ALGORITHM FOR MINIMIZATION OF PARTIAL BOOLEAN FUNCTIONS

Jan Hlavička, Petr Fišer
Czech Technical University, Karlovo nám. 13, 121 35 Prague 2
e-mail: hlavicka@fel.cvut.cz, fiserp@fel.cvut.cz

Abstract. *The paper presents a minimization algorithm for Boolean functions defined by truth tables. It treats only minterms having defined output values and is thus efficient above all in case of highly undefined functions, where the methods processing also the don't care states lose much time. The algorithm proves its full strength for functions with several hundreds of variables and several hundreds of minterms with defined output values. Its programmed version gave in these cases better results (in terms of the minimality of the solution and runtime) than the state-of-the-art ESPRESSO.*

1 Introduction

Synthesis of combinational functions with large numbers of input variables is a problem that appears in different contexts. It may be encountered e.g. in the design of control systems, where a large number of sensors deliver their input data to be processed by an automaton, or in diagnostics of logic circuits, where a sequence of code words is to be transformed into the sequence of test patterns. A common feature of those problems is the disproportion between the total number of minterms existing for the given number of input variables and the number of really used minterms, for which the output value is defined (care minterms). Hence an efficient synthesis method, whose time and memory complexity will be acceptable even for large problems with several hundreds of input variables and several hundreds of care minterms, is desirable.

One of the best known and probably the most successful of all programs for minimization of switching functions, called ESPRESSO [1], [2], uses a heuristic procedure for local search. This means that starting from some initial solution, the procedure tries to improve the quality of the solution through successive modifications. These modifications are directed by some quality criterion evaluating the newly reached situation. A similar approach was used also in other methods listed in [1], like e.g. [3]. A kind of local search is used also in our case, although the heuristic is different.

2 Problem Statement

Let us have a Boolean function of n input variables $F(x_1, x_2, \ldots x_n)$, whose output values are defined by a truth table. The number of 1-minterms and 0-minterms is equal to u and z respectively, the rest are don't care states. The function is highly undefined, i.e. only few of the 2^n minterms have an output value assigned ($u+v<<2^n$). Our task is to formulate a synthesis algorithm, which will produce a two-level disjunctive form of F, whose complexity is close to the minimal disjunctive form.

3 Prime Implicant Generation

When generating a prime implicant (PI) for a given function F, we may proceed either by increasing the dimensionality of an implicant (which is the usual approach), or by reducing the size of a hypercube which contains the PI as its subset. In a special case it may be an implicate of the function F. When reducing the size of a hypercube, we must repetitively add literals to its term, until the hypercube becomes an implicant (possibly a PI). This happens when no 0-minterm is covered any more. Symmetrically, we generate a PI by removing literals from an implicant until we reach its maximum size without covering any 0-minterm. These two principles will be used as basis for two implicant generation methods. The first one, denoted as **coverage-directed search**,

combines the implicant generation with solution of the covering problem. The second one is denoted as **sequential search**, because the candidates for rejection from a term are selected one by one.

3.1 Coverage-Directed Search

First we start with PI generation method based on hypercube reduction. Let us have a single-output Boolean function *F* defined by its on-set and off-set (1-minterms and 0-minterms). As the first step, we select the most frequent input literal from the given on-set and use it as a term from which an implicant will be derived. After every selection we must verify whether the term already corresponds to an implicant by comparing it with all 0-minterms. If we obtain an implicant, we record the term and continue searching for other implicants (see below). If not, some other literal must be added. We divide the given on-set into two subsets. One subset contains those minterms, which cannot be covered by any term containing the selected literal (minterms containing the literal with the opposite polarity). This subset will not be considered any more. From the other subset, we again select the most frequent literal and add it to the previous one, so we have a two-literal product term. Again we compare this term with the 0-minterms and check if it is an implicant. We repeat this procedure until we get an implicant. We record this implicant and remove from the original on-set those minterms, which are covered by this implicant. Thus we obtain a reduced on-set containing only uncovered minterms. Now we repeat the procedure from the beginning selecting the next most frequently used literal, until the whole on-set is covered. The output of this algorithm is a sum of product terms, covering all 1-minterms and no 0-minterm.

When selecting the most frequent literal, it may happen that two or more literals have the same frequency of occurrence. In this cases another heuristics could be applied. We construct temporary terms as candidates for implicants by adding different literals to the previously selected one(s). From them we select those terms which are implicants. This will prevent a useless term prolongation. If there are more or none terms that are implicants, the third selection method takes place. For each term we count the number of yet uncovered 1-minterms that could be covered by this term and we select the term with maximal count. In other words, we add a literal that will prevent covering the least of yet uncovered 1-minterms. When there are still more literals to choose from, we select one randomly.

A certain drawback of this algorithm is that it is greedy. Thus, once a literal is selected, it's kept till the end of the term generation. Therefore the obtained implicants need not be prime. In other words, we have to check whether some literals can be removed without any harm. Here the second part of the PI generation algorithm, called sequential search, finds its application.

3.2 Sequential Search

To check the results of the previous search, we may try and systematically remove from each term all literals one by one starting from the random position. If the hypercube obtained after removal of one literal does not cover any 0-minterm, we make the removal permanent. If, on the contrary, some 0-minterm is covered, we put the literal back and proceed to the next literal. After removing all removable literals we obtain one prime implicant covering the original term. This algorithm is also greedy.

The sequential search will not reduce the number of product terms, but it reduces the number of literals in the terms by approximately 25% (experimental result). After reducing the number of literals in the terms (and therefore extending the range they cover) some implicants may absorb others. Hence solving the covering problem is desirable. Yet, this situation doesn't occur very often.

There is also a possibility to use an exhaustive search algorithm instead of sequential search. More PIs are then generated from the terms and solving the covering problem is thus necessary. This method gives slightly better results than the previous one. However, it can be used only for small or highly undefined functions, where the number of literals in the implicants obtained from the previous search is small, because the complexity of this search algorithm is basically exponential.

Note: The sequential search and exhaustive search algorithms are also applicable for minimization without preprocessing by the coverage-directed search algorithm. However, the minimization process is then much slower and less effective (see example).

3.3 Solution of the Covering Problem

The covering problem is solved by an algorithm similar to CD-search. First, we prefer implicants covering minterms covered by the lowest number of other implicants. If there are more of such implicants, we select implicants covering the highest number of yet uncovered 1-minterms. From these primes we select the "shortest" ones – terms constructed of the least of literals. When still more primes could be selected, we select one randomly.

3.4 Repetitive Minimization

The minimization process consists of three above-mentioned stages (CD-search, Sequential search and Covering problem solution). The results of all these methods more or less depend on random events – when there are more possibilities to choose and no selection criterion is available, a random number generator is used. This means, that the results are not determined by the input function (as in ESPRESSO), but they are coincidental. Thus, we can improve the quality of solution by repeating the whole minimization process several times and choosing the best result judging by some criterion (e.g. total number of literals). Or, the program could run in an infinite loop, recording the best result found so far, and the program is stopped manually when the solution complies with some criterion (like e.g. the total number of literals does not exceed a given limit or the quality of solution does not change during a given number of iterations). The practical experience shows that in many cases the result converges to the minimal one. The results from one-pass runs and 200-pass runs are compared in the comparative example.

4 Illustrative Examples

In this section we show several simple examples illustrating the three methods.

4.1. CD-Search Example

Let's have a partially defined boolean logic function of ten input variables and ten defined minterms given by a truth table. Input variables are named x_0..x_9. The 1-minterms are highlighted.

```
var: 0123456789
0.   0000000010 1
1.   1000111011 1
2.   0000011001 1
3.   1111011000 0
4.   1011001100 0
5.   1111000100 1
6.   0100010100 0
7.   0011011011 0
8.   0010111100 1
9.   1110111000 1
```

As the first step of minimization we count the numbers of literals in the input table. The "0"-line and "1"-line give counts of x_n' and x_n literals respectively. In this table we select the most frequent literal.

```
var: 0123456789
0:   3435322444
1:   3231344222
```

The most frequent literal is x_3' with five occurrences. This literal describes a function that is a superset of an implicant. This literal alone is not an implicant yet because it covers the 6th minterm in the original function. Hence another literal must be added. When searching for the next literal, we can reduce the scope of our search by suppressing 1-minterms containing the selected literal with the opposite polarity. An implicant containing a literal x_3' cannot cover the 5th minterm (because it contains the x_3 literal). Thus, we temporarily suppress this minterm. In the remaining 1-minterms we find the most frequent literal:

```
var: 0123456789
0.   0000000010 1
1.   1000111011 1
2.   0000011001 1
3.   1111011000 0
4.   1011001100 0
5.   1111000100 1
6.   0100010100 0
7.   0011011011 0
8.   0010111100 1
9.   1110111000 1

var: 0123456789
0:   343-211433
1:   212-344122
```

As there are several literals with maximal frequency of occurrence (x_1', x_5, x_6, x_7'), the second selection criterion must be applied. We use them to create four product terms x_3'x_1', x_3'x_5, x_3'x_6, x_3'x_7'. First they are checked if some of them are already implicants. The term x_3'x_5 is not an implicant (it covers the 6th minterm), so it is discarded. Now one of the remaining terms representing implicants must be chosen. We choose a term, which covers the maximum of yet uncovered (yellow) 1-minterms. Unfortunately, all these implicants cover four 1-minterms. So one of them is selected randomly – e.g. x_3'x_6. This implicant is stored and the search continues. For the future search we omit minterms covered by this implicant (blue) and the most frequent literal is selected:

```
var: 0123456789
0.   0000000010 1
1.   1000111011 1
2.   0000011001 1
3.   1111011000 0
4.   1011001100 0
5.   1111000100 1
6.   0100010100 0
7.   0011011011 0
8.   0010111100 1
9.   1110111000 1

var: 0123456789
0:   1111222112
1:   1111000110
```

We have three equal possibilities, so we choose one randomly – e.g. x_5'. When we add the x_6' literal we have an implicant covering remaining 1-minterms. The resulting form of the function is x_3'x_6+ x_5'x_6'.

4.2 Sequential Search Example

We have the function from the previous example:

```
var: 0123456789
0.   0000000010 1
1.   1000111011 1
2.   0000011001 1
3.   1111011000 0
4.   1011001100 0
5.   1111000100 1
6.   0100010100 0
7.   0011011011 0
8.   0010111100 1
9.   1110111000 1
```

Let us suppose that a function

$x_1'x_3'+x_1x_5'x_7+x_3'x_4$

is an input for sequential search algorithm. We process all product terms, the order is not significant. We start with a term $x_1'x_3'$ and try to remove one literal. We remove x_1' and compare the remaining term (x_3') with all the 0-minterms. We'll find that it collides with the 6th minterm and thus x_1' cannot be removed. We continue with x_3' literal. It cannot be removed either, because the remaining term covers 4th and 7th minterm. This term cannot be reduced any more and thus it is a prime implicant. Now we try to reduce a second term ($x_1x_5'x_7$). We can find that x_7 literal can be removed without covering any 0-minterm, hence the original term was not prime. Thus it is simplified to x_1x_5'. Likewise, we find that the $x_3'x_4$ term can be reduced to x_4. The function is reduced to $x_1'x_3'+x_1x_5'+x_4$. This is a sum of prime implicants.

4.3 Covering Problem Solution Example

We solve the covering problem visualized by a following table:

```
       abcdefgh
1. (3) -X-X-X--
2. (4) XX--X-X-
3. (3) -XX-----
4. (2) X----X-X
5. (4) ---X--XX
```

The lines represent terms entering the algorithm, in the parentheses are the numbers of literals of appropriate terms and the columns represent 1-minterms, which must be covered.

First, we find the implicants covering minterms covered by the lowest number of other implicants. These are the 2nd and the 3rd implicants, which are necessary for covering the c- and e-minterms. Thus we record them and delete from the search set. The covered minterms are removed too.

```
       abcdefgh
1. (3) -X-X-X--
2. (4) XX--X-X-
3. (3) -XX--X--
4. (2) X----X-X
5. (4) ---X--XX
```

All the remaining 1-minterms are covered by two implicants and all implicants cover two yet uncovered minterms. So the shortest one is selected – the 4th implicant with 2 literals.

```
       abcdefgh
1. (3) -X-X-X--
2. (4) XX--X-X-
3. (3) -XX--X--
4. (2) X----X-X
5. (4) ---X--XX
```

Now, the d-minterm must be covered. We select the 1st implicant, because it is shorter than the 5th one. The 5th term could be omitted and all 1-minterms will be covered.

```
       abcdefgh
1. (3) -X-X-X--
2. (4) XX--X-X-
3. (3) -XX--X--
4. (2) X----X-X
5. (4) ---X--XX
```

5 Experimental Results

An extensive experimental work was done to evaluate the efficiency of the proposed algorithm, especially in the context of problems of large dimensions. The efficiency was compared above all with ESPRESSO. The problems solved were above the MCNC benchmarks. As the size of these benchmark problems is relatively small, some larger problems had to be prepared.

The proposed algorithm was programmed in Borland C++ Builder and tested under MS Windows NT. The processor used was a Celeron 433 MHz with 160 MB RAM, the runtime was measured in seconds. The quality of the results was measured by two parameters: number of product terms (implicants) and total number of literals. Although every implementation basis requires different evaluation criteria, these two figures are good representatives of the overall complexity of the solution obtained.

5.1 Comparative Example

First a small example with 10 input variables and 20 care terms, where also the true minimization by Quine-McCluskey algorithm could be used, was solved by different methods. All methods gave the same results (minimal disjunctive form). The runtimes in seconds were the following: sequential search 0.04, exhaustive search 0.27, CD-search 0.03, Espresso 0.03 Quine-McCluskey 438.32.

This example shows the ineffectiveness of the Quine-McCluskey's algorithm. The computing time is unacceptable and for more input variables this algorithm cannot be used. Exhaustive search algorithm is also very complex and for more than 20 input variables cannot be used either.

5.2 Comparison of Coverage-Directed Search and Espresso

The following set of examples show the advantages of the CD-search algorithm. Problems with large number of input variables and minterms were solved. The truth tables of single-output functions were generated by random number generator, for which only the number of input variables and number of care minterms in the truth table were specified. The selection of care minterms and the assignment of output values were made randomly with on-set and off-set kept approximately at the same size. For every size of the input table were generated 10 samples and the average was taken. The results are shown in Tab.1 and Tab.2. The first row of every cell contains the CD-search results, the second row shows ESPRESSO results. The entry format is: "time in seconds/ #of implicants/ #of literals". Yellow cells show the cases where CD-search algorithm gave in the average results with less or equal number of literals than ESPRESSO. The first table shows the results of Repetitive search with 200 iterations, the second table shows the results of only one iteration.

Number of defined minterms	Number of input variables														
	20	40	60	80	100	120	140	160	180	200	220	240	260	280	300
20	0.27/2/7 0.15/2/6	0.39/2/7 0.20/2/6	0.44/2/5 0.23/2/5	0.58/2/5 0.43/2/5	0.70/2/5 0.46/1/4	0.79/2/4 0.61/1/4	0.85/1/4 0.69/1/4	1.22/2/5 1.21/2/4	1.20/2/4 1.29/2/4	1.04/1/4 1.62/1/4	1.62/1/4 1.75/1/4	1.09/2/4 2.63/1/4	1.84/2/4 2.75/2/4	1.70/1/4 3.30/1/4	1.99/1/4 3.37/1/4
40	0.68/5/17 0.19/4/14	1.01/3/14 0.34/3/13	1.29/3/12 0.66/4/11	1.61/3/11 0.91/3/11	1.89/3/11 1.39/3/10	2.27/3/9 1.60/3/9	2.49/3/9 2.10/3/9	2.92/3/9 2.86/3/9	3.29/2/8 3.80/3/9	3.74/3/8 4.68/3/9	3.85/2/8 5.53/3/9	4.18/2/8 6.67/2/8	5.23/2/8 7.95/2/8	5.95/3/8 9.70/2/8	6.54/2/8 10.40/3/9
60	1.63/7/27 0.21/6/23	2.27/5/21 0.66/5/20	2.74/5/19 1.02/4/17	3.66/4/17 1.60/4/17	4.46/4/15 2.54/4/15	4.65/4/15 3.29/4/15	5.88/4/15 3.91/4/14	5.72/4/14 5.85/4/14	7.30/3/14 6.88/4/14	7.60/3/13 9.40/4/14	8.17/3/13 11.22/4/14	8.81/3/12 13.35/4/13	10.53/3/12 15.02/3/13	12.16/3/13 18.44/4/14	10.88/3/12 19.71/3/14
80	2.54/8/35 0.31/7/31	4.06/7/30 0.94/6/27	4.84/6/27 1.55/6/23	6.59/5/23 2.87/5/22	7.36/5/22 4.36/5/21	8.77/5/21 6.01/5/21	10.27/5/21 9.57/5/21	12.47/5/20 10.06/5/20	12.75/5/20 14.28/5/21	14.04/5/19 15.82/5/21	15.15/4/17 19.30/5/20	22.80/4/17 23.02/4/18	26.88/4/18 25.91/4/19	28.38/4/16 30.17/4/17	25.67/4/17 33.18/4/17
100	4.62/11/50 0.37/9/44	5.84/8/36 1.12/8/33	7.91/7/33 2.15/7/31	9.76/7/30 4.54/6/29	11.38/6/28 5.78/6/29	13.57/6/27 8.25/6/26	15.13/6/26 11.57/6/26	16.75/6/25 14.40/6/26	19.53/5/24 20.12/6/24	22.51/5/24 21.78/6/25	23.88/5/24 27.96/6/25	31.51/5/23 31.78/5/24	28.60/5/24 35.69/5/24	31.63/5/21 42.99/5/24	31.36/5/22 47.46/5/22
120	5.05/13/60 0.50/11/51	7.83/10/47 1.69/9/43	10.42/9/41 3.27/8/36	12.81/8/37 5.68/8/36	15.26/7/35 8.12/7/34	17.23/7/33 12.43/7/32	20.45/7/31 17.66/7/33	25.13/7/31 19.41/7/30	28.00/6/30 27.60/6/30	29.24/6/29 34.57/7/32	32.28/6/29 38.23/6/30	35.96/6/28 45.51/6/29	37.73/6/27 55.03/6/29	39.67/6/26 63.59/6/28	43.72/6/27 63.22/6/28
140	7.43/15/77 0.52/13/63	10.77/11/55 2.12/10/51	14.53/10/50 4.51/10/45	17.02/9/45 7.61/9/43	20.95/9/42 11.04/9/42	25.32/8/39 18.00/8/39	28.31/8/39 23.41/8/38	31.46/7/36 25.30/8/39	35.05/7/35 35.57/8/37	39.34/7/33 37.26/7/36	43.02/7/33 45.64/8/36	50.58/7/34 60.40/8/36	53.69/7/33 72.49/7/33	54.54/7/33 75.58/7/36	56.78/7/31 80.75/7/33
160	9.33/17/87 0.72/14/73	13.96/12/63 2.83/12/59	19.42/11/56 5.99/11/52	24.50/10/53 11.07/10/48	28.79/10/49 16.00/10/47	33.64/9/46 26.23/9/46	37.55/9/45 31.87/9/44	43.58/9/43 39.72/9/43	46.64/8/41 50.55/9/43	52.11/8/40 56.26/8/42	57.21/8/39 69.96/9/42	71.56/8/39 87.40/8/40	74.34/7/37 94.79/8/40	83.44/8/38 105.66/8/41	100.44/7/37 114.47/8/39
180	13.84/19/102 1.05/16/83	19.96/15/76 3.07/13/64	25.44/13/68 6.61/12/61	32.98/11/60 13.16/11/58	37.59/11/57 18.63/11/55	42.86/10/53 28.29/10/51	47.69/9/48 31.59/10/51	57.58/10/49 44.71/10/49	65.49/9/48 53.07/10/48	68.21/9/46 63.32/9/47	77.08/9/45 79.32/9/46	87.89/9/46 96.70/9/46	91.01/8/43 118.29/9/46	96.34/8/43 128.90/9/45	102.32/8/42 131.19/9/44
200	14.80/21/114 1.01/18/94	22.58/16/86 4.24/14/74	31.03/14/73 8.84/13/68	41.79/12/66 17.50/12/63	45.80/11/62 23.34/11/59	52.76/11/60 33.93/11/57	57.63/11/56 44.29/11/58	69.43/10/55 54.24/11/56	76.10/10/54 67.52/10/54	78.86/10/52 84.97/10/53	87.60/10/50 104.74/10/54	102.59/10/51 112.62/10/52	102.94/9/46 129.48/10/52	114.08/9/48 154.88/10/50	120.31/9/47 196.61/10/49

Tab. 1: *Repetitve search with 200 iterations results*

Number of defined minterms	Number of input variables														
	20	40	60	80	100	120	140	160	180	200	220	240	260	280	300
20	0.00/3/9 0.15/2/6	0.00/2/5 0.20/2/6	0.00/2/4 0.23/2/5	0.00/2/6 0.43/2/5	0.00/2/6 0.46/1/4	0.00/2/6 0.61/1/4	0.00/2/5 0.69/1/4	0.01/2/5 1.21/2/4	0.00/2/5 1.29/2/4	0.00/2/4 1.62/1/4	0.01/2/7 1.75/1/4	0.00/2/3 2.63/1/4	0.01/2/6 2.75/2/4	0.01/3/8 3.30/1/4	0.01/2/4 3.37/1/4
40	0.00/5/20 0.19/4/14	0.00/4/12 0.34/3/13	0.01/4/12 0.66/4/11	0.01/3/11 0.91/3/11	0.01/4/15 1.39/3/10	0.01/3/11 1.60/3/9	0.01/3/11 2.10/3/9	0.02/4/15 2.86/3/9	0.01/3/8 3.80/3/9	0.02/3/10 4.68/3/9	0.02/4/11 5.53/3/9	0.02/3/9 6.67/2/8	0.03/3/11 7.95/2/8	0.02/3/11 9.70/2/8	0.03/3/11 10.40/3/9
60	0.01/8/32 0.21/6/23	0.01/6/20 0.66/5/20	0.02/6/23 1.02/4/17	0.02/5/20 1.60/4/17	0.02/4/15 2.54/4/15	0.03/5/22 3.29/4/15	0.03/4/16 3.91/4/14	0.03/4/16 5.85/4/14	0.04/5/20 6.88/4/14	0.04/5/16 9.40/4/14	0.04/4/14 11.22/4/14	0.04/4/14 13.35/4/13	0.06/4/15 15.02/3/13	0.07/6/25 18.44/4/14	0.05/4/17 19.71/3/14
80	0.01/9/39 0.31/7/31	0.02/8/34 0.94/6/27	0.03/8/35 1.55/6/23	0.03/7/31 2.87/5/22	0.04/6/24 4.36/5/21	0.05/7/30 6.01/5/21	0.05/6/28 9.57/5/21	0.05/5/23 10.06/5/20	0.05/5/19 14.28/5/21	0.07/6/29 15.82/5/21	0.08/6/25 19.30/5/20	0.11/4/18 23.02/4/18	0.13/5/20 25.91/4/19	0.14/5/18 30.17/4/17	0.13/5/21 33.18/4/17
100	0.03/14/69 0.37/9/44	0.02/8/37 1.12/8/33	0.04/9/40 2.15/7/31	0.04/7/33 4.54/6/29	0.06/6/25 5.78/6/29	0.06/7/39 8.25/6/26	0.07/6/25 11.57/6/26	0.07/5/22 14.40/6/26	0.09/6/28 20.12/6/24	0.10/6/26 21.78/6/25	0.12/6/29 27.96/6/25	0.16/8/35 31.78/5/24	0.14/6/31 35.69/5/24	0.15/5/21 42.99/5/24	0.17/6/28 47.46/5/22
120	0.02/14/67 0.50/11/51	0.03/11/53 1.69/9/43	0.05/10/53 3.27/8/36	0.06/9/47 5.68/8/36	0.07/7/33 8.12/7/34	0.08/10/49 12.43/7/32	0.10/8/39 17.66/7/33	0.13/8/36 19.41/7/30	0.13/7/33 27.60/6/30	0.14/8/34 34.57/7/32	0.16/8/39 38.23/6/30	0.18/8/35 45.51/6/29	0.18/8/35 55.03/6/29	0.18/6/27 63.59/6/28	0.23/6/29 63.22/6/28
140	0.04/16/77 0.52/13/63	0.05/14/76 2.12/10/51	0.07/10/47 4.51/10/45	0.08/9/42 7.61/9/43	0.10/10/50 11.04/9/42	0.14/11/54 18.00/8/39	0.15/10/49 23.41/8/38	0.16/9/44 25.30/8/39	0.19/8/39 35.57/8/37	0.22/9/42 37.26/7/36	0.22/9/50 45.64/8/36	0.24/8/43 60.40/8/36	0.28/8/43 72.49/7/33	0.29/9/44 75.58/7/36	0.31/9/50 80.75/7/33
160	0.04/18/94 0.72/14/73	0.06/12/63 2.83/12/59	0.10/13/61 5.99/11/52	0.12/14/74 11.07/10/48	0.16/12/70 16.00/10/47	0.17/11/51 26.23/9/46	0.18/10/51 31.87/9/44	0.24/11/50 39.72/9/43	0.22/10/54 50.55/9/43	0.29/10/53 56.26/8/42	0.31/10/48 69.96/9/42	0.34/9/49 87.40/8/40	0.38/11/61 94.79/8/40	0.42/10/52 105.66/8/41	0.52/9/41 114.47/8/39
180	0.07/21/113 1.05/16/83	0.10/18/99 3.07/13/64	0.13/15/80 6.61/12/61	0.17/15/84 13.16/11/58	0.19/12/61 18.63/11/55	0.21/11/59 28.29/10/51	0.23/12/66 31.59/10/51	0.31/12/68 44.71/10/49	0.32/11/56 53.07/10/48	0.30/11/60 63.32/9/47	0.40/11/54 79.32/9/46	0.46/12/65 96.70/9/46	0.45/10/54 118.29/9/46	0.52/11/61 128.90/9/45	0.53/12/61 131.19/9/44
200	0.08/24/133 1.01/18/94	0.10/17/91 4.24/14/74	0.15/17/90 8.84/13/68	0.20/15/77 17.50/12/63	0.25/15/89 23.34/11/59	0.23/12/63 33.93/11/57	0.26/11/60 44.29/11/58	0.37/13/72 54.24/11/56	0.41/14/78 67.52/10/54	0.39/12/63 84.97/10/53	0.45/12/63 104.74/10/54	0.50/12/66 112.62/10/52	0.51/10/52 129.48/10/52	0.58/14/75 154.88/10/50	0.62/11/61 196.61/10/49

Tab. 2: *CD-Search, 1 iteration results*

The dependency of the run times on the two parameters is visualized by the graphs in Fig. 1. It should be noted that the vertical scales are different, because ESPRESSO is in all cases much slower.

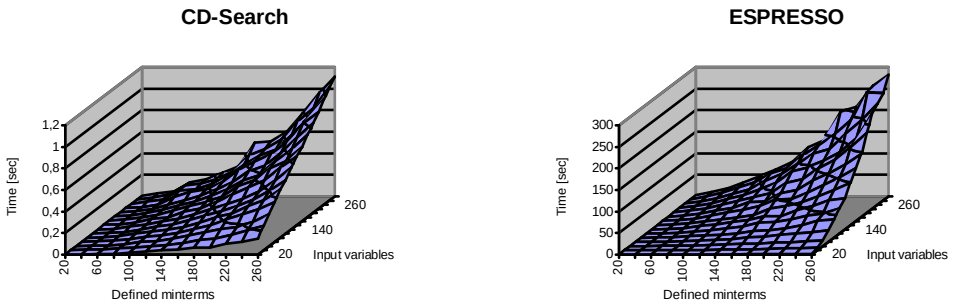

Fig. 1. Dependency of runtimes on the numbers of variables and care minterms

For problems with more input variables the difference was still bigger. A problem with 1000 variables and 1000 vectors was solved by C-D in a couple of minutes. ESPRESSO did not solve the problem at all.

5.3 Standard Benchmarks

A set of standard Berkeley benchmarks [7] was also used for comparison. All these benchmarks use a relatively small number of inputs. Hence the runtimes needed by the C-D search were longer than those of ESPRESSO.

6 Conclusions

A method of single-output Boolean function minimization applicable to problems with large dimensions and large number of don't care states has been presented. Although the PI generation method is rather straightforward, the results achieved are comparable with ESPRESSO and the runtimes are shorter. For large problems with several hundreds of variables the program beats ESPRESSO both in minimality of the result and in the runtime.

Acknowledgment

The research was in part supported by the grant of the Czech Grant Agency GACR 102/99/1017.

References

[1] HACHTEL, G.D. - SOMENZI, F.: Logic synthesis and verification algorithms. Boston, MA, Kluwer Academic Publishers, 1996, 564 pp.
[2] BRAYTON, R.K. et al.: Logic minimization algorithms for VLSI synthesis. Boston, MA, Kluwer Academic Publishers, 1984
[3] COUDERT, O. - MADRE, J.C.: Implicit and incremental computation of primes and essential primes of Boolean functions. In Proc. of the Design Automation Conf. (Anaheim, CA, June 1992), pp. 36-39
[4] COUDERT, O. - MADRE, J.C.: New ideas for solving covering problems. In Proc. of the Design Automation Conf. (San Francisco, CA, June 1995), pp. 641-646
[5] McCLUSKEY, E.J.: Minimization of Boolean functions. The Bell System Technical Journal, 35, No. 5, Nov. 1956, pp. 1417-1444
[6] QUINE, W.V.: The problem of simplifying truth functions. Amer. Math. Monthly, 59, No. 8, 1952, pp. 521-531
[7] ftp://ftp.mcnc.org/pub/benchmark/Benchmark_dirs/LGSynth93/testcases/pla/