International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:5, 2007

Heuristic Set-Covering-Based Postprocessing for Improving the Quine-McCluskey Method

Miloš Šeda

***Abstract*—**Finding the minimal logical functions has important applications in the design of logical circuits. This task is solved by many different methods but, frequently, they are not suitable for a computer implementation. We briefly summarise the well-known Quine-McCluskey method, which gives a unique procedure of computing and thus can be simply implemented, but, even for simple examples, does not guarantee an optimal solution. Since the Petrick extension of the Quine-McCluskey method does not give a generally usable method for finding an optimum for logical functions with a high number of values, we focus on interpretation of the result of the Quine-McCluskey method and show that it represents a set covering problem that, unfortunately, is an NP-hard combinatorial problem. Therefore it must be solved by heuristic or approximation methods. We propose an approach based on genetic algorithms and show suitable parameter settings.

***Keywords*—**Boolean algebra, Karnaugh map, Quine-McCluskey method, set covering problem, genetic algorithm.

I. Introduction

FOR minimisation of logical functions, laws of the Boolean algebra and the Karnaugh maps are mostly used.

The Karnaugh maps represent a very efficient graphical tool for minimising logical functions with no more than 6 variables. However, their use is based on visual recognition of adjacent cells and, therefore, the method is not suitable for automated processing on computers. A direct application of the Boolean algebra laws is not restricted in this way, but there is no general algorithm defining the sequence of their application and thus this approach is not suitable for computer implementation either.

With the growing strength of computational techniques, further investigations were focused on an algorithm-based technique for simplifying Boolean logic functions that could be used to handle a large number of variables.

The well-known method usable on computers is the algorithm proposed by Edward J. McCluskey, professor of electrical engineering at University of Standford, and philosopher Willard van Orman Quine from Harvard University [13], [18].

We will assume that the number of variables n may be high but restricted in the sense that all 2^n rows of the corresponding truth table may be saved in a memory and thus may be processed. If this condition is not satisfied, then we could only work with a selected number of truth table rows. This approach is possible, e.g., in image filters [19]. Of course, this is inacceptable in real logic control applications where all input combinations must be tested. In the next considerations we will assume the full truth table.

Manuscript received June 15, 2007. This work was supported in part by the Ministry of Education, Youth and Sports of the Czech Republic under research plan MSM 0021630518 "Simulation Modelling of Mechatronic Systems".

Miloš Šeda works in the Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology, Technická 2896/2, CZ 616 69 Brno, Czech Republic (phone: +420-54114 3332; fax: +420-54114 2330; e-mail: seda@fme.vutbr.cz).

II. Quine-McCluskey Method

The Quine-McCluskey method [13], [17], [18] is an exact algorithm based on systematic application of the *distributive law* (1), *complement law* (2) and *idempotence law* (3), i.e. the laws as follows:

$$x.(y+z) = xy + xz \tag{1}$$

$$x + \bar{x} = 1 \tag{2}$$

$$x + x = x \tag{3}$$

From (1) and (2), we can easily derive the *uniting theorem* (4)

$$xy + x\bar{y} = x.(y+\bar{y}) = x.1 = x \tag{4}$$

In general, this means that two expressions with the same set of variables differing only in one variable with complementary occurrence may be reduced to one expression given by their common part, e.g.

$$f = xy\bar{z}w + \bar{x}y\bar{z}w = x\bar{z}w(x+\bar{x}) = x\bar{z}w.1 = x\bar{z}w$$

For next considerations we will need the following notions:

Literal is a variable or its negation.

Term is a conjunction (= product) of literals.

Minterm is a conjunction of literals including every variable of the function exactly once in true or complemented form.

For example, if $f(x,y,z) = x\bar{y}z + \bar{x}\bar{y}$, then $x\bar{y}z$ and $\bar{x}\bar{y}$ are terms and minterms are represented by conjunctions $x\bar{y}z$, $\bar{x}\bar{y}z$ and $\bar{x}\bar{y}\bar{z}$ because $\bar{x}\bar{y} = \bar{x}\bar{y}(z+\bar{z}) = \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$.

Disjunctive normal form (DNF) of a formula f is a formula f' such that f' is equivalent to f and f' has the form of a disjunction of conjunctions of literals.

Canonical (or *complete*) *disjunctive normal form* (CDNF) is a DNF where all conjunctions are represented by minterms.

Now we can describe a skeleton of the Quine-McCluskey method:

1. Minterms of a given Boolean function in CDNF are divided into groups such that minterms with the same number of negations are in the same group.
2. All pairs of terms from adjacent groups (i.e. groups whose number of negations differ by one) are compared.

3. If compared terms differ only in one literal (and the uniting theorem may be applied) then these terms are removed and a term reduced to a common part is carried to the next iteration.
4. All terms are used only once, i.e. if necessary the idempotence law is applied.
5. If the number of terms in the new iteration is nonzero, then steps 2-5 are repeated, otherwise the algorithm finishes.

The result of the algorithm is represented by the terms that were not removed, i.e. that could not be simplified.

Now, we will apply the Quine-McCluskey method to a simple logical function from [5] and, by means of the Karnaugh map, we will show that it will not find its minimal form.

Example 1 Minimise the following logical function in CDNF.

$$f = \bar{x}yzw + x\bar{y}zw + xy\bar{z}w + \\ + \bar{x}yz\bar{w} + x\bar{y}z\bar{w} + xyz\bar{w} + x\bar{y}\bar{z}w + \\ + xy\bar{z}\bar{w}$$

Solution: If we divide the minterms into groups by the number of their negations, then we get this initial iteration:

(0) $\bar{x}yzw$; $x\bar{y}zw$; $xy\bar{z}w$; $\bar{x}yz\bar{w}$; $x\bar{y}z\bar{w}$; $xyz\bar{w}$; $x\bar{y}\bar{z}w$; $xy\bar{z}\bar{w}$

If we compare all pairs of terms from adjacent groups in the initial iteration (0) and place in frames terms that can be simplified and divide the resulting terms in the next iteration (1) into groups again, then we get:

(0) $\boxed{\bar{x}yzw}$; $\boxed{x\bar{y}zw}$; $\boxed{xy\bar{z}w}$; $\boxed{\bar{x}yz\bar{w}}$; $\boxed{x\bar{y}z\bar{w}}$; $\boxed{xyz\bar{w}}$; $\boxed{x\bar{y}\bar{z}w}$; $\boxed{xy\bar{z}\bar{w}}$; $\boxed{x\bar{y}\bar{z}\bar{w}}$

(1) $\bar{x}yz$; $\bar{x}zw$; $x\bar{y}z$; $\bar{y}zw$; $x\bar{y}w$; $xy\bar{z}$; $x\bar{z}w$; $\bar{x}z\bar{w}$; $\bar{y}z\bar{w}$; $x\bar{y}\bar{z}$;

Applying the previous steps to iteration (1), we get:

(1) $\boxed{\bar{x}yz}$; $\boxed{\bar{x}zw}$; $\boxed{x\bar{y}z}$; $\boxed{\bar{y}zw}$; $x\bar{y}w$; $xy\bar{z}$; $x\bar{z}w$; $\boxed{\bar{x}z\bar{w}}$; $\boxed{\bar{y}z\bar{w}}$; $\boxed{x\bar{y}\bar{z}}$;

(2) $\bar{x}z$; $\bar{y}z$

In iteration (2) we have only one group of terms and thus no simplified terms can be generated and the algorithm finishes. Its result f_m is given by a disjunction of those terms that cannot be simplified.

$$f_m = \bar{x}z + \bar{y}z + x\bar{y}w + xy\bar{z} + x\bar{z}w$$

Let us consider the same example and solve it using the Karnaugh map. For the given Boolean function f, we get the following map

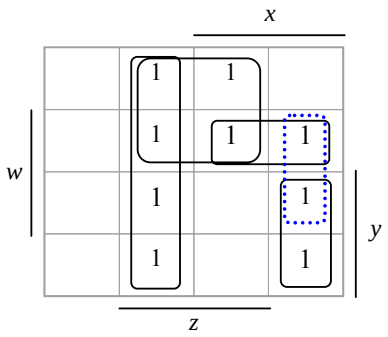

Fig. 1 Karnaugh map

From this map, we get two possible solutions of a minimal logical function depending on the way of covering the first two cells with logical 1 in the last columns.

a) $f_{1\min} = \bar{x}z + \bar{y}z + x\bar{y}w + xy\bar{z}$

b) $f_{2\min} = \bar{x}z + \bar{y}z + x\bar{z}w + xy\bar{z}$

We can see that the result gained using Karnaugh map includes a lower number of terms than the result of computation by means of Quine-McCluskey method.

S. R. Petrick proposed a modification of Quine-McCluskey method that tries to remove the resulting redundancy [16]. The Quine-McCluskey simplification technique with special modifications is presented in [21].

Before describing the second phase of computation, we will define several notions.

(i) A Boolean term g is an *implicant* of a Boolean term f if:
- each literal in g is also obtained in f (i.e. if g has the form $g(x_1, \ldots, x_n)$ then f has the form $f(x_1, \ldots, x_n, y_1, \ldots, y_m)$), and
- for all combinations of literals the implication $g \to f$ has a value of True.

We know that the implication is defined by Table 1. From this definition, we get that g is an *implicant* of f if and only if $g \le f$.

In our case, that means that the second property is satisfied if $g(x_1, \ldots, x_n) \le f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ for each selection of $x_1, \ldots, x_n$ and each selection of $y_1, \ldots, y_m$.

TABLE I
IMPLICATION

g	f	$g \to f$
0	0	1
0	1	1
1	0	0
1	1	1

Therefore, we can consider only those values of variables at which g is true and, for all these cases, f must also be true.

(ii) The terms resulting from the Quine-McCluskey method are called *prime implicants*.

In the Petrick modification of the Quine-McCluskey method, a table of prime implicants given by the results of the first phase is built. It has prime implicants in the heads of columns, minterms from a given CNDF are in the heads of rows and, in cells representing intersections of rows and columns, a selected symbol (e.g. ∗) is placed if the prime implicant in question is a subset of a corresponding minterm. We say that prime implicants *cover* minterms.

In our example then we get Table II as follows.

TABLE II
COVERING OF MINTERMS BY PRIME IMPLICANTS

	$\bar{x}z$	$\bar{y}z$	$x\bar{y}w$	$xy\bar{z}$	$x\bar{z}w$
$\bar{x}yzw$	*				
$\bar{x}\bar{y}zw$		*	*		
$x\bar{y}\bar{z}w$				*	*
$\bar{x}y\bar{z}\bar{w}$	*				
$\bar{x}\bar{y}z\bar{w}$		*			
$xy\bar{z}\bar{w}$				*	
$\bar{x}\bar{y}zw$	*	*			
$x\bar{y}\bar{z}w$			*		*
$\bar{x}\bar{y}\bar{z}w$	*	*			

Although the prime implicants cannot be simplified, some of them can be redundant and thus may be omitted. However, symbols ∗ in columns of considered prime implicants must be spread into all rows, otherwise the disjunction of prime implicants would not express the initial canonical (complete) disjunctive normal form, i.e., there would be a non-covered minterm.

If there is a row in which ∗ occurs only once, then the prime implicant in the corresponding column cannot be omitted. Such implicants are called *essential prime implicants*. In our example there are three essential prime implicants: $\bar{x}z$, $\bar{y}z$ and $xy\bar{z}$.

Now we will simplify Table II by deleting columns of essential prime implicants and rows that contain their ∗ symbols.

We get Table 3.

TABLE III

	$x\bar{y}w$	$x\bar{z}w$
$\bar{x}\bar{y}\bar{z}w$	*	*

It can be easily seen that, in simplified Table III, the minterm $\bar{x}\bar{y}\bar{z}w$ may be covered by the prime implicant $x\bar{y}w$ or by $x\bar{z}w$. Hence we get two minimal disjunctive forms:

$f_{1\min} = \bar{x}z + \bar{y}z + x\bar{y}w + xy\bar{z}$ and

$f_{2\min} = \bar{x}z + \bar{y}z + x\bar{z}w + xy\bar{z}$,

which agrees with the previous solution by the Karnaugh map.

Although computations by the Quine-McCluskey method with the Petrick modification have a wider use than the approach based on the Karnaugh map, however, this improved method has also its restrictions. The main problem is in the procedure of covering minterms by prime implicants. It can be easily found if the number of the essential prime implicants is high and thus the table of prime implicants will be very reduced. In general, we cannot expect this, and it is even possible that none of the prime implicants is essential. From the combinatorial optimisation theory, it is known that the *set covering problem* is NP-hard and its time complexity grows exponentially with the growing input size.

It can be shown that, for a CNDF with n minterms, the upper bound of the number of prime implicants is $3^n/n$. If n=32, we can get more than $6.5 * 10^{15}$ prime implicants. To solve such a large instance of the set covering problem is only possible by heuristics (stochastic or deterministic) [1], [2], [4], [6], [10], [11], [14], [15], [20], [22] or by approximation [7], [8] methods. However, they do not guarantee an optimal solution.

A formal definition of the problem and possible solution by a genetic algorithm is proposed in the next section.

III. SET COVERING PROBLEM (SCP)

The set covering problem (SCP) is the problem of covering the rows of a m-row, n-column, zero-one matrix (a_{ij}) by a subset of columns at a minimal cost. Defining x_j=1 if column j (with cost c_j=>0) is in the solution and x_j=0 otherwise, the SCP is:

$$\text{Minimise } \sum_{j=1}^{n} c_j x_j \tag{5}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \ i = 1,2,...,m, \tag{6}$$

$$x_j \in \{0,1\}, \ j = 1,2,...,n \tag{7}$$

Constraints (6) ensure that each row is covered by at least one column and (7) is a constraint guaranteeing integers.

In general, the cost coefficients c_j are positive integers. Here, in the application of the SCP for the second phase of the Quine-McCluskey method, we assume all c_j equal to 1 because we try to minimise the number of the covering columns. This special case of SCP is called a *unicost SCP*. ❑

SCP has, besides Quine-McCluskey method, a wide range of applications, for example *vehicle* and *crew scheduling* [12], *facilities location* [1], *assembly line balancing* and *Boolean expression simplification*. There are number of other combinatorial problems that can be formulated as, or transformed to, SCP such as the *graph colouring problem* [1] and the *independent vertex set problem* [7].

A fuzzy version of SCP is studied in [3] and [9].

Since we use, for solving the SCP, a modified version of the genetic algorithm (GA) proposed by Beasley and Chu [2], [4] for the non-unicost SCP, we summarise the basic

properties of GAs

IV. Genetic Algorithm

The skeleton for GA can be described as follows:

```
generate an initial population ;
evaluate fitness of individuals in the population ;
repeat
   select parents from the population;
   recombine (mate) parents to produce children ;
   evaluate fitness of the children ;
   replace some or all of the population by the children
until a satisfactory solution has been found ;
```

Since the principles of GAs are well-known, we will only deal with GA parameter settings for the problems to be studied. Now we describe the general settings.

Individuals in the population (*chromosomes*) are represented as binary strings of length n, where a value of 0 or 1 at the i-th bit (*gene*) implies that $x_i = 0$ or 1 in the solution respectively.

The *population size* N is usually set between n and $2n$. Many empirical results have shown that population sizes in the range [50, 200] work quite well for most problems.

Initial population is obtained by generating random strings of 0s and 1s in the following way: First, all bits in all strings are set to 0, and then, for each of the strings, randomly selected bits are set to 1 until the solutions (represented by strings) are feasible.

The *fitness function* corresponds to the objective function to be maximised or minimised.

There are three most commonly used methods of *selection* of two parent solution for *reproduction*: proportionate selection, ranking selection, and tournament selection. The tournament selection is perhaps the simplest and most efficient among these three methods. We use the *binary tournament selection* method where two individuals are chosen randomly from the population. The more fit individual is then allocated a reproductive trial. In order to produce a child, two binary tournaments are held, each of which produces one parent.

The *recombination* is provided by the *uniform crossover* operator, which has a better recombination potential than do other crossover operators as the classical *one-point* and *two-point* crossover operators. The uniform crossover operator works by generating a random crossover mask B (using Bernoulli distribution) which can be represented as a binary string $B = b_1 b_2 b_3 \cdots b_{n-1} b_n$ where n is the length of the chromosome. Let P_1 and P_2 be the parent strings $P_1[1]$, **...** ,$P_1[n]$ and $P_2[1]$, **...** ,$P_2[n]$ respectively. Then the child solution is created by letting: $C[i] = P_1[i]$ if $b_i = 0$ and $C[i] = P_2[i]$ if $b_i = 1$. *Mutation* is applied to each child after crossover. It works by *inverting M* randomly chosen bits in a string where M is experimentally determined. We use a mutation rate of $5/n$ as a lower bound on the optimal mutation rate. It is equivalent to mutating five randomly chosen bits per string.

When v child solutions have been generated, the children will replace v members of the existing population to keep the population size constant, and the reproductive cycle will restart. As the replacement of the whole parent population does not guarantee that the best member of a population will survive into the next generation, it is better to use *steady-state* or *incremental replacement* which generates and replaces only a few members (typically 1 or 2) of the population during each generation. The *least-fit* member, or a randomly selected member with *below-average fitness*, are usually chosen for replacement.

Termination of a GA is usually controlled by specifying a maximum number of generations t_{max} or relative improvement of the best objective function value over generations. Since the optimal solution values for most problems are not known, we choose $t_{max} \le 5000$.

V. Genetic Algorithm for SCP

The chromosome is represented by an n-bit binary string S where n is the number of columns in the SCP. A value of 1 for the j-th bit implies that column j is in the solution and 0 otherwise.

Since the SCP is a minimisation problem, the lower the fitness value, the more fit the solution is. The fitness of a chromosome for the unicost SCP is calculated by (8).

$$f(S) = \sum_{j=1}^{n} S[j] \tag{8}$$

The binary representation causes problems with generating infeasible chromosomes, e.g. in initial population, in crossover and/or mutation operations. To avoid infeasible solutions a *repair operator* is applied.

Let
$I = \{1, \ldots, m\}$ = the set of all rows;
$J = \{1, \ldots, n\}$ = the set of all columns;
$\alpha_i = \{j \in J \mid a_{ij} = 1\}$ = the set of columns that cover row i, $i \in I$;
$\beta_j = \{i \in I \mid a_{ij} = 1\}$ = the set of rows covered by column j, $j \in J$;
S = the set of columns in a solution;
U = the set of uncovered rows;
w_i = the number of columns that cover row i, $i \in I$ in S.

The repair operator for the unicost SCP has the following form:

initialise $w_i := | S \cap \alpha_i |$, $\forall i \in I$;
initialise $U := \{ i \mid w_i = 0, \forall i \in I \}$;
for each row i in U (in increasing order of i) **do**
  begin find the first column j (in increasing order of j)
    in α_i that minimises $1/|U \cap \beta_j|$;
    $S := S + j$;
    $w_i := w_i + 1, \ \forall i \in \beta_j$;
    $U := U - \beta_j$
  end ;
for each column j in S (in decreasing order of j) **do**
  if $w_i \ge 2, \ \forall i \in \beta_j$
  then begin $S := S - j$;
    $w_i := w_i - 1, \ \forall i \in \beta_j$
  end ;
{ S is now a feasible solution to the SCP and contains no redundant columns }

Initialising steps identify the uncovered rows. **For** statements are "greedy" heuristics in the sense that in the 1st

for, columns with low cost-ratios are being considered first and in the 2nd **for**, columns with high costs are dropped first whenever possible.

VI. Conclusion

In this paper, we studied the problem of minimising the Boolean functions with a rather high number of variables. Since traditional approaches based on the Boolean algebra or Karnaugh maps have restrictions in the number of variables and the sequence of laws that could be applied is not unique, we focus on the well-known Quine-McCluskey method. Since it does not guarantee the finding of an optimal solution, we must apply a postprocessing phase. Unfortunately, the data resulting from the Quine-McCluskey method are in the form of a unicost set covering problem, which is NP-hard. Therefore, for logical functions, the obvious Petrick's extension of the Quine-McCluskey method cannot be applied, and heuristic or approximation method must be used instead. We proposed a genetic algorithm-based approach and discussed problem-oriented parameter settings.

In the future, we are going to implement also other stochastic heuristics, such as simulated annealing and tabu-search, and compare them with the genetic algorithm.

References

[1] L. Brotcorne, G. Laporte and F. Semet, "Fast Heuristics for Large Scale Covering-Location Problems," *Computers & Operations Research*, vol. 29, pp. 651-665, 2002.
[2] J.E. Beasley P.C. Chu, "A Genetic Algorithm for the Set Covering Problem," *Journal of Operational Research*, vol. 95, no. 2, pp. 393-404, 1996.
[3] C.I. Chiang, M.J. Hwang and Y.H. Liu, "An Alternative Formulation for Certain Fuzzy Set-Covering Problems," *Mathematical and Computer Modelling*, vol. 42, pp. 363-365, 2005.
[4] P. Chu, "A Genetic Algorithm Approach for Combinatorial Optimisation Problems," PhD thesis, The Management School Imperial College of Science, Technology and Medicine, London, 1997.
[5] K. Čulík, M. Skalická, I. Váňová, *Logic* (in Czech). Brno: VUT FE, 1968.
[6] P. Galinier and A. Hertz, "Solution Techniques for the Large Set Covering Problem," *Discrete Applied Mathematics*, vol. 155, pp. 312-326, 2007.
[7] F.C. Gomes, C.N. Meneses, P.M. Pardalos and G.V.R. Viana, "Experimental Analysis of Approximation Algorithms for the Vertex Cover and Set Covering Problems," *Computers & Operations Research*, vol. 33, pp. 3520-3534, 2006.
[8] T. Grossman and A. Wool, "Computational Experience with Approximation Algorithms for the Set Covering Problem," *European Journal of Operational Research*, vol. 101, pp. 81-92, 1997.
[9] M.J. Hwang, C.I. Chiang and Y.H. Liu, "Solving a Fuzzy Set-Covering Problem," *Mathematical and Computer Modelling*, vol. 40, pp. 861-865, 2004.
[10] G. Lan and G.W. DePuy, "On the Effectiveness of Incorporating Randomness and Memory into a Multi-Start Metaheuristic with Application to the Set Covering Problem," *Computers & Industrial Engineering*, vol. 51, pp. 362-374, 2006.
[11] G. Lan, G.W. DePuy and G.E. Whitehouse, "An Effective and Simple Heuristic for the Set Covering Problem," *European Journal of Operational Research*, vol. 176, pp. 1387-1403, 2007.
[12] M. Mesquita and A. Paias, "Set Partitioning/Covering-Based Approaches for the Integrated Vehicle and Crew Scheduling Problem," *Computers & Operations Research*, in press, 2006.
[13] E.J. McCluskey, "Minimization of Boolean Functions," *Bell System Technical Journal*, vol. 35, no. 5, pp. 1417-1444, 1956.
[14] A. Monfroglio, "Hybrid Heuristic Algorithms for Set Covering," *Computers & Operations Research*, vol. 25, pp. 441-455, 1998.
[15] M. Ohlsson, C. Peterson and B. Söderberg, "An Efficient Mean Field Approach to the Set Covering Problem," *European Journal of Operational Research*, vol. 133, pp. 583-595, 2001.
[16] S.K. Petrick, "On the Minimization of Boolean Functions," in *Proceedings of the International Conference Information Processing*, Paris, 1959, pp. 422-423.
[17] Ch. Posthoff and B. Steinbach, *Logic Functions and Equations: Binary Models for Computer Science*. Berlin: Springer-Verlag, 2005.
[18] W.V. Quine, "The Problem of Simplifying Truth Tables," *American Mathematical Monthly*, vol. 59, no. 8, pp. 521-531, 1952.
[19] L. Sekanina, *Evolvable Components*. Berlin: Springer-Verlag, 2003.
[20] M. Solar, V. Parada and R. Urrutia, "A Parallel Genetic Algorithm to Solve the Set-Covering Problem," *Computers & Operations Research*, vol. 29, pp. 1221-1235, 2002.
[21] S.P. Tomaszewski, I.U. Celik and G.E. Antoniou, "WWW-Based Boolean Function Minimization," *International Journal of Applied Mathematics and Computer Science*, vol. 13, no. 4, pp. 577–583, 2003.
[22] M. Yagiura, M. Kishida and T. Ibaraki, "A 3-Flip Neighborhood Local Search for the Set Covering Problem," *European Journal of Operational Research*, vol. 172, pp. 472-499, 2006.