An Efficient Heuristic Approach to Solve the Unate Covering Problem

Roberto Cordone
DEI - Politecnico di Milano
e-mail: cordone@elet.polimi.it

Fabrizio Ferrandi
DEI - Politecnico di Milano
e-mail: ferrandi@elet.polimi.it

Donatella Sciuto
DEI - Politecnico di Milano
e-mail: sciuto@elet.polimi.it

Roberto Wolfler Calvo
ISIS - Joint Research Centre (JRC) - Ispra
e-mail: roberto.wolfler@jrc.it

Abstract

The classical solving approach for two-level logic minimisation reduces the problem to a special case of unate covering and attacks the latter with a (possibly limited) branch-and-bound algorithm. We adopt this approach, but we propose a constructive heuristic algorithm that combines the use of Binary Decision Diagrams with the lagrangian relaxation. This technique permits to achieve an effective choice of the elements to include into the solution, as well as cost-related reductions of the problem and a good lower bound on the optimum. The results support the effectiveness of this approach: on a wide set of benchmark problems, the algorithm nearly always hits the optimum, and in most cases proves it to be such. On the problems whose optimum is actually unknown, the best known result is strongly improved.

1. Introduction

The classical solving approach for two-level logic minimization in the *VLSI* literature goes back to Quine's [20] and McCluskey's [17] works. It reformulates the problem as a special case of the *Unate Covering Problem* and applies algorithms conceived for the latter, or even for the more general *Binate Covering Problem*. These are a common model in most fields of Computer Science (circuit synthesis, software engineering, artificial intelligence, and so on), so that optimal solutions would be extremely precious. Unfortunately, both problems are NP-hard, and no efficient algorithm to solve them exactly is known. Conversely, heuristics can lead to good solutions in a reasonable amount of time, and yield tight bounds, by means of which even large-size problems can be solved to optimality by branch-and-bound.

This paper describes a heuristic algorithm that adopts specialised data structures, named *Zero-suppressed binary Decision Diagrams* (*ZDD*s), in order to provide a compact representation of the data [18]. The algorithm follows a greedy constructive approach: it selects one product at a time, until their sum is equivalent to the given function; then, redundant products are removed. The choice of each product is driven by a sophisticated process, exploiting the integer programming formulation of unate covering [19]. This leads to strong improvements in several fundamental aspects with respect to the classical approaches adopted in the *VLSI* literature. In particular, some well-known pruning techniques can be strengthened and generalised. The method yields a lower bound which is tighter than the classical one, so that it is often possible to prove the optimality of the solution found, or, at least, to give a better estimate of how far this is from the optimum. Experience proves that even a single run of the algorithm determines nearly optimal solutions. Yet, the whole process can be repeated, taking different choices in the intermediate steps and possibly obtaining better results.

The paper is organised as follows. Next section formally defines two-level minimisation and its relation to unate covering. Section 3 presents the approach used to solve the unate covering problem, while section 4 introduces the algorithm. Experimental results close the paper, together with some conclusions.

2. Classical solving methods

Most approaches to solving covering problems have largely followed the pioneering work of Quine and McCluskey [20] [17], in which two-level minimisation is turned into the equivalent covering problem. This leads to two subsequent bottlenecks. First, the reformulated problem can have an exponential dimension with respect to the number of variables n. In fact, prime implicants are at most $O(3^n/\sqrt{n})$, and can be at least $O(3^n/n)$ [8], while minterms, of course, vary from 0 to 2^n. Second, the covering problem is NP-hard: in the worst case, it can be solved only in exponential time with respect to the number of prime implicants and minterms.

However, the covering formulation is often redundant [10], so that the problem can be hugely reduced by means of logical remarks. The early works devoted a strong effort to the development of such reductions: partitioning, essentiality, row dominance, column dominance, Gimpel's reduction, and so on. A good survey on the subject can be found in [23].

More recent works [4] on reduction methods proposed implicit techniques, to avoid even generating redundant rows and columns. In particular, much interest has been aroused by efficient data structures, which are well suited to support the solving procedures. In the first attempts [22], minterms and prime implicants were encoded into a couple of Binary Decision Diagrams (*BDD*s), a canonical representation for boolean functions [5]. Later on, however, the use of Zero-suppressed binary Decision Diagrams (*ZDD*s), which are a canonical representation for combinational sets, has proven better [18].

Logical reductions can be iteratively applied, until they produce a stable covering matrix, called *cyclic core*. If this is empty, the essential columns found form a minimal solution. Otherwise, the problem can be solved exactly by branch-and-bound algorithm [11]. In this scheme, a high lower bound and a low upper bound on the optimal solution are extremely effective in terminating useless searches. Most algorithms in the *VLSI* literature compute lower bounds by maximal independent sets [23]. A theoretically better lower bound is presented in [11], but in practice this is worse than the classical one, apart from ill-conditioned or high-density problems. Liao and Devadas [15] show that a tighter lower bound may be obtained by formulating unate covering as an integer linear program and relaxing the integrality constraint. This bound is computationally more expensive, but unsuccessful branches are pruned at earlier stages. Liao and Devadas also present a faster technique, *dual ascent*, based on the heuristic solution of the dual problem [19]. The resulting bound is weaker than, or at most equal to the linear relaxation one. In particular, if the costs are uniform (which is the common case in *VLSI*), a simple dual ascent yields the classical independent-set based bound. In [14] a two phases approach is adopted: first the upper bound of the solution is strengthened by searching a set of promising solutions with the classical branching rules, second, when no further better solutions are expected, the maximum independent set lower bound is incrementally strengthened by adding suitable rows to the set and determining the optimum of this reduced problem.

As for heuristics and the computation of upper bounds, a simple greedy algorithm for unate covering problems was proposed by Johnson and Lovász [16] and extended by Chvátal [9] to non uniform cost problems. Improved variants, were proposed by Balas and Ho, who combined them with dual heuristics in a cutting plane exact algorithm [1]. Beasley [2] adopted several reduction procedures, in addition to a lagrangian approach similar to the one we describe in Section 3. Fisher and Kedia [13] added dual heuristics to this framework, which has been recently improved by Ceria, Nobili and Sassano [7]. Caprara, Fischetti and Toth [6] used a lagrangian heuristic coupled with a dynamic pricing scheme, in order to determine a good relaxation in short time, even in large-scale problems.

3. The proposed approach

This section presents an integer formulation of the unate covering problem and describes how to improve its lagrangian relaxation through a subgradient scheme, so as to derive useful information on the original problem. First, we introduce the formulation and the concept of lagrangian relaxation. Then, we describe the subgradient method to tighten this relaxation and the way to exploit the dual problem in order to proceed more quickly. As the process requires feasible solutions both to the primal and the dual problem, we discuss the heuristics used to build them. The following subsection deals with penalty procedures, allowing to fix or discard some of the columns, without affecting the optimum. In the end, the whole information is summed up into two heuristic rules to determine which columns are likely to be optimal, and should therefore be fixed in order to proceed.

3.1. The linear and the lagrangian relaxation

The unate covering problem (M, P, R, c) can be formulated as an integer linear programming problem [19] by associating to set P a cost coefficient vector c and a binary variable vector p, so that c_j be the cost of column j, $p_j = 1$ if column j belongs to the current solution, $p_j = 0$ otherwise.

$$\begin{aligned} &\min z = c'p \\ \text{s.t.}\quad & Ap \geq e \qquad p \in \{0,1\}^{|P|} \end{aligned} \tag{UCP}$$

where e stands for a suitable all-one vector ($e_i = 1, i = 1, 2, \ldots, |M|$) and A is the covering matrix. Vector inequalities are meant to be componentwise, so that $Ap \geq e$ represents the covering constraints $\sum_{j=1}^{|P|} a_{ij} p_j \geq 1$, for all i: each row must be covered by at least one column.

This NP-hard problem can be simplified by turning constraint $p_j \in \{0,1\}$ into $0 \leq p_j \leq 1$ (linear programming relaxation):

$$\begin{aligned} &\min z = c'p \\ \text{s.t.}\quad & Ap \geq e \qquad 0 \leq p \leq e \end{aligned} \tag{P}$$

If an optimal solution of (P) is integer, it is optimal for (UCP), as well. In general, however, they are all fractionary and yield a lower bound $z_P^* \leq z_{UCP}^*$.

The linear relaxation, though efficiently solvable, is still a rather difficult problem. A simpler one, called *lagrangian problem* is obtained by removing all of its constraints and combining them to the objective function with suitable multipliers $\lambda_j \geq 0$, so as to penalise their violation:

$$\min z = \tilde{c}'p + \lambda' e \qquad \text{s.t.}\ 0 \leq p \leq e \tag{LP}$$

where $\tilde{c} = c - A'\lambda$ is named the *lagrangian cost* vector.

An optimal solution to (LP) is trivially $p_j^* = 0$ when $\tilde{c}_j > 0$, $p_j^* = 1$ when $\tilde{c}_j \leq 0$. It is integer and, though it violates some of the covering constraints, it is often a good starting point to build a feasible solution $p^H(\lambda)$ for (UCP). Its value is, for all $\lambda \geq 0$, a lower bound on the linear relaxation: $z_{LP}^*(\lambda) \leq z_P^* \leq z_{UCP}^*$. The lagrangian approach consists in determining λ so that $z_{LP}^*(\lambda)$ be as high as possible, since it is likely that in this way the heuristic solution obtained will be better. A technique known as *subgradient ascent* permits to start from an arbitrary value λ_0 and to improve it, potentially up to z_P^*.

3.2. Subgradient ascent

The main idea is to update λ_k step by step, based on the result of the current lagrangian problem $LP(\lambda_k)$. Its optimal solution $p_k^* = p_{LP}^*(\lambda_k)$ violates the covering constraints by $s_{\lambda_k} = e - Ap_k^*$ and its cost is $z_{\lambda_k} = z_{LP}^*(\lambda_k)$. This drives the update of λ_k through the following formula [19]

$$\lambda_{k+1} = \max\left(\lambda_k + s_{\lambda_k}\frac{\left|z_P^* - z_{\lambda_k}\right|}{\left\|s_{\lambda_k}\right\|^2}, 0\right) \qquad (1)$$

At the next step, the violated constraints (corresponding to the positive elements of s_{λ_k}) will be penalised more strongly by increasing the corresponding multiplier, whereas the respected ones will be penalised less strongly, by decreasing it, though never below zero.

The gap $\left(z_P^* - z_{\lambda_k}\right)$ measures how far λ_k is from its optimal value: the larger it is, the larger the update should be. This aims at improving z_{λ_k}, but the resulting process is not monotonous: z_{λ_k} oscillates from step to step. Only its best known value LB progressively rises, getting closer and closer to z_P^*. At the same time, the best value of λ includes more and more information on the problem.

Formula (1) theoretically guarantees convergence. However, in practice z_P^* is unknown, and it is replaced by an upper bound UB. The use of an upper bound could prevent the method to converge, since $\left(UB - z_{\lambda_k}\right)$ can be always positive. Therefore, a decreasing "step" coefficient t_k is added: this is halved each time LB does not improve for a suitable number N_t of consecutive steps, since the lack of improvements is likely due to an excessively large update. This results in a slightly different updating formula:

$$\lambda_{k+1} = \max\left(\lambda_k + t_k s_{\lambda_k}\frac{\left|UB - z_{\lambda_k}\right|}{\left\|s_{\lambda_k}\right\|^2}, 0\right) \qquad (2)$$

The process ends when either z_{λ_k} is very close to UB (i.e. the gap is lower than a given threshold δ) or t_k has become too small to allow further changes ($t_k < t_{min}$). A third stopping condition holds when the cost function c is integer, as we assume in the following. In this case z_{UCP}^* is integer, too: if a known integer solution costs $\lceil z_{\lambda_k} \rceil$, it is optimal, since no other one can be better.

The initial value for λ is arbitrary, as far as non negative, but experience shows that the better it is, the quicker the method converges. If the current problem results from fixing some columns in a previous one, the optimal value of λ is likely to be only slightly different. Therefore, the best value determined for the previous problem is assumed as the initial one in the new problem. At the first step, when no information is available, a good estimate λ_0 is provided by the dual problem.

3.3. The dual problem

Given the linear problem (P), its *dual problem* is

$$\begin{array}{llll} & \max w = e'm & & \\ \text{s.t.} & A'm \leq c & 0 \leq m \leq \bar{c} & (D) \end{array}$$

where m is the dual variable vector and $\bar{c_i} = \min_{j:a_{ij}=1} c_j$.

The dual problem (D) is strictly related to the primal lagrangian problem (LP): if a dual feasible solution m is used as a lagrangian multiplier vector, the corresponding optimum $z_{LP}^*(m)$ equals the value $w(m)$ of the dual solution.

$$c - A'm = \tilde{c}(m) \geq 0 \Rightarrow p_{LP}^*(m) = 0 \Rightarrow$$
$$\Rightarrow z_{LP}^*(m) = m'e = e'm = w_D(m)$$

Thus, any good dual solution is a good lagrangian multiplier vector. In particular, any optimal dual solution is an optimal lagrangian multiplier, since $z_{LP}^*(m^*) = w_D(m^*)$, and it is a well-known consequence of the duality theorem [19] that $w_D^* = z_P^*$. This is why a dual heuristic is used to determine an initial estimate for λ_0.

Of course, solving the dual problem is approximately as hard as solving the primal one. Yet, one can relax it in a lagrangian fashion:

$$\max w = \tilde{e}'m + \mu'c \;\; \text{s.t.} \qquad 0 \leq m \leq \bar{c} \qquad (LD)$$

where $\tilde{e} = e - A\mu$, and the optimum of (LD) is an upper bound $w_{LD}^*(\mu) \geq w_D^* = z_P^*$. This bound possibly improves the value UB used in the subgradient updating formula (2) as an estimate of z_P^*, and it is updated step by step through the subgradient process. Of course, the initial estimate for μ_0 is determined by a primal heuristic.

In short, the subgradient method is applied both to the primal and the dual lagrangian problem. Each of them improves the bound on z_P^* used in the other one, so that the lagrangian multipliers are optimised in a lower number of steps.

3.4. Relations between different lower bounds

In this section, we prove a proposition about the relative strenght of the four lower bounding techniques described in this paper. In particular, we discuss the case of uniform costs, which is rather common in *VLSI* design.

Let us consider the example in Figure 3.4. If MIS is a maximum independent set of rows, the corresponding lower bound is $LB_{MIS} = \sum_{i \in MIS} \min_{j:a_{ij}=1} c_j = 1$: in fact, every

$$c' = \begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 1 & 2 & 1 \\ \hline\end{array}$$

$$A = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 & 1 \\ \hline\end{array}$$

Figure 1. An example about independent-set, dual ascent and linear bounds

row intersects with every other one, and the minimal column covering each row costs 1. However, the dual feasible solution $m = (1, 1, 0, 0)$ yields a lower bound equal to $LB_{DA} = w(m) = 2$. In the end, the optimal solution of the linear relaxation is $p = (0.5, 0.5, 0.5, 0, 0.5)$, and the corresponding lower bound $LB_{LR} = z^*(P) = 2.5$ can be raised up to 3, taking into account that the cost function is integer. If all the costs were equal to 1, the maximum independent set and the dual ascent bound would be $LB_{MIS} = LB_{DA} = 1$, whereas the linear relaxation bound would be $LB_{LR} = 2$.

Proposition 1 *The linear relaxation bound always dominates the lagrangean lower bound. This, if properly initialized, dominates the dual ascent bound, which dominates the independent set bound. In uniform cost problems, the two latter bounds are equal.*

Proof. The linear relaxation bound z^*_P is the optimum of the linear relaxation, which is higher than the value of any feasible dual solution by the weak duality theorem [19]. But the dual ascent bound is by definition such a value.

On the other hand, any independent set of rows M' corresponds to a dual feasible solution m ($m_i = \min_{j:a_{ij}=1} c_j$ for $i \in M'$, $m_i = 0$ for $i \notin M'$): each row variable in the independent set is assigned its maximum feasible value, the cost of the cheapest covering column, whereas the other variables are set to zero. In general, though the independent set variables cannot increase without violating some constraint, the other ones can. Therefore, dual ascent can improve the independent set bound.

In case the cost function is uniform ($c_j = 1$, for all j), all integer dual solutions correspond to independent sets. Since $m_i \leq \min_{j:a_{ij}=1} c_j = 1$, either $m_i = 1$ or $m_i = 0$. If rows i and i' intersect in column j, $a_{ij} = a_{i'j} = 1$, and m_i and $m_{i'}$ cannot be both set to 1. The rows whose dual variables are set to 1 always form an independent set.

The lagrangean lower bound $z^*_{LP}(\lambda)$ depends on the multipliers λ employed. If these derive from a dual heuristic, it is equal to the dual ascent bound. Step by step, however, its value grows and, theoretically, reaches z^*_P. ■

3.5. Lagrangian heuristics

This section describes the primal and dual auxiliary heuristics used to estimate λ_0 and μ_0 and to improve the best-known upper and lower bounds on z^*_P, UB and LB. Unlike the linear relaxation (P), both the primal and the dual lagrangian relaxations, (LP) and (LD), have an integer solution. This is usually unfeasible, but often proves useful as a starting point to build good feasible solutions. In fact, if nearly optimal multipliers are available, the corresponding lagrangian solution is likely to be nearly optimal and the lagrangian costs should give a good information on how to modify it in order to reach feasibility [2].

The auxiliary primal heuristic follows a simplified version of the greedy scheme that drives the whole algorithm [1]. Starting from the unfeasible solution, it selects a column at a time until it finds a feasible one; then reductant columns are removed. It chooses the column which minimises a suitable combination γ_j of the lagrangian cost $\tilde{c}_j$ and the number n_j of currently uncovered rows. Experience shows that using the lagrangian costs instead of the original ones leads to better results, because they weigh the importance of the rows, through the multipliers λ. Four different versions of this heuristic are used in our algorithm. The first three, respectively, define γ_j as $\tilde{c}_j / n_j$, $\tilde{c}_j / \lg_2(n_j + 1)$ and $\tilde{c}_j / n_j \lg_2(n_j + 1)$. The fourth one furtherly keeps into account the importance of the rows, by counting how many columns cover them: rows covered by few columns are more important [10]:

$$\gamma_j = \frac{\tilde{c}_j}{\sum\limits_{mRp} \frac{1}{|\{p \in P : mRp\}| - 1}}$$

Moreover, it applies reductions at each fixing step. Since it is slower, it is applied only to the initial problem, when determining μ_0.

The dual heuristic, known as *dual ascent*, is the counterpart of the primal one. During the first phase, the variables are decreased until all of the constraints get satisfied. The rows covered by the maximum number of columns are considered first, in order to reduce the infeasibility as much as possible with the minimum decrease in the objective function. Then, we set:

$$m_i = \max\left\{ m_i - \max\left[\max_j \left(\sum_{i=1}^{|M|} a_{ij} m_i - c_j \right), 0 \right], 0 \right\}$$

Term $\sum_{i=1}^{|M|} a_{ij} m_i - c_j$, if positive, is the amount by which dual constraint j is violated by solution m. We determine the most violated constraint including the current variable m_i. Then, we reduce m_i until either the constraint is satisfied or m_i gets equal to zero. In either case, the constraints become somewhat more respected; in the end, they will all be honoured, possibly by setting all variables to zero.

During the second phase, the variables are increased, improving the objective function, but keeping the solution feasible. They are taken into account in increasing occurrence order, so that the effect is maximum on the objective and minimum on the constraints. Since term $c_j - \sum_{i=1}^{|M|} a_{ij} m_i$ measures how respected the constraints are, we set:

$$m_i = m_i + \min_j \left(c_j - \sum_{i=1}^{|M|} a_{ij} m_i \right)$$

3.6. Penalty conditions

The lagrangian and dual information permits to reduce the problem, by determining whether a given column belongs or not to the optimal solution. The concept is to accomplish an implicit branching on this variable and immediately prune one of the two subproblems. Let us branch on p_j, by setting, respectively $p_j = 0$ and $p_j = 1$, and generating problems $(UCP0)$ and $(UCP1)$.

The lagrangian relaxations $(LP0)$ and $(LP1)$ are equal to problem (LP), apart from the additional branching constraint.

$$\begin{array}{llll} & \min z = \tilde{c}' p + \lambda' e & & \\ \text{s.t.} & p_j = 0 & 0 \le p \le e & (LP0) \end{array}$$

$$\begin{array}{llll} & \min z = \tilde{c}' p + \lambda' e & & \\ \text{s.t.} & p_j = 1 & 0 \le p \le e & (LP1) \end{array}$$

If $\tilde{c}_j \le 0$, problem $(LP1)$ has the same optimal solution as (LP), since this satisfies the additional constraint. In problem $(LP0)$, on the contrary, the value of p_j is different, but the other variables are not affected. Thus, $z^*_{LP0} = z^*_{LP} - \tilde{c}_j$. This is a lower bound for $(UCP0)$: no feasible solution with $p_j = 0$ costs less. If the best known heuristic solution p_{best} is not worse, subproblem $(UCP0)$ can be pruned:

$$z^*_{LP} - \tilde{c}_j \ge z_{best} \Rightarrow p^*_j = 1 \qquad (3)$$

Conversely, if $\tilde{c}_j > 0$ and $z^*_{LP1} = z^*_{LP} + \tilde{c}_j \ge z_{best}$, subproblem $(UCP1)$ can be pruned:

$$z^*_{LP} + \tilde{c}_j \ge z_{best} \Rightarrow p^*_j = 0 \qquad (4)$$

Dual penalties exploit the same principle, but with a different lower bound and cost vector. Let $(P0)$ and $(P1)$ denote the linear relaxations of $(UCP0)$ and $(UCP1)$. We reformulate their dual problems $(D0)$ and $(D1)$, so as to obtain the same variables as in (D), though one more constraint is imposed. As for $(P0)$, p_j can be forced to zero by setting $c_j = +\infty$. The dual problem $(D0)$ is identical to D, but constraint j is relaxed, since its right-hand side has risen to ∞. Any feasible solution to $(D0)$ yields a lower bound on z^*_{UCP0}. So, if a better solution z_{best} exists, problem $(UCP0)$ can be pruned:

$$w^*_{D0} = w^*_D|_{c_j=+\infty} \ge z_{best} \Rightarrow p^*_j = 1. \qquad (5)$$

Conversely, setting $p_j = 1$ $(P1)$ is equivalent to adding a constant c_j to the objective and imposing $c_j = 0$. The dual problem $(D1)$ is identical to (D), but the right-hand side of the j-th constraint is null and forces to zero all the variables occuring in it. If the value of a dual heuristic solution is not lower than z_{best}, $(UCP1)$ can be pruned:

$$w^*_{D1} = w^*_D|_{c_j=0} + c_j \ge z_{best} \Rightarrow p^*_j = 0. \qquad (6)$$

These conditions generalize a well-known pruning technique, the *limit bound theorem* [10].

Theorem 2 *Let M' be an independent set of rows for the unate covering problem (M, P, R, c) and $LB_{M'}$ the corresponding lower bound on the optimum. Let z_{best} be any upper bound on the optimum. If column p_j does not cover any element of M' and*

$$LB_{M'} + c_j \ge z_{best}$$

then column p_j can be removed from the problem.

Proposition 3 *The limit bound theorem is a special case of the dual penalties.*

Proof. Any independent set of rows M' corresponds to a feasible dual solution m', whose value $w(m')$ equals $LB_{M'}$. If the hypothesis of the limit bound theorem hold, $w(m') + c_j \ge z_{best}$. Since column p_j does not cover any row in M', m' is feasible for the dual problem modified by imposing $c_j = 0$. Therefore, $w^*_D|_{c_j=0} \ge w(m')$, and the dual penalty condition for removing column p_j, which is $w^*_D|_{c_j=0} + c_j \ge z_{best}$, holds. However, if the cost function is not uniform, better dual solutions than m' can be found, and dual penalties can also fix columns. ■

3.7. Fixing conditions

Each step of the algorithm adds a number of columns to the current solution. The choice is based on the information deriving both from the primal and the dual lagrangian relaxation. In fact, a column is likely to be optimal whenever it has a low lagrangian cost and whenever the corresponding dual lagrangian multiplier is close to 1. The variables which have both a very low lagrangian cost ($\tilde{c}_j \le \hat{c} = 0.001$) and a dual lagrangian value very close to 1 ($\mu_j \ge \hat{\mu} = 0.999$) are added to the current solution.

In any case, the algorithm always adds a column, to guarantee convergence. This column is the one which minimises a suitable combination of $\tilde{c}_j$ and μ_j:

$$\sigma_j = \tilde{c}_j - \alpha \mu_j$$

where parameter α needs to be experimentally tuned (we set $\alpha = 2$).

4. The algorithm

In this section, we assemble all the elements we have presented so far into an outline of the algorithm (Figure 4). Essentially, the algorithm follows a greedy constructive

```
Algorithm ZDD_SCG($f, c$)

Begin

    $(ZDDr, ZDDc) = Encode(f)$;                                    {Turn $f$ into a couple of ZDDs}
    $p = 0$;                                                       {Start from an empty solution}
    Repeat
        $ZDD'r = ZDDr; ZDD'c = ZDDc$;
        $(ZDDr, ZDDc, p) = ZDD\_Reductions(ZDD'r, ZDD'c, p)$;
    until $(ZDD'r = ZDDr$ and $ZDD'c = ZDDc)$ or $(Card(ZDDr) \leq MaxR$ and $Card(ZDDc) \leq MaxC)$;
          {The covering matrix is a cyclic core or it is small }
    $A = Decode(ZDDr, ZDDc)$;                                      {Turn the ZDDs into a sparse matrix}
    $A = Explicit\_Reductions(A)$;                                 {Perform explicit reductions}
    $(\lambda, \mu, LB, p_{best}, z_{best}, p) = SubgradientAscent(A, c)$;
    If $z_{best} = \lceil LB \rceil$ then return $p_{best}$;        {The optimal solution has been found}
    $A_e = A; p_e = p$;                         {Save the exact cyclic core and the essential columns}
    For $Iter := 1$ to $NumIter$ do
        Repeat                                                     {Fixing step}
            Add the promising columns to $p$;
            $\sigma = c - A\lambda - \alpha\mu$;                   {Rate the columns}
            Choose at random one of the best $BestCol$ columns and add it to $p$;
            Repeat
                $A' = A$;
                $A = Explicit\_Reductions(A', p)$;
            until $A = A'$;
            $(\lambda, \mu, LB, p_{best}, z_{best}, p) = SubgradientAscent(A, c)$;
        until $A = \emptyset$ or $z_{best} \leq \lceil LB \rceil$;    {Either $p$ or $p_{best}$ is the best solution found}
        If $z_{best} > \lceil LB \rceil$ then $p_{best} = p$;
        While $p_{best}$ is redundant do
            Remove the highest cost redundant column from $p_{best}$;
        EndWhile
        $A = A_e; p = p_e$;                                        {Restore the exact cyclic core}
    EndFor;

Return $p_{best}$;

End
```

Figure 2. Pseudocode of the algorithm

scheme: starting from an empty solution p, columns are subsequently added, until all of the rows are covered; then, p is made irredundant by discarding as many of its elements as possible. First of all, the minterms and the prime implicants of function f are derived from a given circuit implementation and encoded into a couple of decision diagrams, respectively *ZDDr* for the former (rows) and *ZDDc* for the latter (columns). Procedure *ZDD_Reductions* runs the reduction procedures [10] as long as they have any effect on *ZDDr* and *ZDDc* or the size of the esplicit representation is small enough (i.e. less than $MaxR = 5000$ rows and less than $MaxR = 10000$ columns). The essential columns found are added to p. Then, the resulting problem is converted into a sparse matrix A and reduced using the standard routines for row and column dominance.

Procedure *SubgradientAscent* rates the remaining columns in order to determine which of them should be added to p. In short, the underlying concept is to solve a lagrangian relaxation of both the primal and the dual problem and to update the corresponding multipliers, λ and μ, so that the relaxations be as tight as possible. When this is achieved, λ and μ contain a good deal of precious information about the optimal solution. If lagrangian or dual penalty conditions are satisfied (Section 3.6), some columns, proven optimal, are added to p; some others, proven non optimal, are removed. As dual penalties are computationally heavier, they are executed only once during each subgradient phase and they are skipped if the number of columns exceeds a given value ($DualPen = 100$).

At each step, *SubgradientAscent* yields a heuristic solution and a lower bound on the optimum. The best heuristic solution is saved as p_{best}, and its value as z_{best}. The best lower bound, LB, is used to measure the quality of p_{best}: if $z_{best} = \lceil LB \rceil$, p_{best} is surely optimal and the process terminates. Otherwise, a number of “promising” columns, which satisfy suitable conditions on the primal and the dual problem, are added to p, even though this cannot be strictly proven to be correct. In the end, procedure *Explicit_Reductions* removes redundant rows and columns from the covering matrix, through classical reduction algorithms. *SubgradientAscent*, the heuristic fixing and *Explicit_Reductions* are cyclically executed, until either p gets feasible or LB exceeds z_{best}. In the latter case, in fact, p cannot become better than p_{best}, and any further fixing would be of no use.

The algorithm is run *NumIter* times. During the first run, the column fixing strictly chooses the best-rating column; in the following ones, it is partly stochastic, so that presumably different solutions are obtained, without recurring to specific data structures: the best-rating *BestCol* candidates are taken into account and one of them, at random, is fixed. Whereas a depth-first branch-and-bound algorithm punctiliously visits solutions near one another, this heuristic approach first explores the most promising choices, so that possibly better results are obtained earlier. The value of *BestCol* grows from run to run, in order to explore a wider region of the solution space. Of course, each run needs not start from the beginning: it is enough to retrieve the exact cyclic core of the problem (matrix A_e) and the essential columns (vector p_e), saved before any heuristic choice is taken.

5. Experimental results

In this section, we compare the results obtained by *ZDD_SCG* to the best-known results for 72 benchmark boolean minimisation problems in the Berkeley Programmed Logic Array (*PLA*) test set. Some of them have *don't care* sets and their size varies from 4 to 128 inputs, from 1 to 109 outputs, while the initial cover size extends from 4 to 2406 terms. The cost function is assumed to be the number of products producing a cover of the circuit functionality, with only a secondary concern given to the number of literals. As for their complexity, these problems are divided into five categories, based on the state of the art at the time they were proposed: *easy cyclic* (the cyclic core is not empty and the covering problem had been solved - 49 instances), *difficult cyclic* (the cyclic core is not empty and

	ZDD_SCG				Espresso		Espr. Strong	
Name	Sol	CC(s)	T(s)	M	Sol	T(s)	Sol	T(s)
bench1	121	1.90	14.26	13	139	1.01	127	2.83
ex5	65	186.40	294.66	51	74	0.54	74	1.15
exam	63	0.49	6.99	12	67	2.11	64	5.46
max1024	260	0.51	36.55	11	274	4.32	267	5.39
prom2	287	8.93	18.91	29	287	6.77	287	7.23
t1	100*	6.27	6.69	18	102	0.62	102	0.93
test4	96	24.83	617.54	15	120	6.70	104	17.48

Table 1. Results for *ZDD_SCG* and *Espresso* on the difficult cyclic problems

the covering problem was unsolved - 7 instances) and *challenging* (the prime implicants had not even been completely enumerated - 16 instances).

For a few *challenging* and *difficult cyclic* problems the optimal value has not yet been found: *bench1*, *exam*, *test4*, *ex1010*, *test2* and *test3*. Only an upper bound on the optimum of these problems is known and in particular we report the ones published in [10]. At the best of our knowledge they are the best solution identified up to now, with the exception of *test2* where *Espresso* gives a better upper bound.

The algorithm described in this paper has been implemented in C language, exploiting the *CUDD* library from the University of Boulder, Colorado [21], for the management of *ZDD*s. All the CPU times are obtained by running the experiments on a UtraSparc30/248 with 1Gbyte RAM.

The first experiment considers the *easy cyclic* problems. In short, the algorithm, though heuristic, solves to optimality all of the problems. The total cost on all the 49 instances is 5225, and the total lagrangian lower bound is 5213, with a gap from the optimum equal to 0.22%. *Espresso* [3] obtains a total cost of 5330 with its normal heuristic mode and 5281 with the *Espresso* strong mode.

The second experiment (see Table 1 and Table 2) compares *ZDD_SCG* to *Espresso* in the heuristic normal and strong modes, not in the *Exact* or *Signature* mode. The tables report the cost of the solution, the time $CC(s)$ to compute the cyclic core (summing the implicit and explicit phases), the CPU total time $T(s)$ and the memory required by *ZDD_SCG*, in megabytes. All times are measured in seconds. A star marks the solution cost when it is proved to be optimal. The solution cost and the total analysis time are also reported for *Espresso* in both modes.

In some cases the memory requirement is quite high but this is mainly due to the cyclic core computation. Note that [12] has a much lower memory requirement for the same cyclic core computation (roughly, less than $100MB$).

From the CPU time point of view, *Espresso* is alway faster than *ZDD_SCG* and this is mainly due to the time required by the cyclic core computation. Note that, for all benchmarchs where *ZDD_SCG* and *Espresso* determine equivalent solutions (with the exclusion of *prom2*) we prove that this is an optimal solution. In all other cases, *ZDD_SCG* always identifies better solutions than *Espresso* in heuristic

	ZDD_SCG				Espresso		Espr. Strong	
Name	Sol	CC(s)	T(s)	M	Sol	T(s)	Sol	T(s)
ex1010	239	146	1501	23	284	9.25	262	16.83
ex4*	279	10.38	10.38	13	279	3.79	279	4.22
ibm*	173	43.56	43.56	48	173	0.28	173	0.31
jbp*	122	74.56	74.58	15	122	0.98	122	1.11
misg*	69	0.60	0.60	9	69	0.11	69	0.17
mish*	82	0.76	0.76	9	82	0.19	82	0.25
misj*	35	0.16	0.16	9	35	0.02	35	0.04
pdc	96	72.56	77.54	51	145	12.61	119	15.46
shift*	100	73.16	73.16	51	100	0.04	100	0.04
soar.pla	352	4294	4333	158	353	8.84	352	11.16
test2	865	19105	108058	414	1103	128.7	**946**	356.2
test3	436	7978	16145	218	541	70.73	489	129.6
ti*	213	955	954.88	88	213	3.28	213	3.37
ts10*	128	1.11	1.11	10	128	0.05	128	0.06
x2dn*	104	10.24	10.24	13	104	0.54	104	0.63
xparc*	254	297	297.31	89	254	6.11	254	6.26

Table 2. Results for *ZDD_SCG* and *Espresso* on the challenging problems

	ZDD_SCG			Scherzo	
Name	Sol	T(s)	MaxIter	Sol	T(s)
bench1	**121(120)**	12.36	1	**122H**	
ex5	65(60)	108.26	12	65	31113
exam	63(59)	6.50	1	63H	
max1024	260(255)	36.04	2	259	15110
prom2	287(285)	9.98	1	287	4111
t1	100*	0.42	1	100	0.02
test4	**96(78)**	592.71	1	**100H**	

Table 3. Results on the difficult cyclic problems

mode.

The last experiment concerns the quality of the solution proposed by *ZDD_SCG* with respect to the exact one. Table 3 and Table 4 compare the results obtained by our approach with respect to *Scherzo* [10]. Since *ZDD_SCG* and *Scherzo* adopt the same algorithm in the implicit part, we only compare the solution cost and the time required to solve the cyclic core. As before, a single starred value marks a proved optimum. Otherwise, a lower bound in parenthesis follows the heuristic solution. When the optimal value is unknown, the last column contains the previous best-known result, marked by an "H", which is only an upper bound. In particular, we report the one published in [10], for which no CPU time is given.

Concerning *difficult cyclic* problems, all but one of these are solved to optimality: for the industrial problem *max1024* the gap is equal to 1. On the other hand, improved solution costs have been found for two instances: *test4* and *bench1*.

For the *challenging* problems, 11 instances are proved to be optimal while on three unsolved problems, we obtain a large improvement over the best-known results. Our result for *ex1010* is 239 against 246. We obtain a lower bound equal to 220 and an upper bound equal to $z^H_{new} = 239$; the old heuristic result was equal to $z^H_{old} = 246$. So, the error has been reduced from at most 26 to at most 19, which is a $\left(z^H_{old} - z^H_{new}\right) / \left(z^H_{old} - LB\right) = 7/26 * 100 = 27\%$ decrease.

	ZDD_SCG			Scherzo	
Name	Sol	T(s)	MaxIter	Sol	T(s)
ex1010	**239 (220)**	1355.56	1	**246H**	
ex4	279*	0.00	1	279	0.00
jbp	122*	0.02	1	122	0.00
pdc	96(92)	5.21	1	96	1.80
soar.pla	352(350)	39.87	1	352	56.83
test2	**865(756)**	88956	1	**995H**	
test3	**436(390)**	8167.62	1	**477H**	
ti	213*	0.50	1	213	0.15
xparc	254*	0.03	1	254	0.02

Table 4. Results on the challenging problems

This is an underestimate: should the optimum z^*_{UCP} be higher than *LB*, the error reduction would be stronger. Our result for *test2* is 865 against 946 (obtained in less than 25 hours). As the lower bound is equal to 756, the error has been reduced at least by 43%. In the end, our result for *test3* is 436 against 477 (in less than 3 hours). As the lower bound is equal to 390, the error has been reduced at least by 47%. Finally the exact solver *Aura* [14] analyzes *pdc*, *soar.pla* and *ex5*, *prom2*, *max1024* in 1.81, 56.37, 1994.1, 3853.1 and 8174.10 seconds, respectively.

6. Conclusion

We have presented a heuristic algorithm for unate covering problems, specifically for two-level logic minimisation. The algorithm proves extremely effective and fast, obtaining in a tolerable amount of time results which are nearly always optimal. On some problems, whose exact solution is unknown, our results are much better than the best-known ones. The algorithm is heuristic, but it also gives an estimate of the quality of the solution attained, through the evaluation of a lower bound. On most benchmark problems, this lower bound equals the value of the solution, thus proving its optimality. In most of the other cases, the gap reduces to few units.

7. Acknowledgements

We would like to thank Tiziano Villa and Luca Carloni who provided a version of *Aura* (the version integrated in *Scherzo*) and for the interesting discussion on Unate and Binate Covering Problems. In addition, we wish to thank the anonymous DATE-2000 reviewers for their useful comments on an earlier draft of this paper.

References

[1] E. Balas and A. Ho. Set Covering using cutting planes, heuristics and subgradient optimisation: A computational study. *Mathematical Programming Study*, 12:37–60, 1980.

[2] J. E. Beasley. An algorithm for Set Covering problems. *European Journal of Operational Research*, 31:85–93, 1987.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Dordrecht, 1984.

[4] R. K. Brayton, P. C. McGeer, J. Sanghavi, and A. L. Sangiovanni-Vincentelli. A new exact minimizer for two-level logic synthesis. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 1–31. Kluwer Academic Publishers, Dordrecht, 1993.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–692, August 1986.

[6] A. Caprara, M. Fischetti, and P. Toth. A heuristic algorithm for Set Covering problem. In *Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference (IPCO)*, 1996.

[7] S. Ceria, P. Nobili, and A. Sassano. A lagrangian-based heuristic for large-scale Set Covering problems. *Mathematical Programming*, 81:215–228, 1998.

[8] A. K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24:7–11, 1978.

[9] V. Chvátal. A greedy heuristic for the Set Covering problem. *Mathematics of Operations Research*, 4(3):233–235, August 1979.

[10] O. Coudert. Two-level logic minimization: An overview. *INTEGRATION, the VLSI journal*, 17:97–140, 1994.

[11] O. Coudert. On solving covering problems. In *Proceedings of the 33rd ACM/IEEE Design Automation Conference*, pages 197–202, Las Vegas, NV, June 1996. ACM-SIGDA; IEEE, ACM Press.

[12] O. Coudert, J. C. Madre, and H. Fraisse. A new viewpoint on twolevel logic minimization. In *Proceedings of the 3oth ACM/IEEE Design Automation Conference*, pages 625–630, Dallas, TX, June 1993.

[13] M. L. Fisher and P. Kedia. Optimal solution of the Set Covering/Partitioning problem using dual heuristics. *Management Science*, 36:674–688, 1990.

[14] E. I. Goldberg, L. P. Carloni, and A. L. Sangiovanni Vincentelli T. Villa, R. K. Brayton. Negative thinking by incremental problem solving: Application to unate covering. In *Proceedings of the IEEE /ACM International Conference on CAD*, Santa Clara, California, November 1997. ACM/IEEE, IEEE Computer Society Press.

[15] S. Liao and S. Devadas. Solving covering problems using LPR-based lower bounds. In ACM-SIGDA; IEEE, editor, *Proceedings of the 34th ACM/IEEE Design Automation Conference*, Anaheim, CA, June 1997. ACM Press.

[16] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[17] E. L. McCluskey Jr. Minimization of boolean functions. *Bell System Technical Journal*, 35:1417–1444, April 1959.

[18] S.-i. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In ACM-SIGDA; IEEE, editor, *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 272–277, Dallas, TX, June 1993. ACM Press.

[19] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[20] W. V. O. Quine. On cores and prime implicants of truth functions. *American Mathematics Monthly*, 66:755–760, 1959.

[21] F. Somenzi. *CUDD: Colorado University Decision Diagram Package*. University of Colorado, Boulder, 1994.

[22] G. M. Swamy, P. McGeer, and R. K. Brayton. A fully Quine-McCluskey procedure using BDD's. Report UCB/ERL M92/127, UCB, November 1992. Also in: Proceedings of the International Workshop on Logic Synthesis, Lake Tahoe, CA, May 1993.

[23] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Explicit and implicit algorithms for binate covering problems. *IEEE Transactions on Computers*, 16(7):677–691, July 1997.