BOOLEAN MINIMIZATION AND THE VICINITY METHOD

Norman M. Martin

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-86-03 January 1986

ABSTRACT

Boolean minimization remains a useful tool in the design of digital hardware. In minimization theory, an interesting concept is that of the vicinity, (i.e., the least n-cube covering a given minterm but none of its adjacent zeroes). Its special properties yield an extremely convenient and flexible method for minimizing sum-product and related Boolean forms. The paper presents a summary of standard minimization theory and an explanation of this method. Among the interesting features of this method are the direct calculation of essential prime implicants and the generation of the remaining terms of minimal cost expressions in order of increasing complexity. This avoids much of the calculation involved in the Quine-McCluskey and related minimization methods.

INTRODUCTION

In designing digital hardware, issues of efficiency occur in several ways. In the first place, the completion of a particular task can frequently be accomplished by one of several algorithms, not necessarily strictly equivalent. (Where the algorithms are all correct for the purpose involved, there is usually, presumably always, a structure under which they are all equivalent up to an appropriate isomorphism; this does not however imply that they are equivalent for other purposes). Since the algorithm can affect the efficiency and cost, it affects the optimality of the final design. Even when the algorithm in the mathematical abstract sense does not vary, the digital device can be organized in more than one way, resulting in a choice of detailed algorithms. Finally with a final detailed algorithm, more than one combination of gates will realize that algorithm. (Actually this always is an infinite number although in any actual case only a finite number of them are simple enough to be practically attractive). In a classic paper in 1937, Claude Shannon pointed out the correlation between switching circuits and Boolean expressions. A corollary of his work is that this last problem is equivalent to that of choosing the minimal cost expression among a set of equivalent Boolean formulae, relative to specified cost criteria. For an important class of cost functions, the monotonic, this problem was first solved by W.V.O. Quine in 1955. The method described in this paper is an improvement of the methods derived from Quine's work which are commonly known as the Quine-McCluskey method

. It compares favorably (how favorably is an open question) with Quine-McCluskey and other well-known methods and unlike

other methods developed because of convergence properties superior to the Quine-McCluskey method, is very easy to learn and use.

It should be emphasized that, like any combinational minimization method, only the third problem referred to above is dealt with. Since, more often than not, improvements from algorithmic variation or from the architecture, i.e. from considerations of the first and second types, result in more radical design improvements than combinatorial minimization, it is important that we avoid the conceptual trap of identifying this part of the optimality problem with the whole problem.

COST FUNCTIONS

In order to evaluate characteristics which we wish to optimize, it is convenient to express them in mathematical form. To do so we use the concept of cost function, that is a function from the set of Boolean terms into the positive integers. Over a very wide variety of cases, we can then express the optimization problem in which we are interested as the problem of obtaining a minimal cost expression relative to a particular cost function. In this connection there are a number of classes of cost functions of special interest, either because of their frequent occurrence in practical applications or because of results obtainable concerning them.

One such class are the symmetric cost functions, that is, those satisfying:

1. $C(A \cdot B) = C(B \cdot A)$, for all term A and B
2. $C(A+B) = C(B+A)$, for all terms A and B
3. If A, B, D and E are terms, D is like E except for having an occurrence of A where D has an occurrence of B, and $C(A) = C(B)$, then $C(D) = C(E)$. Virtually all cost functions which are of practical importance are symmetric and, in the remainder, assume the cost functions to be symmetric except where we explicitly say otherwise.

Another significant class is called the strictly monotonic cost functions. These are the cost functions satisfying: For all terms A, B, and D:

1. $C(A) < C(A \cdot B)$
2. $C(B) < C(A \cdot B)$
3. $C(A) < C(A+B)$
4. $C(B) < C(A+B)$

5. $C(A) < C(B) \Rightarrow C(A \cdot D) < C(B \cdot D)$

6. $C(A) < C(B) \Rightarrow C(D \cdot A) < C(D \cdot B)$

7. $C(A) < C(B) \Rightarrow C(A+D) < C(B+D)$

8. $C(A) < C(B) \Rightarrow C(D+A) < C(D+B)$

If, in the above conditions, we substitute "$\leq$" for '<', the cost function is called weakly monotonic.

While monotonic cost functions are not quite as universal among the practically interesting ones as symmetric cost functions are, they are nevertheless of wide practical import. Among the strictly monotonic cost functions (either simply or with very mild assumptions) are:

1. Number of literal inputs
2. Diode cost (number of diodes or transistors in a combinational circuit)
3. Actual cost of components (in dollars or the like)
4. Minimum required weight
5. Minimum required volume (usually weakly monotonic)
6. Required power (usually weakly monotonic)
7. Number of required integrated circuits (normally weakly monotonic)

In addition the strictly monotonic functions are closed under weighted average with positive weights and the entire class (both strict and weak) is closed under weighted average with non-negative weights. This fact when combined with the number and variety of monotonic cost functions is important where what is to be optimized is a mixture of criteria, provided all are monotonic.

Of course, not all useful optimality criteria yield monotonic cost functions. For instance, under most circumstances, neither average nor maximum speed is monotonic and as a result where speed functions as an

optimality, rather than a boundary, condition, the kind of minimization we are discussing is of decreased or no importance. Where, as occurs quite frequently, optimality depends on a mixture of criteria some connected with monotonic costs and some not, the importance of minimization acquires a rather mixed character. In addition, it should be noted that the one-many relation between mathematical and switching theoretical functions creates the situation that even when the optimality criteria all give rise to monotonic cost functions, issues of organizational and architectural arrangement frequently give rise to simplifications much more drastic than can be achieved by the minimization theory here discussed.

A SUMMARY OF CONVENTIONAL MINIMIZATION THEORY

In what follows we will assume our cost function is symmetric in our main discussion. An interesting question, useful both in itself and because of its connection to other problems, is the question of the minimum cost sum of products expression that expresses a given Boulean function.

Let us first introduce a few definitions. A Boolean expression A (or the function it expresses) is called an implicant of another B provided A+B=B is a Boolean identity (i.e. a Theorem of Boolean algebra).

An expression is called a literal if it is a variable or the complement of a variable. A minterm of a set of variables α is a product of literals of the elements of α such that no variable occurs twice (i.e. each variable occurs once, either complemented or not). A minterm is a 1-minterm of a function if it is a minterm of the variables of that function and is an implicant of it. A product implicant is an implicant which is a product of literals in which no variable occurs twice. Finally a prime implicant of a function A is a product implicant P of A such that if P is an implicant of P' and P is a product implicant of A, then P=P'. This is equivalent to saying that any way of shortening P by eliminating one of the literals will not be an implicant.

It is known that if the cost function is strictly monotonic, every minimum cost sum of products expression is a sum of prime implicants and if the cost function is weakly monotonic, at least one minimum cost expression is a sum of prime implicants.

A prime implicant P of B is called essential if every sum of prime implicants of B which is equivalent to B contains P. The sum of all the essential prime implicants of B is called the core of B. A non-essential

prime implicant of B which is an implicant of the core of B is called redundant. All other prime implicants are called optional. It follows easily that if the cost function is monotonic, every essential prime implicant and no redundant prime implicants of B will be in the minimum cost sum of products expression.

Let us call a product of literals a standard product if no variable occurs more than once. We will call two standard products adjacent if there is exactly one variable which occurs uncomplemented in one and complemented in the other.

With the help of these concepts and results, we can outline a strategy for generating a minimal cost expression. The strategy in most common use is to determine all of the prime implicants and then use the fact that for every essential prime implicant E there is at least one 1-minterm M such that E is the only prime implicant which covers M to determine which prime implicants are essential, redundant, and optional. Then if we term any 1-minterm not an implicant of any essential prime implicant a remaining minterm, the minimum cost sum-of-products expression is the sum of all of the essential prime implicants plus (if there are remaining minterms) the least cost way of covering the remaining minterms with optional prime implicants. (For weakly monotonic cost functions, the same result holds for at least one minimal cost expression.) In principle, only the last step requires the laborious move of looking at all possibilities and even here when the cost function is not too complex, one can often use ad hoc considerations to decrease the number of cases to be considered.

For a method of generating all of the prime implicants we could for example use the fact that a product implicant P is prime if and only if

there is no product implicant P' which has the same variables and is adjacent to P. Thus if we start with the list of minterms, repeatedly use the lemma mentioned in all possible ways, adding shorter product implicants each time a non-prime pair of the kind mentioned in the lemma is detected, the process will terminate with a complete list of prime implicants. The resulting algorithm is what is usually called the Quine-McCloskey method.

THE VICINITY METHOD

There is however an alternative method, generally more convenient and faster, which we will call the vicinity method. To define it we will need a few additional definitions.

We will call the function which takes the opposite value of A (in a two-valued algebra) the complement of A. We will call a 1-minterm of A's complement a <u>0-minterm</u> of A. We will call a literal an <u>adjacency literal</u> of a 1-minterm M provided it appears in M but is absent in some 0-minterm adjacent to M. We will call V the <u>vicinity expression</u> (or simply the vicinity) of M if it is the product of all M's adjacency literals (or 1 if M has none).

It can then be proved that every product implicant expression (and hence, any prime implicant expression) of which M is an implicant contains every adjacency literal of M and hence the vicinity of M. (Note that in the Karnaugh map representation this means that every prime implicant that covers M is covered by the vicinity).

A consequence of this theorem is that a product implicant is an essential prime implicant if and only if it is the vicinity of one of its 1-minterms. This result allows the determination of all essential prime implicants by determining the vicinities of the 1-minterms

(skipping those already covered by essential prime implicants already determined). Note this allows us to generate all the essential prime implicants without first generating all prime implicants. When the function is not simply equivalent to the sum, we can if we wish generate the prime implicants which cover a given remaining minterm M by adding the subsets of the set of literals in M but not in M's vicinity in order of increasing size.

In addition, if M_1, ..., M_n are 1-minterms, let us call the joint vicinity of M_1, ..., M_n the product of vicinities of M_1, ..., M_n and the conjoin of M_1, ..., M_n the product of all literals in all of M_1, ..., M_n. It then follows that there is a prime implicant P which covers all of M_1, ..., M_n exactly if every literal in the joint vicinity is in the conjoin and the conjoin is an implicant.

It follows from this that if P is a minimal cost prime implicant covering both of the conjoin of M_1, ..., M_n and of one of the M_i's, it is in one of the minimal cost sum of products expressions. Let us define the compatability set of a remaining minterm M the set of all remaining minterms M' such that M is an implicants of the vicinity of M' and M' is an implicant of the vicinity of M.

Putting these considerations together, we have the following algorithm:

1. Chose a minterm not yet covered or checked and determine its vicinity.
2. Test whether the vicinity covers any 0-minterms; if it does not, vicinity is essential: List vicinity in minimal expression, check off all 1-minterms covered by vicinity and go back to step 1.
3. If every minterm has been checked or covered, choose a remaining minterm and determine its compatability set (remember all of the vicinities have already been determined).
4. Calculate the joint vicinity and the conjoin of the compatibility set.

5. If the conjoin is not an implicant of the function or is not an implicant of the vicinity, choose a remaining minterm not processed since the last change and go back to step 4.

6. Otherwise, determine the minimal cost covering of the conjoin, starting with the joint vicinity and, if necessary, adjoining literals in the conjoin.

7. Check those coverings against the members of the compatability set of M and determine if any are minimal cost prime implicants of M.

8. If there is a minimal cost covering of the conjoin which is also a minimal cost prime implicant covering M include the term in the minimal expression, and eliminate all members of the compatability set. Otherwise chose another remaining minterm and go to step 4.

9. This process either terminates in the elimination of all remaining minterms or in the failure of all minterms still remaining to generate any term satisfying the required conditions. In the latter case, choose one of the remaining minterms and generate all of the prime implicants which cover it; if the conjoin is an implicant, this can be limited to those whose cost is less than the minimal covering of the conjoin. Then divide the problem into cases (each case using one of these or the conjoin), eliminate in each case the minterms covered and proceed.

Explanation of the algorithm:

The first part of the algorithm determines those terms which must be included if the answer is to be correct: the essential prime implicants. Since we know that all terms and only terms which are vicinities of 1-minterms that they cover and are also implicants are essential, we hence look for the vicinities of 1-minterms and include them (since we know that if a term A is essential and covers a 1-minterm M, it is either M's vicinity or M's vicinity is not essential, we try to choose essential prime implicants which cover many terms early and, in any event, we need not bother with minterms already covered). Having determined the case, our basic strategy changes to looking for terms which it is safe to include, i.e. which are in at least one minimum cost expression. Terms accepted by the main algorithm are ones which are in at least one minimum cost expression (which includes all terms already accepted). If the main loop of the algorithm does not yield a prime implicant which can be accepted

without risk of being non-minimal, we can at least rely on the fact that every remaining minterm is covered by (at least two) prime implicants and generate, for one minterm, all the prime implicants that cover it and one not excluded by dominance considerations (i.e. all which might possibly be in a minimal cost covering) and divide into cases, each case including one of these (as if it had been chosen by the algorithm). At the end, we chose the case which is minimal.

In each case when the algorithm prescribes a choice, such a choice is quite free, although choice of a promising one can well decrease the number of steps.

Let us illustrate with a few examples. Suppose the function to be simplified is and the cost is the number of literal occurrences (Ex.1)

$CD \backslash AB$	00	01	11	10
00	1^a	1^c	0	?
01	1^b	0	0	1^e
11	0	?	0	0
10	0	1^d	1^g	1^f

The vicinity of b is $\bar{B}\bar{C}$ which is the upper two cells in the first and last columns and hence also covers a and e. Since no 0-minterm is covered, $\bar{B}\bar{C}$ is essential. The remaining minterms and their vicinities are: c-$\bar{A}\bar{D}$, d-B, f-$A\bar{D}$, g-$C\bar{D}$. Choosing c we obtain as compatability set {c,d}. The joint vicinity is $\bar{A}B\bar{D}$ which is also the conjoin. Since $\bar{A}B\bar{D}$ covers no 0's it is an implicant and since it is the joint vicinity it is the minimal cost covering of the conjoin. Since its cost is one more than the vicinity of C it is a minimal cost cover of C and hence can be added, leaving only f and g. The compatability set of f is {f,g}.

The joint vicinity and conjoin is $AC\bar{D}$ which satisfies the required conditions so that a minimum cost covering is: $\bar{B}\bar{C} + \bar{A}B\bar{D} + AC\bar{D}$.

Consider with the same cost function (Ex. 2)

CD \ AB	00	01	11	10
00	1^a	1^c	0	0
01	0	1^d	1^e	0
11	0	0	1^b	1^g
10	1^b	0	0	1^h

The vicinities none of which implicants are:

Minterm	Vicinity	Compatability set	Joint Vicinity	Conjoin
a	$\bar{A}\bar{D}$	abc	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{D}$
b	$\bar{B}\bar{D}$	abh	$\bar{A}\bar{B}C\bar{D}$	$\bar{B}\bar{D}$
c	$\bar{A}\bar{C}$	acd	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}\bar{C}$
d	$B\bar{C}$	cde	$\bar{A}B\bar{C}D$	$B\bar{C}$
e	BC	def	$AB\bar{C}D$	BD
f	AD	efg	ABCD	AD
g	AC	fgh	$A\bar{B}CD$	AC
h	$\bar{B}C$	bgh	$A\bar{B}C\bar{D}$	$\bar{B}C$

Since the required conditions are not met in any case, we chose a minterm, say a. The vicinity is $\bar{A}\bar{D}$ and the remaining literals are $\bar{B}$ and $\bar{C}$. We thus get as candidates $\bar{A}\bar{B}\bar{D}$ and $\bar{A}\bar{C}\bar{D}$ both of which are implicants and hence prime implicants. We get then Case I: $\bar{A}\bar{B}\bar{D}$ + ... and Case II: $\bar{A}\bar{C}\bar{D}$ + ... Then for Case II:

b's new compatability set is {b,h} with joint vicinity and conjoin $\bar{B}C\bar{D}$ which satisifes our conditions. Then g's new compatability set is {f,g} with joint vicinity and conjoin ACD which also does. Then c's compatability set is {d,e} with joint vicinity and conjoin $B\bar{C}D$ which does as well. Since this exhausts all minterms the solution for Case II is:

$\bar{A}\bar{C}\bar{D} + \bar{B}C\bar{D} + ACD + B\bar{C}D$. For Case I $\bar{A}\bar{B}\bar{D}$ covers a and b. Then c's compatability set is {c,d} with joint vicinity and conjoin $\bar{A}B\bar{C}$ which satisfies our conditions. Then e's is {e,f} with joint vicinity and conjoin ABD which also does. Finally g's is {g,h} with ABC as joint vicinity and conjoin and also acceptable. So that the solution for Case I is: $\bar{A}\bar{B}\bar{D} + \bar{A}B\bar{C} + ABD + A\bar{B}C$. Since the cost of the two are equal, either may be chosen.

KARNAUGH MAPS

In our examples, we have used a method of representing partial Boolean functions normally called Karnaugh maps. Although the representation of the functions in Karnaugh map form is not a fundamental aspect of the method and indeed for programmed versions of it, other representations (e.g. the truth-table) are likely to be preferable, it is likely that people utilizing the method manually would find a variety of Karnaugh map representation convenient.

If we consider geometric representations of Boolean functions, it seems apparent that the most intuitive of them is the n-cube, i.e. a figure with two points in each dimension and as many dimensions as there are variables. In the n-cube, minterms are corners, products of literals are subcubes and adjacency (as defined above) is the property of being joined by an edge. This nice correspondence between fundamental geometric and fundamental logical properties would make the (marked) n-cube an ideal tool for representing Boolean functions were it not for the fact that, for virtually all of us, our ability to represent and remember figures in more than three dimensions is virtually nil.

As a result, we find it convenient to use a representation which will slow down the point at which our intuition fails. One such is the Karnaugh map.

For 2 variables a Karnaugh map is essentially an n-cube, that is, it is a table with each physical dimension corresponding to a logical one and with tow points (the obvious ones) on a side, namely

$y \backslash x$	0	1
0	f(0,0)	f(1,0)
1	f(0,1)	f(1,1)

When we extend this to 3 variables, we can without leaving a two-dimensional surface still preserve the properties of correspondence between being in the representation, not a sub-cube, but a sub-rectangle with each dimension having 2^i points and the property of being a product of literals. This can be done by placing two variables (and hence 4 points) in one dimension so that in order the points are x=y=0; x=0,y=1; x=y=1; x=1,y=0 and then placing the whole on a cylinder so that x=1,y=0 is adjacent to x=y=0. In practice instead of actually placing the map on a cylinder, we adopt the convention that both ends of the long side are identified, thus obtaining:

z \ xy	00	01	11	10
0	f(0,0,0)	f(0,1,0)	f(1,1,0)	f(1,0,0)
1	f(0,0,1)	f(0,1,1)	f(1,1,1)	f(1,0,1)

The edges identified by convention are here marked by dotted lines.

With some practice identifying adjacent cells and products of given literals becomes quite easy. Identifying product implicants (i.e. products which cover no 0-minterms) is somewhat more difficult but can be handled.

To go to 4-dimensions we need only to perform the same maneuver in both physical dimensions, thus in effect placing the table on a torus (i.e. a doughnut shaped surface).

To extend this further, additional surfaces are not available and various techniques have been used to try to save one or the other feature. For our purposes the important features are (1) easy identification of adjacency, and (2) ease in identifying which rectangle corresponds to a given product. As a result, the method sometimes called the "supermap" is called for.

For 5 to 8 dimensions we line up the dimensions beyond 4 in such a way as to create a figure arranged like a Karnaugh map except that the cells instead of being minterms are Karnaugh maps of the first 4 variables. Two cells are then adjacent. If, and only if, they are on the same basic level map and are adjacent, or they are in the same position on two adjacent maps, calculation of rectangles then uses the same procedure as in 4-variable maps. Note also that this method can be extended in a regular fashion to any finite number of variables (although, of course, the exponential growth in number of minterms probably makes it impractical beyond 10 to 14 variables).

It is well known that Karnaugh maps can be used directly to cover functions with product implicants non-algorithmically. Historically, doing well with maps of more than 4 variables is limited to experienced practitioners and few people claim proficiency for, say, 8 variables. Viewed as a Karnaugh map method, a great virtue of the current method is the ease at which it can be learned well enough to solve 5-8 variable problems.

SOME REMARKS

In general the method is designed to generate as few extraneous calculations as possible. As a result, the algorithm works on the principle "the best that can be done is all that needs be done." Accordingly (1) since (regardless of which monotonic cost function is involved) every essential prime implicant has to be included (2) since any minterm not in the compatability set of M cannot be covered by any prime implicant that also covers M, we cannot do better than cover M with a least cost prime implicant that covers M and also covers the whole compatability set (3) Every 1-minterm must be covered by some minterm and cannot be covered more cheaply than by its minimum cost covering. Accordingly all terms selected by step 8 of the algorithm can be safely added.

There are a number of additional applications of the "best possible" principle. Since for any remaining minterm M, its vicinity V_M is not a prime implicant but every prime implicant which covers M is equal to $A \cdot V_M$, for some product term A, there is, depending on the cost function, a "best conceivable" way M might be covered. For example, if the cost is the number of literal occurrences, the number of variables in the problem is N, and V_M has m literals, the best we could hope for is to cover it with a rectangle with 2^{n-m+1} calls, so that given the whole configuration of remaining minterms we can define a "best conceivable" solution type which if it exists has to be minimal. If we can spot such a covering, say by direct inspection of the map, we can in some cases shorten the determination by a significant number of steps. For

example, in Example 2, either once we have determined all the vicinities (and hence know there are no essential prime implicants) r, at least, once we see that none of the compatability classes generates a term by the main algorithm, we can note that there are 8 remaining minterms none of which can be covered by prime implicants with less than 3 literals. Consequently the best conceivable covering consists of 4 products of 3 literals each, each of which covers 2 minterms. If that can be accomplished no two of the 4 can overlap. In this case, it is easy to see which they are. Additionally, in this type of observation of the "best conceivable," in any case in which either the conjoin is not an implicant or the joint vicinity has literals not in the conjoin, the best conceivable covering of the minterm will not cover more than one minterm less than the number in the compatability set (since then there is no prime implicant covering the entire compatability set).

Three remarks ought to be made. Firstly, "best conceivable" analysis could if desired be carried out in a step-by-step fashion if someone wanted to; secondly, the main algorithm is on the average so efficient that it is only occasionally worthwhile to bother with "best conceivable," in general, only when we would otherwise need to consider cases via step 9. Even then, the chains of reasoning with "best conceivable" analysis are so complicated that it is normally advisible only if it works out quickly. Otherwise literal use of step 9 is usually indicated. Finally, nothing can be obtained by "best conceivable" analysis which won't eventually be reached by the algorithm.

There is an additional representational move which, while it is by no means necessary, helps speed the process in manual application of the vicinity method, namely, representing the minterm by a sequence of radix-4 digits. This is advantageous because it is closely related to the physical structure of the extended Karnaugh map but also because some easily detectible arithmetic property correspond to an expression including the variable uncomplemented or complemented, respectively.

Specifically, for each pair of variables α, β forming a redex 4 digit:

The presence of	corresponds to
α	digit high (i.e. 2 or 3)
$\bar{\alpha}$	digit low (i.e. 0 or 1)
β	digit odd
$\bar{\beta}$	digit even
$\alpha\ \beta$	digit 3
$\alpha\ \bar{\beta}$	digit 2
$\bar{\alpha}\ \beta$	digit 1
$\bar{\alpha}\ \bar{\beta}$	digit 0

Hence in the six variable problem with variables u,v,w,x,y,z (in that order), the minterms covered by $u\ \bar{x}\ \bar{y}\ z$ are those satisfying:

1. First digit high
2. Second digit even
3. Third digit 1

The speed at which this can be visually detected is gratifying.

EXTENSIONS

From results which are well-known in the literature, it is known that every method which simplifies two level sum of products expressions generates, via De Morgan's Theorems $\overline{A \cdot B} = \bar{A} + \bar{B}$ and $\overline{A+B} = \bar{A} \cdot \bar{B}$, a method for two level product of sum expressions, two level NAND-NAND and two level NOR-NOR expressions. In addition, the pioneer work of McNaughton on multi-output minimization shows that a variant of standard minimization theory such that the prime implicants of the functions themselves and of their products correspond to prime implicants in the single output problem. When this is combined with the methods of this article one gets a variant applicable to the multiple output problem. Similarly, if one wanted to minimize sum-product-sum expressions, it is obvious that a minimal cost expression $F = F_1 + \ldots + F_n$ must satisfy:

1. Each F_i is an implicant of F
2. F_i must be a minimum cost product of sums expression.

If we combine these features and we define a subfunction of F to be a partial function G such that

1. every 1-minterm of G is a 1-minterm of F
2. F and G have the same 0-minterms
3. G has at least one 1-minterm (unless F=0)

it follows that the minimum cost sum-product-sum expression for F is a sum of minimum cost product-sum expressions of subfunctions of F, specifically, the minimal such sum that covers all 1-minterms of F. Consequently (with the help of duality), any two level minimization method can be extended to a three-level one. By repetition, it follows

any two level method can be extended to an n-level one and, since an expression is absolutely minimal if and only if there is an n such that it is both n-level and n+1-level minimal, any two-level minimization method can be extended to an absolute minimization method.

These results while interesting in theory, involve except in extremely simple cases, such a computational load as to normally make them impractical (although if for some unusual reason the desirability of minimizing was so strong as to make one willing to do this, the pay-off for a relatively efficient rather than a relatively inefficient two-level method is great indeed).

SOME DETAILED EXAMPLES

We close with a couple of six-variable examples. In actual cases, the output conditions are determined by a combination of system specifications, systems analysis and the mathematical algorithms which it is intended to realize. Although in different cases, the source of the definition of the switching function to be realized will differ, in each case it will result in a set of condition which imply for each combination of inputs, that combination is either positive, negative or don't care.

For our examples we will, to obviate the danger of inadvertently biasing the case by chosing functions especially favorable, instead use functions generated by a random function generator, i.e. combining the output of a random number generator with a remainder modulo 3 sub-routine to generate zeroes, ones and question marks with equal probability. We chose (for easy understanding) as cost function the number of literal occurrences and to make the problems simple enough to follow but complex

enough to be interesting, chose 6 variable problems:

yz \ wx	00	01	11	10
00	?	0	1	1
01	1	0	?	?
11	1	1	1	0
10	1	?	1	?

yz \ wx	00	01	11	10
00	?	1	1	?
01	0	0	?	1
11	?	?	0	1
10	?	0	1	0

yz \ wx	00	01	11	10
00	0	1	?	?
01	1	0	1	?
11	1	?	0	?
10	1	?	1	1

yz \ wx	00	01	11	10
00	0	0	?	?
01	0	1	1	1
11	1	?	?	?
10	?	?	1	0

As we go through the algorithm we make entries on the map indicating the minterms covered and those which have already been checked for vicinity. To indicate what happens we will repeat the map (without marginal markings) inserting "X" for minterms covered and "Y" for "misfires." Assuming we have no particular geometric insight, a fairly good tactic b to examine first minterms that having a larger h number of adjacent zeroes, but in effect for this purpose we choose on the basis of adjacent zeroes on the same map. It is of course not difficult to spot multiple zeroes on the same map. 232, for example, has 3 adjacent zeroes immediately visible (and no others) and, close by, 223 and 210 have two on the same map (and additional ones on adjacent maps). So we have:

Minterm	Vicinity	Essential
232	$wx\bar{z}$	yes
223	$u\bar{x}z$	no
210	$uv\bar{y}\bar{z}$	yes

?	0	X	1	X X X X
1	0	?	?	0 0 ? 1
1	1	1	0	? ? 0 Y
1	?	X	?	? 0 X 0
0	1	X	?	0 0 X ?
1	0	1	?	0 1 1 1
1	?	0	?	1 ? ? ?
1	?	X	1	? ? X 0

Looking for further plausible cases:

311	uyxz	yes	?	0	X	X	X X X X
101	$\bar{u}\bar{x}z$	no	1	0	X	X	0 0 X X
110	$\bar{u}vx\bar{z}$	yes	1	1	1	0	? ? 0 Y
131	$w\bar{y}$	yes	1	?	X	?	? 0 X 0
			0	X	X	X	0 0 X X
			Y	0	X	X	0 X X X
			1	?	0	?	1 X X ?
			1	X	X	1	? ? X 0

Since this exhausts the plausible candidates, we now go through the others in a systematic fashion, entering results in a table for the next step. Of course, if additional essential prime implicants are found, the table is adjusted so that only remaining 1-minterms are listed.

Process Order	Minterm	Vicinity	Ess.?	Compatability Set	Joint Vicinity	Conjoin	Remarks
	001	$\bar{u}\bar{x}$	no				✓
	002	$\bar{1}$	no				✓
	003	$\bar{w}$	no				✓
	013	$\bar{y}$	no				✓
1	033	$\bar{u}vx$	no	013,033	$\bar{u}vxy$	$\bar{u}vxyz$	✓
2	101	$\bar{u}\bar{x}z$	no	001,003, 101,103	$\bar{u}\bar{w}\bar{x}z$	$\bar{u}\bar{w}\bar{x}z$	✓
	102	$\bar{y}$	no				✓
	103	$\bar{1}$	no				✓
4	122	$\bar{u}$	no	002,102,122	$\bar{u}y$	$\bar{u}\bar{x}yz$	✓
3	223	$u\bar{x}z$	no	223,303	$ux\bar{y}z$	$u\bar{x}yz$	✓
	303	$\bar{y}$	no				✓

starting with small vicinities being advisible in the absence of reason to the contrary, we start with 033. Its vicinity, $\bar{u}\bar{v}x$ translates to "first digit 0, second odd." Results are indicated above. $\bar{u}\bar{v}xy$ being a cover of the compatability set, minimal cost for 033 and an implicant it is adopted. 101's vicinity, $\bar{u}\bar{x}z$ translates to "first low, second even, third odd." $\bar{u}\bar{w}\bar{x}z$ satisfies the three conditions and so is adopted. 223's vicinity translates as "first high, second even, third odd." $u\bar{x}yz$ satisfies the three conditions and so is adopted. We now have only three minterms left: 002, 102 and 122. 122's vicinity $\bar{u}$ translates as "first digit low' which all three satisfy. Since (1) any prime implicant which covers a remaining minterm must have at least one move literal than the vicinity expression, (2) any prime implicant that covers both 102 and 122 must be covered by $\bar{u}y$ (3) $\bar{u}y$ is not an implicant (4) $\bar{u}yz$ is an implicant and covers the whole compatability set, it follows that we cannot do better than $\bar{u}yz$ (we could also get this result by examining the five products with $\bar{u}$ and on finding that none are implicants, the main algorithm lets us choose $\bar{u}yz$). A minimum cost covering is therefore: $wx\bar{z} + u\bar{v}\bar{y}\bar{z} + uvxz + \bar{u}vx\bar{z} + w\bar{y} + \bar{u}\bar{v}xy + \bar{u}\bar{w}\bar{x}z + u\bar{x}yz + \bar{u}yz$.

Another randomly generated case is:

x \ u

yz \ wx	00	01	11	10
00	1	1	?	0
01	0	0	0	1
11	0	1	?	?
10	?	?	?	1

yz \ wx	00	01	11	10
00	0	0	0	?
01	?	1	?	?
11	0	0	1	1
10	?	1	1	1

yz \ wx	00	01	11	10
00	1	0	0	0
01	0	1	?	?
11	1	0	?	?
10	0	?	?	1

yz \ wx	00	01	11	10
00	1	1	?	?
01	1	?	0	0
11	1	?	0	1
10	0	0	0	0

Following the same notation and strategy as in the last example, we find the following plausible candidates

Minterm	Vicinity	Essential							
100	$\bar{w}\bar{x}\bar{y}z$	no	1	1	?	0	0	0	0	?
103	$v\bar{x}yz$	yes	0	0	0	1	?	1	?	?
111	$vw\bar{y}z$	no	0	X	X	?	0	0	1	1
013	$\bar{u}\bar{v}xy$	yes	?	X	X	1	?	1	1	1
			Y	0	0	0	1	1	?	?
			0	Y	?	?	1	?	0	0
			X	0	?	X	X	?	0	X
			0	?	?	1	0	0	0	0

Other plausible cases are:

000	$\bar{u}\bar{w}\bar{z}$	no	y	1	?	0	0	0	0	?
021	$w\bar{x}z$	no	0	0	0	Y	?	Y	?	?
122	$\bar{u}wy$	yes	0	X	X	X	0	0	1	1
211	$u\bar{y}z$	no	X	X	X	X	X	X	X	X
212	$\bar{v}y\bar{z}$	yes								
			Y	0	0	0	1	1	?	?
			0	Y	?	?	1	?	0	0
			X	0	X	X	X	?	0	X
			0	?	X	X	0	0	0	0

Our plausible cases being exhausted we now calculate:

Process Order	Minterm	Vicinity	Ess	Compatability Set	Joint Vicinity	Conjoin	
	000	$\bar{u}\bar{w}\bar{z}$	no				3
3	010	$\bar{u}\bar{v}\bar{z}$	no	000,010	$\bar{u}\bar{v}\bar{w}\bar{z}$	$\bar{u}\bar{v}\bar{w}\bar{y}\bar{z}$	3
5	021	$\bar{w}\bar{x}z$	no	021,223	$\bar{w}\bar{x}z$	$\bar{v}\bar{w}\bar{x}z$	5
1, 4	100	$\bar{w}\bar{x}\bar{y}\bar{z}$	no	100,300	$\bar{v}\bar{w}\bar{x}\bar{y}\bar{z}$	$\bar{v}\bar{w}\bar{x}\bar{y}\bar{z}$	4
2	111	$vx\bar{y}z$	no	111	$vx\bar{y}z$	$\bar{u}v\bar{w}x\bar{y}z$	2
6	211	$u\bar{y}z$	no	211,301	$u\bar{w}\bar{y}z$	$u\bar{w}\bar{y}z$	6
	223	w	no				5
8	233	$\bar{v}w$	no	233	$\bar{v}w$	$u\bar{v}wxyz$	8
	300	$v\bar{y}$	no				4
	301	$u\bar{w}$	no				6
7	310	$uv\bar{y}$	no	310	$uv\bar{y}$	$uv\bar{w}v\bar{y}\bar{z}$	7

Starting with small vicinities, we start with 100. Its vicinity translates to "second digit 0, third 0," giving us 000, 100 and 300 as its compatability set. Since its vicinity has 4 literals, every prime implicant covering it will have at least 5 and hence only cover two minterms at most. We hence pass to 111. Its vicinity gives "1st digit odd, 2nd odd, third 1." 111 is the only remaining 1-minterm which satisfies this. Hence since $\bar{u}vx\bar{y}z$ is an implicant, it can thus be added. Moving to 3 literal vicinities, 010 gives "1st digit 0, third even," giving 000 and 010 as compatability set. Since $\bar{u}\bar{v}\bar{w}\bar{z}$ is only one literal move than the vicinity and is an implicant, it can be added. This reduces the compatability set of 100 to 100 and 300. Hence $v\bar{w}\bar{x}\bar{y}\bar{z}$ can be added. Minterm 021 gives condition "second digit 2, third odd" giving as compatability set 021 and 223. Their conjoin is $\bar{v}w\bar{x}z$ which is one literal longer than the vicinity $\bar{w}\bar{x}z$ and is an implicant and hence may be added. Minterm 211's vicinity gives "first digit high third 1" giving as compatability set 211 and 301.

This gives $u\bar{w}\bar{y}z$ as joint vicinity and conjoin. Since it is an implicant and one literal longer than the vicinity, it may be added. This leaves only 233 and 310. Minterm 310's vicinity gives "first digit 3, third low," with only 310 as compatability set. Hence $uv\bar{w}\bar{y}$, being an implicant may be added. This leaves only 233 remaining and since $\bar{v}wy$ is an implicant and is one literal larger, it likewise may be added.

Thus, one minimal cost expression is:

$$v\bar{x}yz + \bar{u}\bar{v}wy + \bar{u}wy + \bar{v}y\bar{z} + \bar{u}vx\bar{y}z + \bar{u}\bar{v}w\bar{z} + v\bar{w}\bar{x}\bar{y}\bar{z} + \bar{v}w\bar{x}z + u\bar{w}\bar{y}z + uv\bar{w}\bar{y} + \bar{v}wy$$