

# ***A New Heuristic for DSOP Minimization***

***V. Ciriani***

Università degli Studi di Milano

***A. Bernasconi, F. Luccio, L. Pagli***

Università di Pisa

# *Disjoint Sum of Products (DSOP)*

- ❖ Given a Boolean function

$$f : \{0,1\}^n \longrightarrow \{0,1\}$$

- ❖ A **DSOP** is a sum (**OR**) of products (**ANDs**) such that no two products cover the same 1 of **f**
- ❖ **DSOP** represents a set of **non intersecting cubes** covering the points of **f**
- ❖ Problem: minimize **|DSOP|**, i.e., find a DSOP with a **minimal number of products**

# Motivations

- ❖ DSOP minimization is relevant in the area of digital circuits:
  - ◆ DSOPs are used as a starting point for the synthesis of **Exclusive-Or-Sum-Of-Products (EPSOPs)**
  - ◆ and for calculating the **spectra of Boolean functions**
- ❖ Heuristic strategies for cube selection have been proposed
  - ◆ working on explicit product expressions (FSC93,ST02)
  - ◆ or on a BDD representation of the function (FD02)

# Example: SOP vs DSOP

f		$x_3 x_4$			
		00	01	11	10
$x_1 x_2$	00	1	1	0	0
	01	1	1	1	1
	11	0	1	1	1
	10	0	0	1	1

**SOP**

f		$x_3 x_4$			
		00	01	11	10
$x_1 x_2$	00	1	1	0	0
	01	1	1	1	1
	11	0	1	1	1
	10	0	0	1	1

**DSOP**

## *Example: SOP vs DSOP*

$$f = X_1X_2 + X_3X_4 + \dots + X_{n-1}X_n$$

**Minimal SOP:**  $|SOP| = n/2$

**Minimal DSOP:**  $|DSOP| = 2^{n/2} - 1$

# *Complexity of the problem*

- ❖ SOP minimization  $\approx$  set covering
- ❖ DSOP minimization  $\approx$  minimal exact cover
- ❖ Minimal exact cover is **NP-hard**

# Intersections

❖ A product  $p = x_1x_2\dots x_k$  represents a cube of dimension  $d(p) = n-k$ , i.e., a cube of  $2^{n-k}$  points

❖ The **intersection** of two cubes is obtained as

$$p = p_1 \square p_2$$

❖  $p$  is void iff there is a literal in  $p_1$  that is complemented in  $p_2$

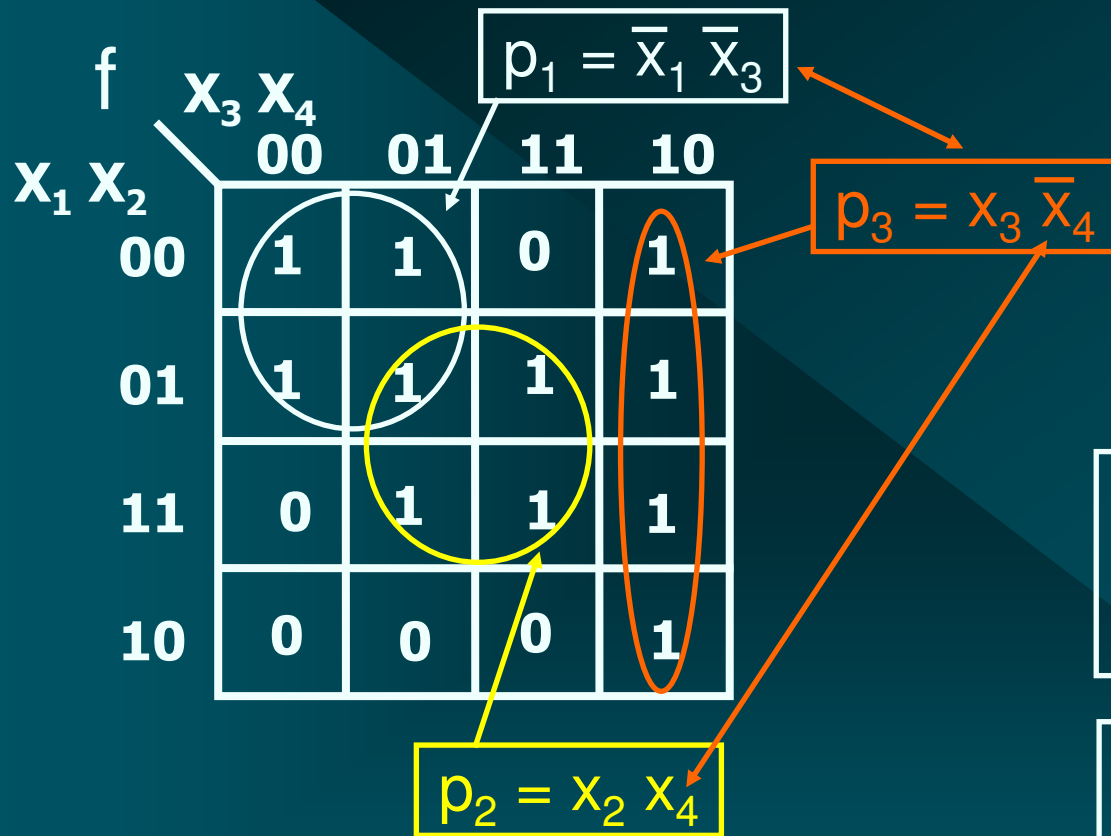
❖ Otherwise  $p$  is a cube of dimension  $r$ , with

$$r = n - (k_1 + k_2 - c)$$

$k_1, k_2$ : number of literals in  $p_1$  and  $p_2$

$c$ : number of their common literals

# Example



$$r = n - (k_1 + k_2 - c)$$

$$= 4 - (2 + 2 - 0)$$

$p_1 \square p_2$  is a cube of  $2^0$  points

## *A simple case*

- ❖ Let  $p_1$  and  $p_2$  partially overlap
- ❖ The set of points of  $p_2 \setminus p_1$  can be covered by a set of  $k_1 - c$  disjoint cubes of dimensions  $r, r+1, \dots, n - k_2 - 1$
- ❖ If  $p_1$  is selected into a DSOP,  $p_2$  must be discarded and the points of  $p_2 \setminus p_1$  must be covered with at least  $k_1 - c$  disjoint cubes instead of one (the single  $p_2$ )

# Example

2 cubes

f		$x_3 x_4$		$p_1$	
		00	01	11	10
$x_1 x_2$	00	1	1	0	0
	01	1	1	1	0
11	0	1	1	0	
10	0	0	0	0	

$p_2$

3 cubes (1 extra cube)

f		$x_3 x_4$		$p_1$	
		00	01	11	10
$x_1 x_2$	00	1	1	0	0
	01	1	1	1	0
11	0	1	1	0	
10	0	0	0	0	

$p_2 \setminus p_1$

# *A simple case*

- ❖ Then  $k_1 - c - 1$  is the number of extra cubes required by the DSOP
- ❖ If the function  $f$  can be represented by a SOP containing only  $p_1$  and  $p_2$ , the selection of  $p_1$  into a DSOP requires a total of  $k_1 - c + 1$  cubes

Special case: If  $k_1 - c = 1$

1.  $p = p_1 \square p_2$  covers exactly one half of the points of  $p_2$
2.  $p_2 \setminus p_1$  is also a cube

# *The general case*

- ❖ The general situation can be not that simple
- ❖ The starting SOP can contain a collection of cubes overlapping in groups
- ❖ The **weight** of a cube  $p_i$  is the **minimum number of extra cubes** that the selection of  $p_i$  would induce in **all** the cubes intersecting  $p_i$

# The “weight” of a cube

❖ Let product  $p$  of  $k$  literals intersect  $t$  products  $p_1, \dots, p_t$ , such that  $p$  and  $p_j$  have  $c_j$  common literals

❖ The weight of  $p$  relative to  $p_j$  is

$$w(p/p_j) = k - c_j - 1$$

❖ and the weight of  $p_j$  is

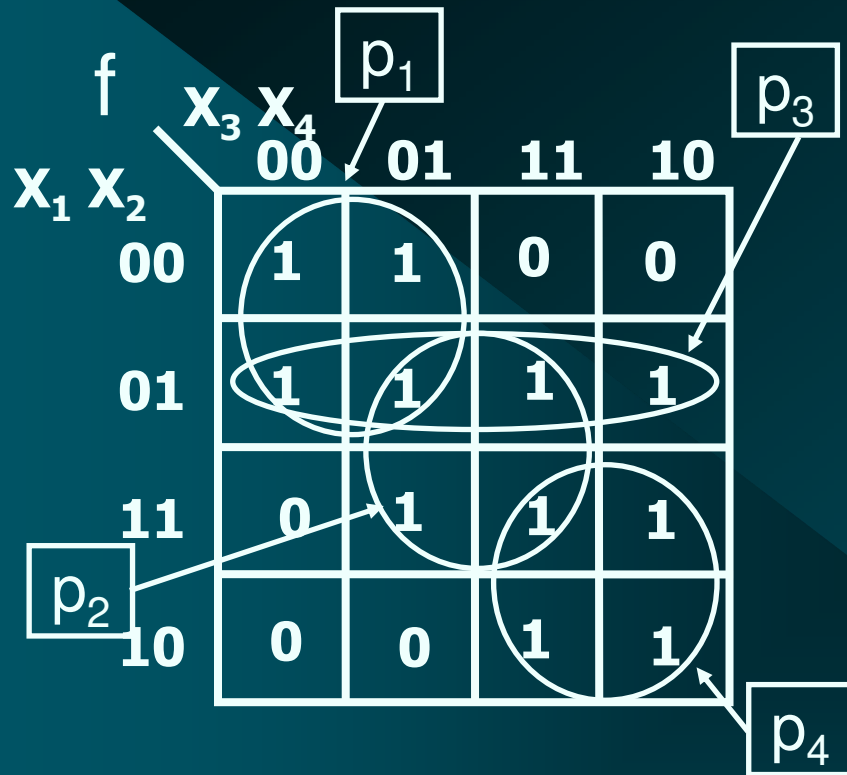
$$w(p) = \sum_{j=1}^t w(p/p_j)$$

❖ If  $p$  does not intersect any product,  $w(p) = -1$

## *The “weight” of a cube*

When  $p$  intersects  $p_j$ , the weight of  $p$  relative to  $p_j$  is the *minimum number of additional products* that we would have in the cover keeping  $p$  and covering  $p \setminus p_j$  with non-overlapping products

# Example



$w(p_1/p_2) = 1$ : selecting  $p_1$  in a DSOP would require covering the remaining three points of  $p_2$  with at least two disjoint cubes

$w(p_1/p_3) = 0$ : the residual two points of  $p_3$  can be covered with one cube

$\rightarrow w(p_1) = 1$

$w(p_2/p_1) = 1$ ;  $w(p_2/p_3) = 0$ ;  $w(p_2/p_4) = 1 \rightarrow w(p_2) = 2$

$w(p_3/p_1) = 0$ ;  $w(p_3/p_2) = 0 \rightarrow w(p_3) = 0$

$w(p_4/p_2) = 1 \rightarrow w(p_4) = 1$

# *A family of heuristic algorithms*

- ❖ Heuristics for DSOP that uses **four basic procedures**
  - ❖ **Explicit representation** of cubes
1. **BUILD-SOP(C,P)** builds a minimal SOP **P** from the cover **C** of **f** (**Espresso** heuristic)
  2. **WEIGHT(P)** builds the weights for a set of cubes **P**

# *A family of heuristic algorithms*

3. **SORT(P)** sorts a set **P** of weighted cubes

Two versions:

- ◆ the cubes are ordered for **decreasing dimension** and, if the dimension is the same, for **increasing weight**
- ◆ the cubes are ordered for **increasing weight** and, if the weight is the same, for **decreasing dimension**

If two or more cubes have same weight and same dimension, their order is arbitrary

# *A family of heuristic algorithms*

4. **BREAK**( $q, Q$ ) works on  $q = p_1/p_2$ , and builds an arbitrary minimal set  $Q$  of disjoint cubes covering  $q$

BREAK is the procedure suggested in HCO74 and Sa99 as **DISJOINT-SHARP**

# *A family of heuristic algorithms*

- ❖ We use four sets of cubes: **C**, **P**, **B**, **D**
  - ◆ **C** = {the cubes still to be covered with a DSOP}
  - ◆ **P** = {the cubes of a SOP under processing}
  - ◆ **B** = {cubes produced by BREAK as fragmentation of cubes of **P**}
  - ◆ **D** = {the cubes in the building DSOP solution}
- ◆ At the beginning **C** contains the cubes defining **f**, while **P**, **B**, **D** are empty

# A family of heuristic algorithms

```
algorithm DSOP( $C, D$ )
  while ( $C \neq \emptyset$ ) {
    BUILD-SOP( $C, P$ ); WEIGHT( $P$ ); SORT( $P$ );
    while ( $P \neq \emptyset$ ) {
      let  $p$  be the first element of  $P$ ;
      move  $p$  from  $P$  to  $D$ ;
       $\forall q \in P : p \cap q \neq \emptyset$  {
        remove  $q$  from  $P$ ;
        BREAK( $q \setminus p, Q$ );  $B = B \cup Q$ ; *OPT* }
       $\forall q \in B : p \cap q \neq \emptyset$  {
        remove  $q$  from  $B$ ;
        BREAK( $q \setminus p, Q$ );  $B = B \cup Q$ ; } }
     $C = B$ ;  $B = \emptyset$ ; }
```

# A family of heuristic algorithms

\*OPT\* (optional):

1. recomputes the weight of each cube  $r \in P$  such that  $r \cap q \neq \emptyset$
  2. checks the fragments of  $q$  with which  $r$  now intersects
  3. sorts  $P$  again
- ❖ More sophisticated optimization procedures, as for instance inserting all broken cubes into  $P$  and ordering them immediately, do not seem to provide better quality results
  - ❖ **BUILD-SOP**: the idea of re-synthesizing the remaining cubes seems to be crucial for obtaining compact DSOPs

# *A family of heuristic algorithms*

## Complexity

The complexity of the algorithm is **polynomial** in the size of the output, i.e., in the number of products of the computed DSOP form

# *Experimental results*

- ❖ ESPRESSO benchmark suite
- ❖ 1.8 GHz PowerPC with 1 GB of RAM
- ❖ Three different variants of our heuristic

# *Experimental results*

## DSOP-1

is the simplest, and computationally fastest: it consists in just the basic algorithm, **without** the optional optimization phase \*OPT\*

## DSOP-2

**with** the optional optimization phase \*OPT\*: after a cube **p** has been chosen, the algorithm updates the weight of all cubes **q**  $\square$  **P** intersecting **p**, and then sorts the cubes in **P** again

# *Experimental results*

- ❖ whenever a cube  $p$  is moved from  $P$  to  $D$ , all cubes  $q$  intersecting  $p$  are fragmented and removed from  $P$

## Disadvantage:

- ❖ Hence, the fragments, even the big ones, are “out of the game” and cannot participate in the construction of the DSOP  $D$  until  $P = \emptyset$  and a new SOP covering all fragments in  $B$  is computed
- ❖ Thus, small cubes in  $P$  could be selected first, possibly damaging the size of the final DSOP

# *Experimental results*

## DSOP-3

1. As usual, whenever  $p \in P$  is selected and moved to  $D$ , each cube  $q$  intersecting  $p$  is fragmented and moved to  $B$
2. But now all cubes  $r \in P$  intersecting  $q$  are moved to  $B$  as well

in this way, we prevent small and high weight cubes of  $P$  from generating high fragmentations of big fragments of low weight cubes already in  $B$

# Experimental results

- ❖ For each variant, we run both versions of **SORT**:
  - ◆ we first ordered the cubes for decreasing dimension and, for equal dimension, for increasing weight ( $d/w$ )
  - ◆ then we ordered the cubes for increasing weight and, for equal weight, for decreasing dimension ( $w/d$ )
- ❖ Multi-output functions:
  - ◆ we have considered each output separately, but the minimization phase with **Espresso** is performed in a multi-output way
  - ◆ common disjoint cubes of different outputs are counted only once

# Comparison among different versions of the heuristic

	in	out	SOP	DSOP-1		DSOP-2		DSOP-3	
				d/w (time)	w/d (time)	d/w (time)	w/d (time)	d/w (time)	w/d (time)
accpla	50	69	175	1457 (11.68)	1779 (24.57)	1458 (13.42)	1717 (32.46)	<b>1190</b> (10.55)	1317 (16.80)
addm4	9	8	200	218 (0.26)	220 (0.26)	221 (0.31)	222 (0.27)	<b>214</b> (0.40)	217 (0.29)
b10	15	11	100	130 (0.31)	136 (0.32)	130 (0.33)	130 (0.33)	<b>115</b> (0.49)	120 (0.39)
b2	16	17	106	131 (0.42)	131 (0.41)	131 (0.49)	131 (0.44)	121 (0.54)	<b>120</b> (0.62)
b3	32	20	211	308 (0.78)	315 (0.73)	308 (0.88)	317 (0.77)	<b>279</b> (1.21)	321 (1.77)
bc0	26	11	179	214 (0.47)	230 (0.60)	214 (0.53)	218 (0.53)	<b>202</b> (0.68)	210 (1.18)
dist	8	5	123	135 (0.22)	138 (0.21)	135 (0.23)	138 (0.23)	<b>130</b> (0.38)	133 (0.36)
ex1010	10	10	284	828 (0.58)	855 (0.72)	<b>820</b> (0.75)	835 (0.79)	876 (1.34)	893 (1.42)
ex4	128	28	279	485 (3.23)	546 (3.18)	470 (3.27)	487 (3.14)	<b>438</b> (5.93)	551 (5.90)
ex5	8	63	74	126 (0.79)	128 (0.38)	126 (0.44)	125 (0.44)	<b>122</b> (0.80)	142 (0.77)
gary	15	11	107	134 (0.52)	139 (0.28)	134 (0.35)	131 (0.27)	<b>124</b> (0.49)	132 (0.43)
ibm	48	17	173	366 (1.23)	431 (0.69)	366 (0.59)	393 (0.64)	<b>361</b> (0.99)	416 (1.24)
in4	32	20	212	312 (1.36)	329 (0.72)	312 (0.84)	331 (0.81)	<b>280</b> (1.33)	321 (1.29)
intb	15	7	631	811 (2.03)	955 (1.79)	818 (1.61)	932 (1.83)	<b>798</b> (2.57)	1125 (3.65)
jbp	36	57	122	135 (0.57)	151 (0.35)	135 (0.26)	147 (0.36)	<b>127</b> (0.43)	128 (0.33)
mainpla	27	54	172	296 (4.67)	459 (2.86)	296 (3.00)	405 (2.47)	<b>293</b> (3.23)	387 (3.33)
misex3	14	14	690	1070 (2.72)	1132 (1.28)	1073 (1.49)	1155 (1.75)	<b>1032</b> (2.68)	1317 (4.58)
soar	83	94	353	447 (1.93)	451 (1.16)	447 (1.30)	449 (1.25)	434 (1.58)	<b>430</b> (1.41)
spla	16	46	260	359 (0.83)	363 (0.43)	359 (0.44)	363 (0.44)	<b>347</b> (0.64)	361 (0.86)
table3	14	14	175	181 (0.41)	181 (0.21)	181 (0.23)	181 (0.24)	<b>180</b> (0.33)	<b>180</b> (0.24)
table5	17	15	158	167 (0.39)	167 (0.31)	167 (0.36)	167 (0.32)	<b>161</b> (0.38)	<b>161</b> (0.28)
test3	10	35	541	1368 (1.69)	1365 (1.31)	<b>1363</b> (1.36)	1364 (1.18)	1462 (1.81)	1454 (1.84)
tial	14	8	581	943 (1.78)	1121 (2.22)	937 (1.96)	1060 (2.13)	<b>874</b> (2.98)	1371 (6.68)

# Comparison with other techniques

Bench	in	out	PLA	SOP	DSOP ESPR.	DSOP [2]	DSOP [10]	DSOP [3]	DSOP
5xp1	7	10	75	65	99	70	–	82	70
9sym	9	1	87	86	209	166	148	148	134
alu4	14	8	1028	575	3551	–	–	1545	881
b12	15	9	431	43	691	57	–	60	51
clip	9	5	167	120	359	162	–	262	140
co14	14	1	47	14	14	–	14	14	14
max1024	10	6	1024	274	775	–	–	444	334
misex1	8	7	32	12	18	15	–	34	15
misex2	25	18	29	28	29	28	–	30	28
mlp4	8	8	256	128	206	–	–	203	143
rd53	5	3	32	31	31	31	–	35	31
rd73	7	3	141	127	127	127	–	147	127
rd84	8	4	256	255	255	–	–	294	255
sym10	10	1	837	210	367	–	240	240	232
t481	16	1	481	481	2139	–	2139	1009	841
x7dn	66	15	622	538	1697	–	–	1091	812
xor5	5	1	16	16	16	–	16	16	16

[2]: sorts cubes in a SOP according to their size, and compares the largest cube with all the others, starting from the smallest ones; cubes are merged, where possible (FSC93)

[10]: exploits the property of the most binate variable in a set of cubes (ST02)

[3]: makes use of BDDs (FD02)

# *Future works*

- ❖ Evaluate the time performances of our heuristic on very large benchmarks, and compare them with other methods, especially with the one based on BDDs
- ❖ Use BDDs and symbolically perform all the operations in the heuristic
- ❖ Study the approximability properties of DSOP minimization, with the aim of designing approximation algorithms, instead of heuristics

***Thank You !!!***



[www.dti.unimi.it/~ciriani](http://www.dti.unimi.it/~ciriani)