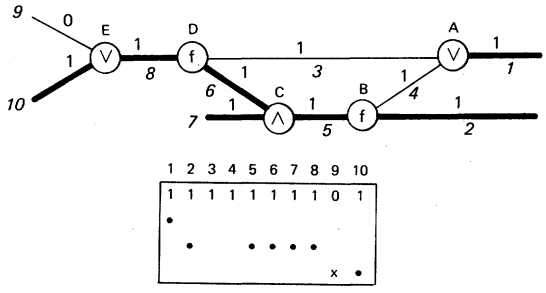


Fig. 8. A redundant graph.

output in a subcritical set. The set of input lines to this node is *maximally desensitized* if the input values are such none of the inputs are subcritical or in a subcritical set.

In Fig. 7, the input lines 4 and 5 to the AND node are maximally desensitized. Under their present value of 0, the faults 4/1 and 5/1 cannot mask any fault in the present test. Therefore, they are neither subcritical nor in a subcritical set. Hence, they are maximally desensitized.

Maximal desensitization is desirable during test generation because it reduces the number of subcritical lines and lines in a subcritical set. It contributes to the inclusion of more critical values in a test when reconvergent fan-out is encountered, as illustrated by the example of Figs. 6 and 7.

## VI. CONCLUSION

In this correspondence, we have given definitions for the equivalence, dominance, and masking relations among faults in a graph. We have also introduced a criticality measure which can represent these relations. We have shown how these fault properties should be identified and appropriate actions taken in generating test sets for combinational networks.

The criticality measure is a helpful notation to use in a test generating algorithm. It is different from that of the $D$-algorithm [7]. The notations of $D$ and $\bar{D}$ in the $D$-algorithm identify a sensitized path. The notations of critical 1 and ciritical 0 also identify a sensitized path, and, in addition, they prepresent faults that can be detected in the network. For example, in Fig. 8, the $D$-algorithm would place a $D$ in all the lines with critical 1's and also in lines 3 and 4 because they are in the two-dimensional sensitized path from line 10 to line 1 although there are no detectable faults on these lines.

One big advantage of using the notation of critical 1 and critical 0 is that, after a test is generated, the critical values readily exhibit the faults detected by this test. The author has already developed an algorithm to generate test sets for combinational networks using the criticality measure [9], [10].

## ACKNOWLEDGMENT

The author is grateful to an anonymous referee for suggesting a simpler proof of Theorem 6.

## REFERENCES

[1] E. J. McCluskey and F. W. Clegg, "Fault equivalence in combinational logic networks," *IEEE Trans. Comput.*, vol. C-20, pp. 1286–1293, Nov. 1971.
[2] J. F. Poage and E. J. McCluskey, "Derivation of optimum test sequences for sequential machines," in *Proc. 5th Ann. Symp. Switching Theory and Logical Design*, 1964, pp. 121–132.
[3] K. C. Y. Mei, "Fault dominance in combinational circuits," Digital Syst. Lab., Stanford Electron. Lab., Stanford Univ., Stanford, Calif., Tech. Notes 2, Aug. 1970.
[4] A. D. Friedman, "Fault detection in redundant circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 99–100, Jan. 1967.
[5] J. P. Hayes, "A NAND model for fault diagnosis in combinational logic networks," *IEEE Trans. Comput.*, vol. C-20, pp. 1496–1506, Dec. 1971.
[6] J. W. Gault, J. P. Robinson, and S. M. Reddy, "Multiple fault detection in combinational networks," *IEEE Trans. Comput.*, vol. C-21, pp. 31–36, Jan. 1972.
[7] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 567–579, May 1967.
[8] D. R. Schertz, "On the representation of faults," Coordinated Sci. Lab., Univ. Illinois, Urbana, Ill., Rep. R-418, May 1969.
[9] D. T. Wang, "An algorithm for the generation of test sets for combinational logic networks," Ph.D. dissertation, Stanford Univ., Dep. Elec. Eng., Oct. 1973; also IBM Development Lab., Endicott, N. Y., IBM Tech. Rep. TR01.1727, Dec. 1973.
[10] ——, "An algorithm for the generation of test sets for combinational logic networks," this issue, pp. 742–746.

# A Branch and Bound Algorithm for Computing $k$-Nearest Neighbors

KEINOSUKE FUKUNAGA AND
PATRENAHALLI M. NARENDRA

***Abstract*—Computation of the $k$-nearest neighbors generally requires a large number of expensive distance computations. The method of branch and bound is implemented in the present algorithm to facilitate rapid calculation of the $k$-nearest neighbors, by eliminating the necesssity of calculating many distances. Experimental results demonstrate the efficiency of the algorithm. Typically, an average of only 61 distance computations were made to find the nearest neighbor of a test sample among 1000 design samples.**

***Index Terms*—Branch and bound, distance computation, hierarchical decomposition, $k$-nearest neighbors, tree-search algorithm.**

## I. INTRODUCTION

The $k$-nearest neighbor approach has been well investigated in the literature, and shown to be a powerful nonparametric technique for density estimation [1] and classification [2]. However, finding the $k$-nearest neighbors of a test sample among $N$ design samples is a time-consuming process, particularly for large $N$.

Several forms of preprocessing have been proposed to minimize the number of distance computations. For purposes of classification, Hart [3] suggested condensing the design set while retaining most of the samples yielding discriminatory information between classes. Fischer and Patrick [4] proposed reordering the design samples in such a way that many distance computations could be eliminated. However, the reordering of the design samples requires $N(N-1)/2$ pairwise distance computations, which is quite expensive for large $N$.

This correspondence is concerned with an efficient procedure to compute the $k$-nearest neighbors. The basic approach is first to hierarchically decompose the design samples into disjoint subsets, and then to apply to the resultant groups, the branch and bound method [5]–[7] which is a powerful tree-search algorithm. The results of several experiments on various data demonstrate considerable computational savings.

## II. THE SEARCH ALGORITHM

Let $\{X_1,\cdots,X_N\}$ be $N$, $n$-dimensional design samples. It is required to compute the $k$-nearest neighbors of a test sample $X$ among $\{X_1,\cdots,X_N\}$, as measured by an appropriate distance function $d(\cdot,\cdot)$.

For the sake of simplicity, let us consider the case for $k = 1$. The extension of the algorithm for $k > 1$ is straightforward as will be shown later.

The proposed procedure consists of two stages. First, the design set is hierarchically decomposed into disjoint subsets. The results of this decomposition are represented by a tree structure. Second, the resulting tree is searched by the branch and bound method.

Manuscript received March 22, 1974; revised August 15, 1974. This work was supported in part by the National Science Foundation nuder Grant GJ-35722.
The authors are with the School of Electrical Engineering, Purdue University, Lafayette, Ind. 47907.

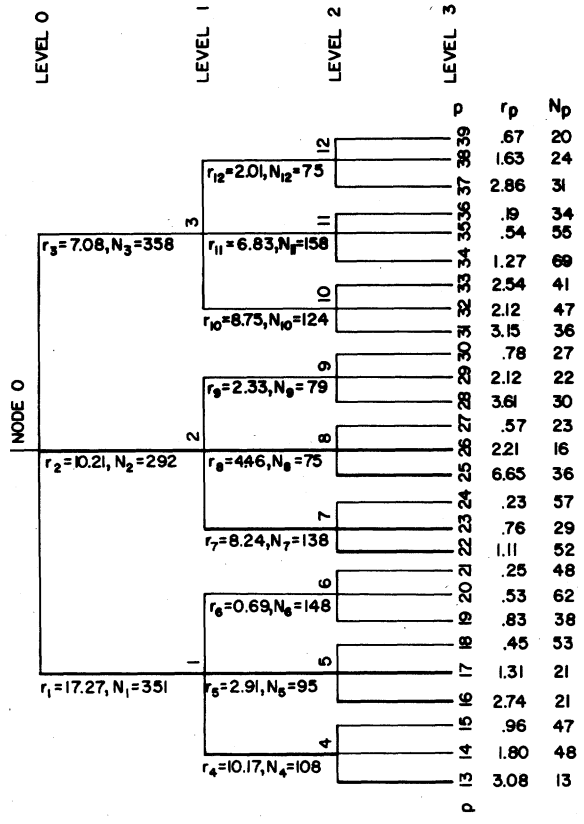


Fig. 1. Results of the hierarchical decomposition (bold lines indicate the nodes expanded by the algorithm for the typical test sample of Experiment 1).

*Decomposition of the Design Samples*

The design sample set is divided into $l$ subsets, each subset is further divided into $l$ subsets, and so on. The results of such a decomposition can be represented by a tree structure as in Fig. 1 ($l = 3$). Each node $p$ of the tree represents a group of samples, and is characterized by the following parameters:

$S_p$ set of samples associated with node $p$;
$N_p$ number of samples associated with node $p$;
$M_p$ sample mean of $S_p$; (1)
$r_p = \max_{X_i \in S_p} d(X_i,M_p)$ farthest distance from $M_p$ to an $X_i \in S_p$

Any clustering technique can be used for decomposing the samples. Obviously, the resulting groups are not required to be "meaningful" clusters, nor is the "cluster splitting tendency" of some clustering techniques, objectionable. Computational economy is the chief consideration affecting the choice of the clustering procedure. In the experiments reported in this correspondence, we used the $k$-means clustering algorithm [8] which forms the basis for ISODATA [9]. As a general guideline, nodes with large populations $N_p$ may be permitted to have large $r_p$.

*Tree Search by Branch and Bound*

After the design samples have been decomposed, and the quantities $M_p$, $r_p$, $N_p$, and $S_p$ evaluated, each node $p$ can be tested as to whether or not the nearest neighbor to $X$ may be in $S_p$, by the application of the following rule.

*Rule 1:* No $X_i \in S_p$ can be nearest neighbor to $X$, if

$$B + r_p < d(X,M_p). \tag{2}$$

$B$ is the distance to the current nearest neighbor of $X$ among the design samples considered up to the present. Initially, $B$ is set to be $\infty$.

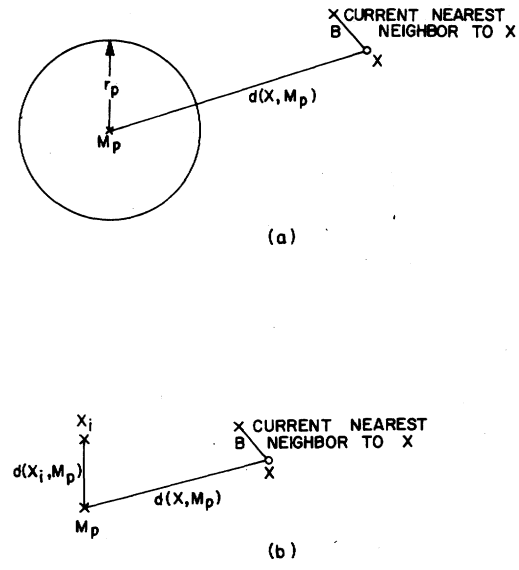


Fig. 2. Illustrating Rules 1 and 2.

The proof of Rule 1 follows.

For an $X_i \in S_p$,

$$d(X,X_i) + d(X_i,M_p) \geq d(X,M_p) \quad \text{(Triangle inequality)}.$$

Since, $d(X_i,M_p) \leq r_p$ by definition,

$$d(X,X_i) \geq d(X,M_p) - r_p.$$

Therefore, $X_i$ cannot be nearest neighbor to $X$, if,

$$d(X,X_i) \geq d(X,M_p) - r_p > B.$$

Equation (2) follows immediately. Fig. 2(a) illustrates the proof for the Euclidean distance measure.

Thus, many nodes $p$ and the corresponding groups of samples $S_p$ could be eliminated from consideration without explicitly computing the distances to the individual samples in $S_p$.

For a node $p$ at the final level (level 3 in Fig. 1) of the tree, if Rule 1 is not satisfied, distances to the individual samples in $S_p$, from $X$ must be computed. However, many distance computations can still be avoided as follows.

*Rule 2:* $X_i$ cannot be the nearest neighbor to $X$, if

$$B + d(X_i,M_p) < d(X,M_p), \tag{3}$$

where $Xi \in S_p$.

The proof is along the same lines as for Rule 1. Fig. 2(b) illustrates the proof. The $d(X_i,M_p)$ are available from the computation of $r_p$ during the decomposition. These distances are stored only for the nodes at the *lowest level* of the tree, requiring the storage of only $N$ real numbers.

We can now apply the branch and bound method which is well known as an efficient tree-search algorithm, in order to search the tree in Fig. 1, testing the nodes by Rules 1 and 2. The tree-search algorithm is as follows.

*Step 0 (Initialization):* Set $B = \infty$, CURRENT LEVEL $L = 1$, and CURRENT NODE $= 0$.

*Step 1 (Expansion of* CURRENT NODE*):* Place all nodes that are immediately direct successors of the CURRENT NODE into the ACTIVE LIST at CURRENT LEVEL. Compute and store the $d(X,M_p)$ for these nodes.

*Step 2 (Test for Rule 1):* For each node $p$ in the ACTIVE LIST at CURRENT LEVEL, if $d(X,M_p) > B + r_p$, remove $p$ from the ACTIVE LIST at CURRENT LEVEL.

*Step 3 (Backtracking):* If there are no nodes left in the ACTIVE LIST at CURRENT LEVEL, backtrack to the previous level, i.e., set $L = L - 1$. If $L = 0$, then terminate the algorithm. If $L \neq 0$, go to Step 2. If there are one or more nodes in ACTIVE LIST at CURRENT LEVEL, go to Step 4.

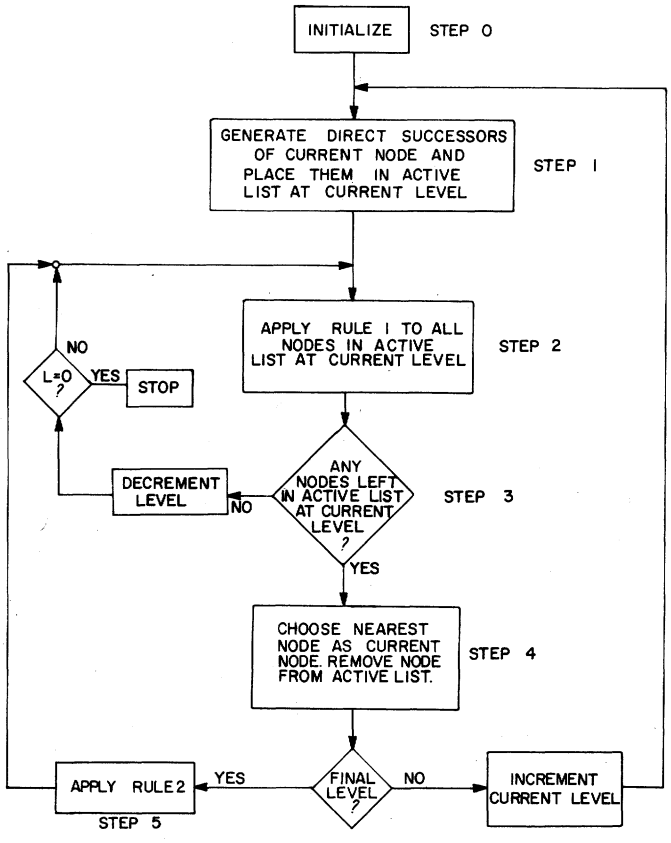


Fig. 3. Flowchart for the search algorithm.

*Step 4 (Choose the Nearest Node for Expansion):* Choose the nearest node $p$ (yielding the smallest $d(X,M_p)$), among the nodes in ACTIVE LIST at CURRENT LEVEL, and call it the CURRENT NODE. Remove $p$ from the ACTIVE LIST at CURRENT LEVEL. If the CURRENT LEVEL is the final level, go the Step 5. Otherwise, set $L = L + 1$, and go to Step 1.

*Step 5 (Test for Rule 2):* For each $X_i$ in CURRENT NODE $p$, do the following. If $d(X,M_p) > d(X_i,M_p) + B$, $X_i$ cannot be the nearest neighbor to $X$, so, do not compute $d(X,X_i)$. Otherwise, compute $d(X,X_i)$. If $d(X,X_i) < B$, set CURRENT NN $= i$, and $B = d(X,X_i)$. After all the $X_i$'s in the CURRENT NODE are tested, go to Step 2.

Fig. 3 shows the flow chart for the algorithm. Upon the termination of the algorithm, the nearest neighbor is given by CURRENT NN and its distance to $X$, by $B$.

Note that a slight improvement may be made in Step 1 by setting $B = \min [B,d(X,M_p) + r_p]$, whenever $d(X,M_p)$ is computed. This will tighten $B$ in the initial stages of the algorithm.

### *Extension to k-Nearest Neighbors*

Extension to $k$-nearest neighbors is straightforward, with $B$ as the distance to the current $k$th nearest neighbor. When a distance is actually calculated in Step 5, it is compared against the distances from $X$ to its current $k$-nearest neighbors, and the current $k$-nearest neighbor table is updated as necessary, discarding the sample from the table that is now $k + 1$th nearest neighbor to $X$.

## III. EXPERIMENTAL RESULTS

*Experiment 1—Gaussian Distribution:* 1000 pseudorandom samples were drawn from a bivariate normal population with mean vector $M = [0 \quad 0]^T$, and covariance matrix

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The samples were divided into 27 final subgroups by successive applications of the three-means algorithm as shown in Fig. 1. It is to be noted that at each of the three levels of splitting, only $N \times$ (average number of iterations of the clustering algorithm) distance computations are required. The average number of iterations in the present example were four. Thus, a total of $4 \times 3 \times 1000$ distance computations were required for the preprocessing. An independent test set of 1000 samples was drawn from the same population, and the search algorithm was applied for each of the test samples to find its nearest neighbor among the preprocessed set of samples. A total of 61 distance computations were required per sample on the average, and no test sample required more than 87 distance computations. As an indication of the effectiveness of the branch and bound method, the algorithm rejected 1 whole group at level 1, an average of 3 groups at level 2, and 11 groups at level 3. Indeed, at the final level of the tree, only 6 of the 27 groups were considered on the average, with only 39 actual distance computations being made on the average at Step 5 of the algorithm. The remaining 22 distance computations were made during the search process to evaluate the distances to the sample means of the groups. As an illustration, for a typical test sample, the following sequence of nodes were selected for expansion as Current Node in the algorithm, (with reference to Fig. 1): 0-1-5-16-17-4-13-2-7-23-22-8-25-26. Nodes not listed in the above sequence were rejected by Rule 1. Note that if a node is rejected by Rule 1, *all* of its successors are implicitly rejected, resulting in considerable savings in distance computations to the successive nodes.

*Experiment 2—Uniform Distribution:* The same experiment was repeated with 1000 pseudorandom samples from a bivariate uniform distribution over $(-0.5 \leq x_1 \leq 0.5)$ and $(-0.5 \leq x_2 \leq 0.5)$. The average number of distance computations per test sample were 46. The average number of iterations of the clustering procedure in the preprocessing were 2 which meant that $2 \times 3 \times 1000$ distance computations were made during the preprocessing.

*Experiment 3—(Eight-Dimensional Data):* To test the behavior of the algorithm in higher dimensional space, it was applied to 3000 pseudorandom samples uniformly distributed within a hypercube of side 1 in eight dimensions. $l$ was chosen to be 4, and the number of levels of dichotomy were also 4, yielding 256 final groups. The average number of distance computations made to find the nearest neighbor of an independent test set from the same population were 451. The number of groups rejected by the algorithm at levels 1 through 4 were, respectively, $-0$, 0.6, 20, and 130.

### *General Remarks*

The sensitivity of the algorithm to the number of levels and to the number of branches at each node was investigated for the data of Experiment 1. It was found that the dominant factors affecting the efficiency of the algorithm are the average population of the final subgroups, and the total number of nodes in the tree. Too many samples (more than 60) in each of the final subgroups yielded fewer nodes, but required more distance computations to the individual samples. Too few samples in each of the final subgroups yielded a large number of nodes. Thus, while much fewer distance computations were made to the individual samples themselves (at Step 5), a larger number of distance computations were made to the nodes in the search process. It appears that a compromise between the size of the terminal subgroups and the number of nodes in the tree has to be made. In the present instance, the cited example in Experiment 1 required the fewest distance computations on the average (with 39 nodes in the tree and 27 final subgroups). The algorithm requires proportionately far fewer distance computations when the number of samples are larger. This renders it even more valuable when large data are being processed.

The preprocessing is so inexpensive in terms of computation time that the algorithm remains viable even when the test set is small in comparison with the design set. For the same reason, the algorithm can still be applied with advantage, when the test set and the

design set are the same. The total number of distance computations, including the preprocessing, would still be a small fraction of the $N(N-1)/2$ pairwise distance computations required otherwise.

## ACKNOWLEDGMENT

The authors would like to express their appreciation to Dr. W. L. G. Koontz, Bell Laboratories, Whippany, N.J., for his suggestions.

## REFERENCES

[1] D. O. Loftsgaarden and C. P. Quesenberry, "A nonparametric density function," *Ann. Math. Stat.*, vol. 36, pp. 1049–1051, 1965.
[2] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 21–27, Jan. 1967.
[3] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 515–516, May 1968.
[4] F. P. Fischer and E. A. Patrick, "A preprocessing algorithm for nearest neighbor decision rules," in *Proc. Nat. Electronics Conf.*, vol. 26, Dec. 1970, pp. 481–485.
[5] S. W. Golomb and L. D. Baumert, "Backtrack programming," *J. Ass. Comput. Mach.*, vol. 12, pp. 516–524, Jan. 1965.
[6] E. L. Lawler and D. E. Wood, "Branch-and-bound methods, A survey," *Oper. Res.*, vol. 149, no. 4, 1966.
[7] N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence*. New York: McGraw-Hill, 1971, ch. 3.
[8] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. on Probability and Statist.*, 1967, pp. 281–297.
[9] G. H. Ball and D. J. Hall, "ISODATA, a novel method of data analysis and pattern classification," Stanford Res. Inst., Stanford, Calif., Tech. Rep., 1965.

# Minimization of Switching Functions—A Fast Technique

SURESHCHANDER

***Abstract*—An algorithm is presented in this correspondence which will generate only those prime implicants (PI's) which are essential or necessary for a minimal sum of products (the technique can be extended for product of sums form also) of a given switching function—given all the minterms. This algorithm eliminates the necessity of finding out all the PI's.**

**The algorithm is suitable both for solving a problem by hand or for programming it on a digital computer. The memory requirement in this algorithm is much less than any existing known method.**

***Index Terms*—Consensus, DON'T CARE term, literal, minimal sum, minterm, prime implicant, product-1, product-0, subcube.**

## I. INTRODUCTION

The minimization of switching functions with a two-level network is a complex problem, particularly when the number of variables is large. The Karnaugh map method is an effective tool up to a maximum of six variables. The Quine–McCluskey technique decomposes the problem into two parts: 1) the determination of prime implicants (PI's); and 2) the selection of PI's for a minimal cover. In the past, a lot of attention had been paid to the first part [3]–[9], namely the generation of all the PI's. Few techniques, like Pyne and McCluskey [13] and weighted method [14], exist for finding a minimal cover, but these are unsuitable when the number of variables is large.

The second part of the problem is of grater complexity. The deduction of minimal covering from the knowledge of all the PI's is practically impossible for functions having more than ten variables [9].

Recently Sureshchander [12] gave a method which deals with the problem as a whole instead of decomposing it into two parts. This paper is an extension of the earlier paper [12]. This technique has been written and programmed for an ICL 1909 computer using Fortran IV. The core required for this program is about $4k$ plus twice the number of minterms. This program can be used to minimize a function up to 23 variables, as ICL 1909 has words of 24-bit length—the 24th bit being the sign bit. The program can be used, after small modifications, even if there are more than 23 variables.

Manuscript received March 27, 1972; revised January 25, 1974.
The author is with the Department of Electrical Engineering, Pant College of Technology, Pantnager, India.

## II. DEFINITIONS AND THEOREMS

*Definition 1.1:* A *product -1* $\boldsymbol{P}$ is defined as a product of only those variables which are true in a given minterm.

*Definition 1.2:* A *product-0* $\boldsymbol{p}$ is defined as a product of only those variables which are not true in a given minterm.

*Definition 1.3:* The *product-1* and *product-0* of a given minterm will be said to be the *corresponding-product* terms.

$$\Delta \boldsymbol{P}_i = \boldsymbol{p}_i$$

and

$$\Delta(\Delta \boldsymbol{P}_i) = \boldsymbol{P}_i.$$

*Example:* The *product-1* $\boldsymbol{P}$ and *product-0* $\boldsymbol{p}$ terms of a minterm $x_1x_2x_3\bar{x}_4\bar{x}_5\bar{x}_6$ are $x_1x_2x_3$ and $\bar{x}_4\bar{x}_5\bar{x}_6$, respectively.

$$\Delta(x_1x_2x_3) = \bar{x}_4\bar{x}_5\bar{x}_6$$

and

$$\Delta(\Delta(x_1x_2x_3)) = (x_1x_2x_3).$$

The product of $\boldsymbol{P}_i$ and $\boldsymbol{p}_i$ completely defines any minterm.

*Definition 2:* Two *product-1* terms, $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$, will said to be $d$-distance apart if $\#(\boldsymbol{P}_i, \boldsymbol{P}_j) = d$, $\boldsymbol{P}_i, \boldsymbol{P}_j$ denotes the number of literals which are either in $\boldsymbol{P}_i$ or $\boldsymbol{P}_j$ but not in both.

*Example:* The *product-1* terms $x_1x_3x_4x_5$ and $x_1x_2x_3$ are at a distance of three.

*Definition 3:* $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ will be said to have consensus if they are one distance apart, i.e., $\#(\boldsymbol{P}_i,\boldsymbol{P}_j) = 1$.

*Definition 4:* If a *product-1* term $\boldsymbol{P}_i$ is at a distance one from exactly $r$ number of *product-1* terms, then $\boldsymbol{P}_i$ is said to have $r$th-degree consensus.

*Definition 5:* Let $\boldsymbol{P}_0$ have $r$th-degree consensus and $\boldsymbol{P}_{0j_i}$ and $\boldsymbol{P}_{0(j-1)_k}$ be at a distance $j$ and $(j-1)$, respectively, from $\boldsymbol{P}_0$ for $j = 2,3,\cdots,r$. If each $\boldsymbol{P}_{0j_i}$ has consensus with exactly[1] $j$ number of $\boldsymbol{P}_{0(j-1)}$ terms, then all these $\boldsymbol{P}_{0j_i}$ terms will be said to belong to the $\boldsymbol{P}_0$ family. $j$ takes the value from 2 to $r$.

*Definition 6:* $\boldsymbol{P}_0$ will be said to have a proper $r$th-degree consensus provided: 1) $\boldsymbol{P}_0$ has $r$th-degree consensus; and 2) There are $\binom{r}{j}$ number of $\boldsymbol{P}_{0j_i}$ terms in the $\boldsymbol{P}_0$ family. Here, $i = 1,2,\cdots,\binom{r}{j}$, and $j = 1,2,\cdots,r$.

*Example:* Consider, for example, the following switching function:

$$f = \sum 5,8,9,10,11,12,13,14,15.$$

The corresponding Karnaugh map representation is shown in Fig. 1. The $\boldsymbol{P}$ terms for the above function are

$$f = \sum (x_2x_4, x_1,x_1x_4,x_1x_3,x_1x_2x_4,x_1x_2,x_1x_2x_4,x_1x_2x_3,x_1x_2x_3x_4).$$

The $\boldsymbol{P}$ term $x_1x_2$ has third-degree consensus namely with

$$x_1,\ x_1x_2x_4,\ \text{and}\ x_1x_2x_3,$$

and has all $\binom{r}{j}$ terms in the $x_1x_2$ family. The $\binom{3}{1}$ terms in this case are $x_1$, $x_1x_2x_4$, and $x_1x_2x_3$. The $\binom{3}{2}$ terms are $x_1x_4$, $x_1x_3$, and $x_1x_2x_3x_4$. The $\binom{3}{3}$ term is $x_1x_3x_4$. These terms meet all the requirements of Definitions 5 and 6. Hence, $x_1x_2$ has a proper consensus of third degree.

*Theorem 1:* If $\boldsymbol{P}_0$ has a proper consensus of $r$th degree, then the entire $\boldsymbol{P}_0$ family will constitute an essential prime implicant (EPI). This family corresponds to an $r$th-order subcube on the Karnaugh map.

*Proof:* For a PI to be an EPI, it is necessary that at least one

[1] $P_{0ij}$ may have a concensus of degree higher than $j$.