 \leq \|a_j\| \leq 25$
The selectivity of transaction i	$0 \leq s_i \leq 10\%$
The relative access frequency of transaction i	$1 \leq f_i \leq 50$
The probability of an attribute accessed by a transaction	0.25

Table 1: Parameters used in the experiment.

Let us compare the performance of the proposed algorithms with that of the algorithms Nav1[6], Nav2[7], and the global optimal vertical fragmentation. The total number of disk accesses is used as the performance measure. We also compare the reduction of the

Algorithm	Avg. Reduction (%)	Max. Reduction (%)	Min. Reduction (%)	Complexity
Global optimal	24.81	84.26	0.00	$\simeq O(m^m)$
BP1	23.85	84.26	0.00	$O(n)$
BP2	24.73	84.26	0.00	$O(n^2)$
BP3	24.79	84.26	0.00	$O(n^3)$
BP4	24.79	84.26	0.00	$O(n^4)$
OBP	24.79	84.26	0.00	$O(2^n)$
Nav1	13.67	68.74	-79.04	$O(m^2)$
Nav2	8.59	68.10	-158.25	$O(m^2)$

Table 2: The average, maximum, and minimum reduction in number of disk accesses.

total number of disk accesses due to the use of the vertical partitioning algorithms.

Table 2 presents the average, maximum, and minimum reduction in terms of the total number of disk accesses by the vertical partitioning algorithms. For example, the global optimal vertical fragmentation achieves an average reduction of 24.81% in the total number of disk accesses as compared to the system without vertical fragmentation that ranges from 0 to 84.26% for the 1000 cases; while the algorithm OBP yields an average reduction of 24.79% that ranges from 0 to 84.26%. Notice that the minus sign (-) represents the cases in which the partitioning algorithms (Algorithms Nav1 and Nav2) yield more disk accesses as also reported in [2]. This is because the total number of disk accesses was not used directly in the algorithms in determining the data fragments.

Table 3 compares the results of the proposed algorithms with that of the global optimal vertical fragmentation algorithm. Deviation (%) from global optimal indicates the number of cases that deviate from the global optimum; 0% means the partitioning algorithms reach the global optimal solution. For example, algorithm BP1 can find 75.8% (758 cases) global optimal solutions out of 1000 cases, while the optimal binary partitioning algorithm yields 97.5% (975 cases). We note that the results can be improved by increasing the number of groups of reasonable cuts of BPi. However, the amount of improvement reduces as the number of groups of reasonable cuts increases. Our experimental results reveal that BPi is a very effective heuristic algorithm. For our experiment, BP4 reaches the same result as OBP.

Table 4 shows the performance improvement of vertical partitioning as compared with the system without vertical fragmentation. We note that using the global optimal vertical

Deviation (%) from	Algorithm						
global optimal	BP1	BP2	BP3	BP4	OBP	Nav1	Nav2
0	758	947	974	975	975	312	142
1	74	29	17	16	16	58	62
2	27	9	4	4	4	26	23
3	19	1	1	1	1	23	24
4	20	4	1	1	1	24	31
5	16	2	1	1	1	21	14
6	12	3	1	1	1	26	24
7	7	1	1	1	1	13	17
8	9	0	0	0	0	20	20
9	9	4	0	0	0	17	22
10	5	0	0	0	0	17	23
11	6	0	0	0	0	20	23
12	3	0	0	0	0	19	21
13	4	0	0	0	0	9	17
14	7	0	0	0	0	18	21
15	2	0	0	0	0	15	17
16	3	0	0	0	0	13	16
17	7	0	0	0	0	19	19
18	3	0	0	0	0	14	15
19	0	0	0	0	0	10	9
20	1	0	0	0	0	11	14
> 20	8	0	0	0	0	295	426

Table 3: Performance comparison of heuristic algorithms with that of the global optimal algorithm.

Reduction (%)	Algorithm							
in disk accesses	Global optimal	BP1	BP2	BP3	BP4	OBP	Nav1	Nav2
0	93	146	102	93	93	93	370	352
10	198	171	190	198	198	198	155	162
20	149	145	149	149	149	149	139	142
30	163	161	163	164	164	164	128	150
40	162	150	162	162	162	162	106	99
50	127	120	126	126	126	126	69	58
60	72	75	72	72	72	72	22	32
70	29	25	29	29	29	29	11	5
80	5	5	5	5	5	5	0	0
90	2	2	2	2	2	2	0	0
100	0	0	0	0	0	0	0	0

Table 4: Performance improvement of vertical fragmentation.

fragmentation algorithm, the costs of 907 cases can be improved, two of them can be reduced by more than 80%, and 93 out of 1000 cases cannot be improved. By using algorithms BP1, BP2, BP3, BP4, and OBP, there are 854, 898, 907, 907, and 907 cases in which the cost can be reduced, respectively, as compared to 630 and 648 cases by algorithms Nav1 and Nav2.

To study the scalability of the proposed algorithms, we increase both the maximum number of transactions and attributes from 10 to 20 and keep the remaining parameters in Table 1 the same. Because of the computation complexity, it is no longer feasible for us to find the global optimal vertical fragmentation via the exhaustive search on our computer (a 12.5 MIPs Sparc station). Therefore, we use OBP (which takes roughly four days to run for this experiment) to compare the results obtained from the heuristic algorithms.

Figure 9 portrays the average reduction of the total number of disk accesses by the vertical partitioning algorithms as a function of the number of transactions. Each point represents the average of a thousand randomly generated cases. The heuristic algorithm BPi diverges from the algorithm OBP as n increases. However, the performance of BPi is fairly close to that of the OBP as we increase the search space (i.e., increasing i) of the heuristic algorithm. Note that the result from the algorithm BP4 is comparable with that of OBP.

Figure 10 displays the probability of finding the binary optimal solutions by the heuristic algorithms. We note that most of the solutions obtained by algorithm BPi are comparable with that of the binary optimal solutions. Figure 11 shows the number of cases obtained by the algorithm BPi, for selected i, with less than 3% deviation from the solutions of the optimal binary partitioning algorithm.

5 Conclusion

A new transaction-based approach is proposed to generate vertical partitioning for relational databases. Since transactions have more semantic meaning than attributes, it allows us to optimize the vertical partitioning based on a selected set of important transactions. An optimal binary vertical partitioning algorithm OBP with computation complexity of $O(2^n)$ has been developed. When the number of attributes m is greater than the number

of the transactions n in the system, using the proposed algorithm yields a computation complexity reduction of $O(2^{m-n})$. A heuristic algorithm BPi, evaluates only up to the ith group of reasonable cuts with complexity varying from $O(n)$ to $O(2^n)$, has also been developed. Our experimental results reveal that the optimal binary partitioning algorithm yields comparable results with that of the global optimal algorithm. However, the computation complexity of OBP is much lower than that of the global optimal algorithm by exhaustive search. Further, BPi converges rather rapidly to OBP. To obtain an optimal or near-optimal vertical fragmentation, we only need to evaluate the first few groups (e.g. $i = 4$) of reasonable cuts. Therefore, the proposed heuristic algorithm is suitable for systems even with a large number of transactions.

References:

1. Cornell, D. W. and Yu, P. S., "An effective approach to vertical partitioning for physical design of relational databases", IEEE TSE, vol. 16, no. 22, February, 1990.

2. Cornell, D. W. and Yu, P. S., "A vertical partitioning Algorithm for relational databases", proc. of data engineering, 1987.

3. Hammer, M., and Niamir, B., "A Heuristic Approach to attribute partitioning", proc. of ACM SIGMOD ICOMOD 1979.

4. Hoffer, J. A., "An integer programming formulation of computer data base design problems", Information sciences 11, pp. 29-48, 1976.

5. Hoffer, J. A., and Severance, D. G., "The use of cluster analysis in physical database design", proc. first VLDB, 1975.

6. Navathe, S., Ceri, S., Wiederhold, G. and Dou, J., "Vertical Partitioning Algorithms for Database Design", ACM TODS, vol. 9, no. 4, pp. 680-710 December 1984.

7. Navathe, S., and Ra, M., "Vertical Partitioning for Database Design: A Graphical Algorithm", proc. of AMC SIGMOD ICOMOD, 1989.

8. Nijenhuis, A. and Wilf, H. S., "Combinatorial Algorithms:For Computers and Calculators", Second Edition, p.32, 1978, Academic Press.

9. Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., Price, T. G., "Access path selection in a relational database management system", proc.of ACM IOMOD, 1979.

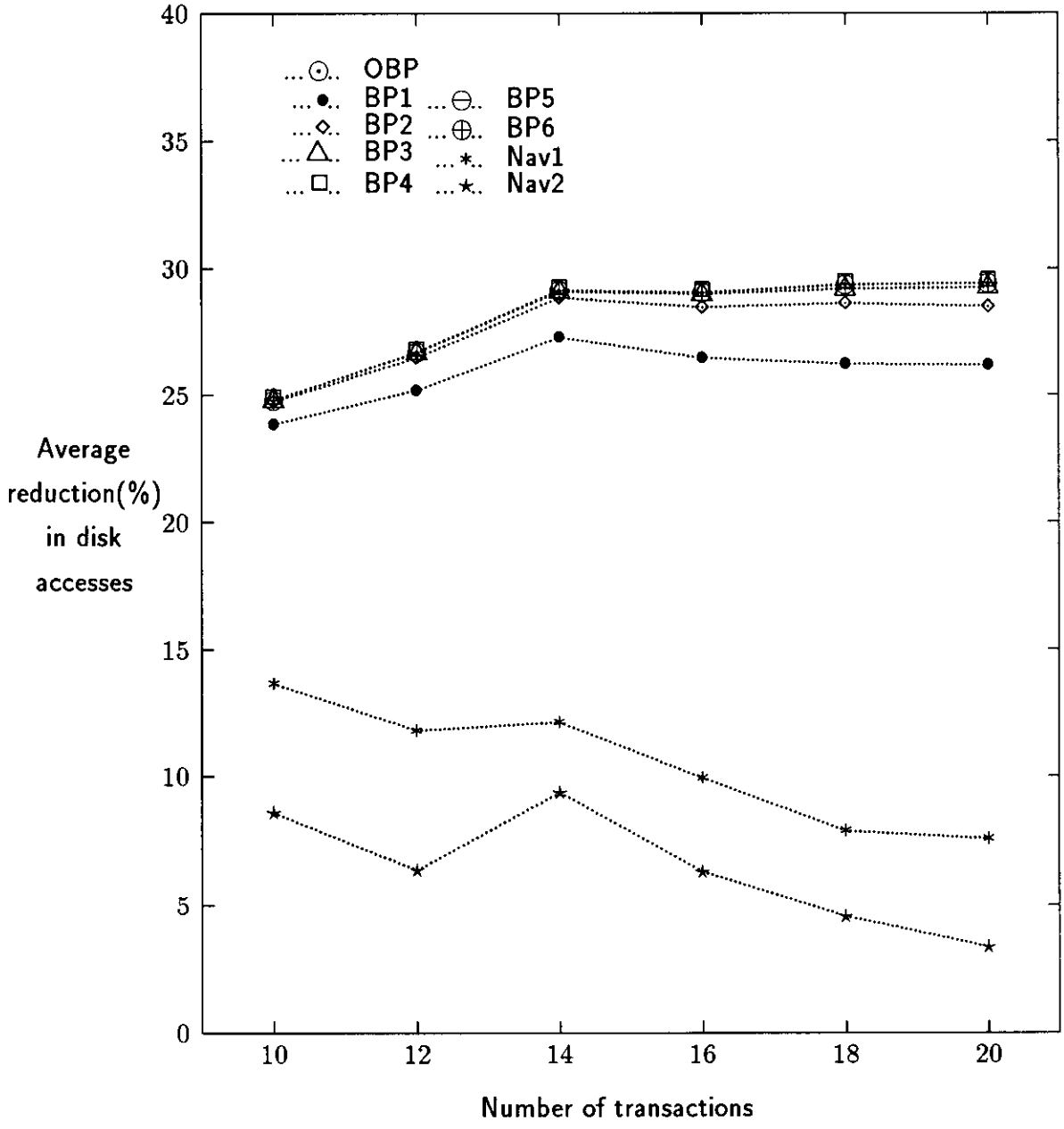


Figure 9: Performance improvement of vertical partitioning algorithms

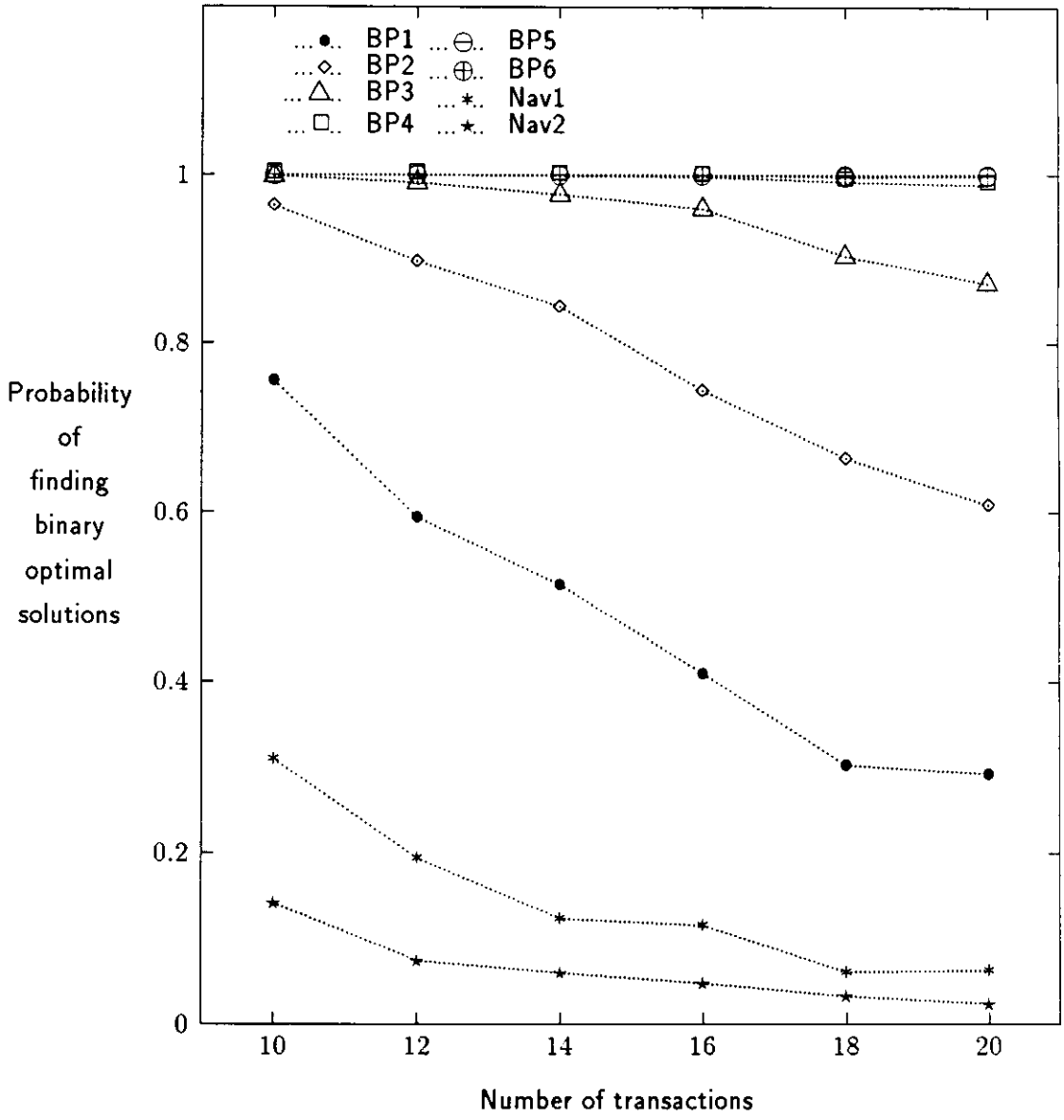


Figure 10: Probability of finding binary optimal solutions.

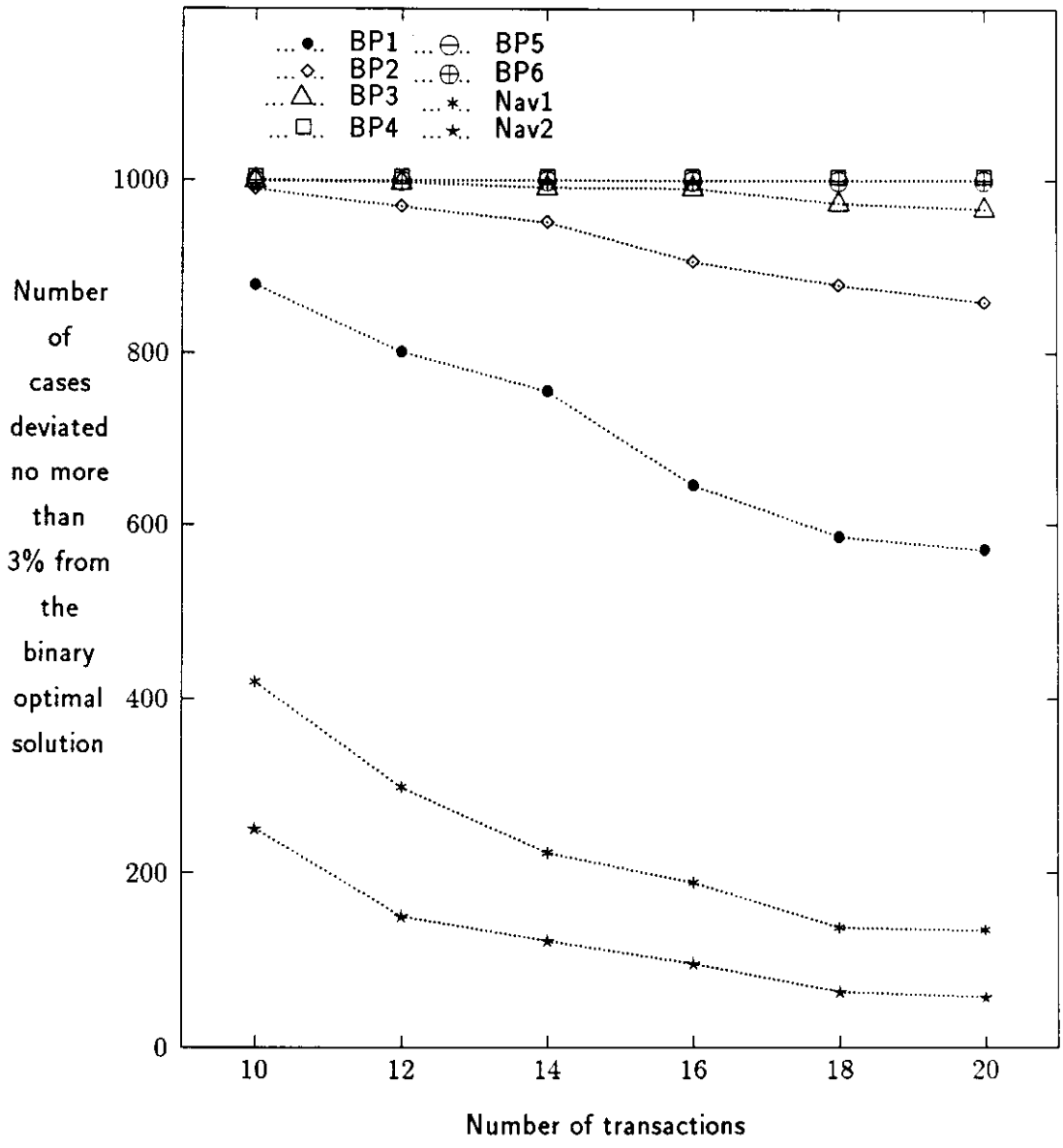


Figure 11: Number of cases deviated no more than 3% from the binary optimal solution.