In Proc. of the IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI), pages 232-239, Nov. 1997.

Prime Implicant Computation Using Satisfiability Algorithms

Vasco M. Manquinho, Paulo F. Flores, João P. Marques Silva, Arlindo L. Oliveira
Cadence European Laboratories
IST/INESC
1000 Lisboa, Portugal
vmm@calvin.inesc.pt,{pff,jpms,aml}@inesc.pt

Abstract

The computation of prime implicants has several and significant applications in different areas, including Automated Reasoning, Non-Monotonic Reasoning, Electronic Design Automation, among others. In this paper we describe a new model and algorithm for computing minimum-size prime implicants of propositional formulas. The proposed approach is based on creating an integer linear program (ILP) formulation for computing the minimum-size prime implicant, which simplifies existing formulations. In addition, we introduce two new algorithms for solving ILPs, both of which are built on top of an algorithm for propositional satisfiability (SAT). Given the organization of the proposed SAT algorithm, the resulting ILP procedures implement powerful search pruning techniques, including a non-chronological backtracking search strategy, clause recording procedures and identification of necessary assignments. Experimental results, obtained on several benchmark examples, indicate that the proposed model and algorithms are significantly more efficient than other existing solutions.

1 Introduction

Given a propositional formula φ in Conjunctive Normal Form (CNF), denoting a boolean function f, the problem of computing a minimum-size assignment (in the number of literals) that satisfies f is referred to as the ***minimum-size prime implicant problem***. Minimum-size prime implicants find application in many areas including, among others, Automated Reasoning, Non-Monotonic Reasoning and Electronic Design Automation. Moreover, interest on computing minimum-size prime implicants of boolean functions has motivated extensive research work (see for example [8, 9], which include comprehensive bibliographic references.).

In this paper we describe an Integer Linear Program (ILP) formulation for computing minimum-size prime implicants of boolean functions described by Conjunctive Normal Form (CNF) formulas. The proposed ILP model, first introduced in [14], significantly simplifies the one originally described in [9]. Moreover, we propose two new algorithms for solving ILPs where the variables have boolean domains (01-ILPs). Both algorithms are based on propositional satisfiability (SAT). The first one generalizes the SAT-based ILP algorithm originally described in [2], whereas the second one describes a branch and bound ILP procedure built on top of a SAT solver. For both algorithms the GRASP SAT solver [12, 15] is used. Preliminary results, obtained on several satisfiable instances of the DIMACS benchmarks [6], indicate that the proposed model and algorithms can be used for computing minimum-size prime implicants for several classes of boolean functions. Furthermore, we show that widely used ILP algorithms, most of which are based on linear-programming (LP) relaxations [7], may be inadequate for computing minimum-size prime implicants. This result strongly suggests using dedicated ILP algorithms for solving the minimum-size prime implicant problem. Finally, we note that the proposed ILP algorithms, implement powerful search pruning techniques commonly used in SAT algorithms, which include a non-chronological backtracking search strategy, clause recording procedures and identification of necessary assignments.

The paper is organized as follows. In Section 2 the notational framework used throughout the paper is introduced. Afterwards, we describe the ILP model for computing minimum-size prime implicants of boolean functions described with CNF formulas. This model is based on the one proposed in [9], but significantly reduces the number of variables as well as the size of their domains. Consequently, the worst-case size of the search space becomes drastically reduced, and hence a smaller search space is expected for most practical examples. In Section 4 we describe two new SAT-based ILP algorithms and illustrate how the identification of prime implicants can be iterated. The procedure proposed in this paper for iterating prime implicants by increasing size is provably more efficient than the one described in [9]. The different algorithms are experimentally evaluated in Section 5. Finally, Section 6 concludes the paper suggesting potential improvements to the proposed model and algorithms.

2 Definitions

The paper follows the definitions introduced in [12, 14], in particular for the organization of the GRASP SAT algorithm which is described in Section 4. In general, a propositional formula φ in CNF denotes a boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where each clause ω is a sum of literals, and a literal l is either a variable x_i or its complement x_i'. For a search-based algorithm for SAT, a ***conflict*** is said to be identified when all literals of at least one clause are assigned value 0. The GRASP SAT [12] algorithm implements several techniques for pruning the amount of search based on the ***diagnosis*** of conflicts identified during the search.

A clause $\omega = (l_1 + \ldots + l_k)$ denotes a constraint which can also be viewed as a linear inequality, $l_1 + \ldots + l_k \geq 1$. We use this alternative representation when appropriate. Furthermore, since a literal $l = x_i'$ can also be defined by $l = 1 - x_i$, we shall in general use this latter representation when viewing clauses as linear inequalities.

3 Prime Implicant Computation Using Integer Programming

Given a description of a Boolean function in CNF, it is straightforward to formulate the computation of the minimum-size prime implicant as an integer linear program [9]. In this paper we show how to simplify the formulation proposed in [9], thus allowing for a significant reduction in the worst-case search space. This improved model was first described in [14].

Given a CNF formula φ, which is defined on a set of variables $\{x_1, \ldots, x_n\}$, with p clauses $\{\omega_1, \ldots, \omega_p\}$, and which denotes a Boolean function $f : \{0,1\}^n \rightarrow \{0, 1\}$, apply the following transformation.

1. Create a new set of boolean variables $\{y_1, \ldots, y_{2n}\}$, where y_{2i-1} is associated with literal x_i, and y_{2i} is associated with literal x_i'.
2. For each clause $\omega = (l_1 + \ldots + l_m)$, replace each literal l_j with y_{2k-1} if $l_j = x_k$, or with y_{2k} if $l_j = x_i'$.
3. For each pair of variables, y_{2i-1} and y_{2i}, require that at most one is set to one. Hence, $y_{2i-1} + y_{2i} \leq 1$.
4. The set of inequalities obtained from steps 2. and 3. can be viewed as a single set of inequalities $\boldsymbol{A} \cdot \boldsymbol{y} \geq \boldsymbol{b}$. Furthermore, define the cost function to be,

$$\min \sum_{j=1}^{2n} y_j \tag{1}$$

5. The complete ILP formulation is thus defined as follows:

$$\begin{aligned} \min \quad & \sum_{j=1}^{2n} y_j \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \boldsymbol{y} \geq \boldsymbol{b} \end{aligned} \tag{2}$$

It is clear that the solution of (2) denotes a minimum-size prime implicant of the original CNF formula φ, and from [9] we have,

Proposition 1. Given a CNF formula φ and associated boolean function f, the solution of the optimization problem (2) is a minimum-size prime implicant of f.

The proposed ILP model is based on the one described in [9]. However, the model proposed in [9] associates an integer variable λ_i with each inequality created from each original clause ω_i. As we showed above, such integer variables are unnecessary and only increase the worst-case search space. Indeed, for the ILP model of (2), the worst-case search space is,

$$2^{2n} = 4^n \tag{3}$$

whereas for the ILP model of [9], the worst-case search space is,

$$4^n \cdot \prod_{i=1}^{p} |\omega_i| \tag{4}$$

where $|\omega_i|$ denotes the number of literals of clause ω_i, and represents the least upper bound on the integer variable λ_i associated with clause ω_i and introduced in the ILP model of [9]. Consequently, the worst-case search space for the ILP model we propose in this paper is provably less than for the model proposed in [9]. We should note that for both models, and for a search-based ILP algorithm, a straightforward arrangement of the order of the decision variables leads to a worst-case search space of 3^n (since only 3 assignments are possible for each of the n pairs of variables), but unfortunately this information cannot in general be made available to the ILP solver.

The construction of the ILP model (2) will be illustrated with the following CNF formula:

$$\varphi = (x_1 + x_2 + x_3) \cdot (x_1' + x_2') \cdot (x_1' + x_3') \tag{5}$$

First, we start by creating a new set of variables $\{y_1, y_2, y_3, y_4, y_5, y_6\}$, and associate y_{2i-1} with each χ_i and y_{2i} with each x_i'. Consequently, from (5) the following modified CNF formula φ' is obtained:

$$\varphi' = (y_1 + y_3 + y_5) \cdot (y_2 + y_4) \cdot (y_2 + y_6) \tag{6}$$

For each clause, simple algebraic manipulation yields

an equivalent inequality:

$$\begin{aligned} y_1 + y_3 + y_5 &\geq 1 \\ y_2 + y_4 &\geq 1 \\ y_2 + y_6 &\geq 1 \end{aligned} \tag{7}$$

The next step is to require that at most one variable of each pair of variables y_{2i-1}, y_{2i} can be set to one, which yields:

$$\begin{aligned} y_1 + y_2 \leq 1 &\Leftrightarrow -y_1 - y_2 \geq -1 \\ y_3 + y_4 \leq 1 &\Leftrightarrow -y_3 - y_4 \geq -1 \\ y_5 + y_6 \leq 1 &\Leftrightarrow -y_5 - y_6 \geq -1 \end{aligned} \tag{8}$$

Thus allowing for a given variable x_i not to be assigned. Equations (7) and (8) define the set of inequalities $\boldsymbol{A} \cdot \boldsymbol{y} \geq \boldsymbol{b}$. The next step is to identify the cost function, which minimizes the number of variables assigned value one, i.e. the number of variables x_i with an assigned value. Finally the resulting ILP model becomes:

$$\begin{aligned} \min \quad & y_1 + y_2 + y_3 + y_4 + y_5 + y_6 \\ \text{s.t.} \quad & \boldsymbol{A} \cdot \boldsymbol{y} \geq \boldsymbol{b} \end{aligned} \tag{9}$$

One solution to the integer linear program (9) is, for example, $x_1 = 0$ and $x_2 = 1$.

4 Search Algorithms for Solving ILPs

In [2] P. Barth described how to solve ILPs using a propositional satisfiability algorithm. However, the ILP algorithm described in [2] is based on the Davis-Putnam [5] procedure, which has been shown to be particularly inefficient for a large number of instances of SAT [12].

In this section we describe two different algorithms for solving ILPs associated with instances of the minimum-size prime implicant problem. Both are based on SAT algorithms. The first algorithm follows P. Barth's approach, whereas the second builds a branch and bound procedure on top of a SAT engine. The two algorithms use the GRASP SAT algorithm described in [12], which includes several powerful pruning techniques for reducing the amount of search associated with instances of SAT. Among the pruning techniques included in GRASP, the following have been shown to be particularly significant:

- GRASP implements a ***non-chronological backtracking*** search strategy[1]. This backtracking strategy potentially permits skipping over large portions of the decision tree for some instances of SAT.
- GRASP utilizes selective ***clause recording*** techniques. During the search process, and as conflicts are diagnosed, new clauses are created from the causes of the conflicts. These clauses represent implicates of the boolean function associated with the CNF formula, and are often referred to as *nogoods* [11]. Newly recorded clauses are then used for pruning the subsequent search. Moreover, bounds on the size of recorded clauses can be imposed, thus preventing an excessive growth of the resulting CNF formula.
- In most practical situations, instances of SAT can have highly structured CNF representations. The intrinsic structure of these representations can be exploited by GRASP, after diagnosing the causes of conflicts, by identifying ***necessary assignments*** required for preventing conflicts from being identified during the search.
- In addition, other pruning techniques can be straightforwardly applied to SAT algorithms. In particular, and as described in [13], several techniques commonly used in algorithms for different variations of the set covering problem [4].

1. Some variations of this strategy are also commonly referred to as ***dependency-directed backtracking*** and ***backjumping*** [10].

4.1 SAT-Based Linear Search Algorithm

The first ILP algorithm follows P. Barth's ILP algorithm [2] and was first described in [14]. Let us consider the cost function (1). The possible values assumed by the cost function for the different assignments to the variables in the set $\{y_1, \ldots, y_{2n}\}$ range from 0, when all variables are assigned value 0, to $2n$, when all variables are assigned value 1. Note however, that for the minimum-size prime implicant problem a trivial upper bound is n, since for any pair of variables y_{2i-1}, y_{2i} at most one can be assigned value 1. P. Barth's [2] approach consists of applying the following sequence of steps, starting from an upper bound of $k = n$ on the value of the cost function:

1. Create a new inequality $\sum y_j \leq k$. This inequality basically requires that a computed solution must have no more than k literals assigned value 1.
2. Solve the resulting instance of satisfiability. (Note that the resulting instance of satisfiability is defined on linear inequalities, but modifying most SAT algorithms for handling this generalization is straightforward.)
3. If the instance of SAT is satisfiable decrement k (i.e. specify a new value for the cost function) and go back to step 1. Otherwise, report that the solution to the ILP is $k+1$.

Note that this ILP algorithm allows for any SAT algorithm to be used as the underlying SAT testing engine, provided the algorithm is modified to handle linear inequalities. The proposed ILP algorithm is illustrated in

```
int min_prime(φ)
{
    k = n;
    while (k≥0) {
        φ = φ ∪ {∑ y_j ≤ k};
        status = solve_sat(φ);
        φ = φ − {∑ y_j ≤ k};
        if (status == SATISFIABLE) {
            k = ∑ y_j;
            --k;
        } else { ++k; break; }
    return k;
}
```

Figure 1: SAT-based linear search algorithm

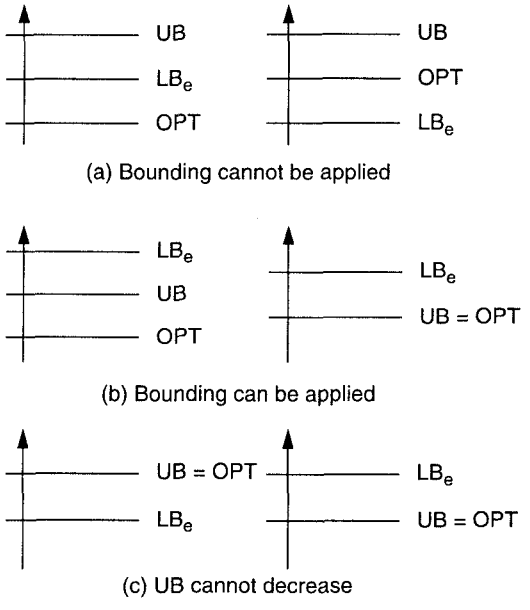

Figure 2: Using bounding in the ILP algorithm

Figure 1. For our particular case, the solve_sat function call invokes the GRASP SAT algorithm [12].

4.2 SAT-Based Branch and Bound Algorithm

A different algorithmic organization consists of using a variation of the branch and bound procedure [7], where upper bounds to the cost function (1) are identified and lower bounds to the current set of variable assignments are estimated. In our implementation, we have used the lower bound estimation procedures described in [4].

The operation of bounding for the proposed procedure is illustrated in Figure 2. Let UB denote the lowest *computed* upper bound on the solution of (2), LB_e denote an *estimated* lower bound on the solution of (2) and OPT denote the solution of (2). If the estimated lower bound is less than the already computed upper bound (as shown in Figure 2-(a)), then the search cannot be bound since it may still be possible to reduce the value of the upper bound. Clearly, the search can be bound whenever the estimated lower bound to the value of (1) is larger than or equal to the existing upper bound to the value of (1), as illustrated in Figure 2-(b). Finally, observe that Figure 2-(c) denotes the conditions after which the upper bound will no longer be updated during the search.

Moreover, since the branch and bound procedure is embedded in the SAT algorithm, every pruning technique used by the SAT algorithm can also be used in solving the ILP. This is particularly useful whenever a constraint of (2) becomes unsatisfied. Consequently, the branch and bound procedure consists of the following steps:

1. Initialize the upper bound to highest possible value. (Valid ILPs must correspond to instances of the minimum-size prime implicant problem.)
2. If no decision can be made (i.e. a solution to the constraints has been identified), then compute an upper bound on the minimum value of the cost function of the ILP formulation. Update current upper bound and issue a conflict to guarantee that the search is bound. Otherwise, branch on a given decision variable (i.e. make decision assignment).
3. Apply boolean constraint propagation [16]. If a conflict is reached, then diagnose conflict, record relevant clauses, and proceed with the search process or backtrack if required.
4. Estimate lower bound. If this value is larger than or equal to the current upper bound, then issue a conflict, diagnose the conflict, backtrack, and continue the search from step 2.

The pseudo-code for the algorithm is shown in Figure 3. Observe that the proposed branch and bound SAT-based ILP algorithm has the following main differences with respect to the linear search ILP algorithm:

- No clauses involving the cost function are created. The exception occurs when the estimated lower bound is no less than the computed upper bound. In this situation a clause involving some of the literals in the cost function is temporarily created, thus causing the search procedure to backtrack. (See [12] for details of the backtrack search SAT algorithm.)
- Lower bounding procedures are required. As mentioned earlier, the lower bounding procedures of [4] are used, but lower bounding procedures based on ***linear-programming relaxations*** or ***Lagrangian relaxations***

```
int bsolo(φ)
{
    _UB = ∞;
    while (TRUE) {
        if (Solution_found() ||
            Decide() != DECISION) {
            Update_UB();
            Issue_UB_based_conflict();
        }
        while (Deduce() == CONFLICT) {
            if (Diagnose() == CONFLICT) {
                return _UB;
            }
        }
        while (Estimate_LB() ≥ _UB) {
            Issue_LB_based_conflict();
            if (Diagnose() == CONFLICT) {
                return _UB;
            }
        }
    }
}
```

Figure 3: SAT-based branch and bound algorithm

can also be used [3, 7]. Clearly, the tightness of the lower bounding procedure is crucial for the efficiency of the branch and bound procedure.

4.3 Extensions to the Basic ILP Algorithms

One extension to the proposed ILP algorithms is the ability to incrementally enumerate prime implicants by increasing size [9]. The procedure proposed in [9] basically recreates the search for each new prime implicant to be computed. Clearly, this solution can introduce significant and unnecessary computational overhead. One possible improvement is based on the SAT-based linear search ILP algorithm of Figure 1, and is organized as follows:

1. Keep a stack of pairs of computed solutions and associated upper bound values k.
2. Use the current top of the stack to find the next minimum-size prime implicant.
3. For a given solution-upper bound pair k, apply the algorithm of Figure 1 until the next optimal solution is found. For this new optimal bound, enumerate all solution assignments.
4. As soon as a given pair solution-upper bound yields no more solutions, pop the stack and go back to step 2. Repeat until stack of solution-upper bound pairs becomes empty.

Since for each prime implicant size only part of the search space is visited, the above algorithm ensures a better worst-case time complexity than the algorithm of [9].

5 Experimental Results

In this section we include experimental results of two tools for computing minimum-size prime implicants, *min-prime* [14] and *bsolo*. *min-prime* is based on linear search through the possible values of the cost function as described in Section 4.1, whereas *bsolo* uses the SAT-based branch and bound algorithm as described in Section 4.2. We also compare these two SAT-based ILP algorithms with other ILP solvers, *lp-solve* [3], *opbdp* [2], and the commercial optimization tool ***CPLEX***. Moreover, the binate covering tool *scherzo* [4] is also evaluated, since minimum-size prime implicant computation can also be viewed as a restricted form of the binate covering problem. For this purpose we selected a set of satisfiable instances of the DIMACS benchmarks [6], from most of the available classes of instances. The CPU times, obtained on a SUN 5/85 machine with 64 MByte of physical memory, are shown in Table 1 and Table 2, where Table 2 includes the results for the SAT-based algorithms. For each benchmark and for each tool 3000 seconds of CPU time were allowed. Column **min** indicates the size of the minimum-size prime implicant, when this size is known. (Observe that for some of the benchmarks the minimum size prime implicant is still unknown.) Table 3 and Table 4 include the upper bound on the minimum size prime implicant computed by each algorithm for each benchmark. When each tool terminates, it reports the minimum size prime implicant if it was identified, otherwise the lowest computed upper bound is reported provided at least one upper bound was identified. For the results shown, whenever a tool quits earlier than 3000 sec, then the tool exceeded the available virtual memory (i.e. 64 MByte).

As can be concluded, general-purpose ILP solvers, such as *CPLEX* and *lp-solve*, may be inadequate for computing minimum-size prime implicants. Similarly, despite the very promising results as an algorithm for solving binate covering problems [4], *scherzo* performs particularly poorly when computing minimum-size prime implicants. The three SAT-based ILP solvers can handle a large number of benchmarks and, in general, *min-prime* and *bsolo* perform better and are more robust than *opbdp*, which is unable to find the solution on a larger number of instances. For the JNH benchmarks, *opbdp* performs better because the amount of search is similar and the overhead of the underlying GRASP SAT algorithm is larger. One key drawback of *min-prime* derives from using an ILP layer around the SAT algorithm which creates large additional clauses. For the minimum-size prime implicant

Benchmark	min	*CPLEX*	*lp-solve*	*scherzo*
aim-50-1_6-y1-1	50	116.50	> 3,000	2.33
aim-50-2_0-y1-2	50	109.50	> 3,000	5.65
aim-50-3_4-y1-3	50	62.90	377.10	0.57
aim-50-6_0-y1-4	50	26.90	96.80	0.73
aim-100-1_6-y1-2	100	> 3,000	> 3,000	> 1,000
aim-100-2_0-y1-3	100	> 3,000	> 3,000	691.57
aim-100-3_4-y1-4	100	> 3,000	> 3,000	35.47
aim-100-6_0-y1-1	100	294.30	> 3,000	2.78
aim-200-1_6-y1-3	200	> 3,000	> 3,000	> 345
aim-200-2_0-y1-4	200	> 3,000	> 3,000	> 1,705
aim-200-3_4-y1-1	200	> 3,000	> 3,000	> 3,000
aim-200-6_0-y1-2	200	> 3,000	> 3,000	619.38
ii8a1	54	63.30	786.90	0.98
ii8b2	—	> 3,000	> 3,000	> 3,000
ii8c2	—	> 3,000	> 3,000	> 3,000
ii8d2	—	> 3,000	> 3,000	> 3,000
ii8e2	—	> 3,000	> 3,000	> 3,000
jnh1	92	> 3,000	> 3,000	70.00
jnh7	89	> 3,000	> 3,000	5.35
jnh12	94	2,529	> 3,000	3.07
jnh17	95	873.90	> 3,000	17.28
ssa7552-038	—	> 3,000	> 3,000	> 223

Table 1: CPU times on selected benchmarks

Benchmark	min	*opbdp*	*min-prime*	*bsolo*
aim-50-1_6-y1-1	50	0.09	0.05	0.07
aim-50-2_0-y1-2	50	0.64	0.02	0.04
aim-50-3_4-y1-3	50	0.40	0.08	0.18
aim-50-6_0-y1-4	50	0.48	0.07	0.17
aim-100-1_6-y1-2	100	> 3,000	0.09	0.22
aim-100-2_0-y1-3	100	42.45	0.17	0.45
aim-100-3_4-y1-4	100	0.81	0.47	1.08
aim-100-6_0-y1-1	100	0.18	0.32	0.52
aim-200-1_6-y1-3	200	> 3,000	0.22	0.76
aim-200-2_0-y1-4	200	> 3,000	0.83	2.60
aim-200-3_4-y1-1	200	41.84	4.32	2.31
aim-200-6_0-y1-2	200	4.59	3.58	2.76
ii8a1	54	1.93	861.53	3.51
ii8b2	—	> 3,000	> 3,000	> 3,000
ii8c2	—	> 3,000	> 3,000	> 3,000
ii8d2	—	> 3,000	> 3,000	> 3,000
ii8e2	—	> 3,000	> 3,000	> 3,000
jnh1	92	2.24	17.96	11.39
jnh7	89	0.45	9.06	2.88
jnh12	94	0.12	0.58	1.10
jnh17	95	0.30	2.53	2.13
ssa7552-038	—	> 3,000	> 1,205	> 500

Table 2: CPU times on selected benchmarks

problem, these additional clauses involve *all* variables in the problem representation. Hence, conflicts involving this clause necessarily lead to chronological backtracking[2], and so the most useful features of GRASP cannot be exploited. Finally, we note that *bsolo* tends to be a more efficient search algorithm than *min-prime*, as the experimental results suggest.

From the obtained experimental results, it can also be concluded that the computation of the minimum-size prime implicant can be a particularly hard problem for specific sets of instances. This is the case, for example, with the ii8 and ssa7552 benchmarks.

2. In such a situation, each conflict involves *all* variables and so backtracking is necessarily chronological, to the most recent decision assignment [12].

6 Conclusions

In this paper we describe a new model and algorithms for computing minimum size prime implicants of boolean functions. The model is based on an ILP formulation and the proposed algorithms are built on top of existing SAT solvers. To our best knowledge *min-prime* and *bsolo* are the first SAT-based ILP algorithms that incorporate conflict diagnosis techniques [12] in solving optimization problems. Both *min-prime* and *bsolo* incorporate several powerful search-pruning techniques which are known to be particularly useful for SAT algorithms, in particular the non-chronological backtracking strategy, clause (nogood) recording procedures, and identification of necessary

Benchmark	min	*CPLEX*	*lp-solve*	*scherzo*
aim-50-1_6-y1-1	50	50	—	50
aim-50-2_0-y1-2	50	50	—	50
aim-50-3_4-y1-3	50	50	50	50
aim-50-6_0-y1-4	50	50	50	50
aim-100-1_6-y1-2	100	—	—	—
aim-100-2_0-y1-3	100	—	—	100
aim-100-3_4-y1-4	100	—	—	100
aim-100-6_0-y1-1	100	100	—	100
aim-200-1_6-y1-3	200	—	—	—
aim-200-2_0-y1-4	200	—	—	—
aim-200-3_4-y1-1	200	—	—	—
aim-200-6_0-y1-2	200	—	—	200
ii8a1	54	54	54	54
ii8b2	—	388	474	—
ii8c2	—	629	668	—
ii8d2	—	588	—	—
ii8e2	—	653	—	—
jnh1	92	93	—	92
jnh7	89	90	—	89
jnh12	94	94	—	94
jnh17	95	95	—	95
ssa7552-038	—	1449	1450	—

Table 3: Computed upper bounds

Benchmark	min	*opbdp*	*min-prime*	*bsolo*
aim-50-1_6-y1-1	50	50	50	50
aim-50-2_0-y1-2	50	50	50	50
aim-50-3_4-y1-3	50	50	50	50
aim-50-6_0-y1-4	50	50	50	50
aim-100-1_6-y1-2	100	100	100	100
aim-100-2_0-y1-3	100	100	100	100
aim-100-3_4-y1-4	100	100	100	100
aim-100-6_0-y1-1	100	100	100	100
aim-200-1_6-y1-3	200	—	200	200
aim-200-2_0-y1-4	200	—	200	200
aim-200-3_4-y1-1	200	200	200	200
aim-200-6_0-y1-2	200	200	200	200
ii8a1	54	54	54	54
ii8b2	—	—	379	379
ii8c2	—	—	525	525
ii8d2	—	—	540	540
ii8e2	—	—	494	494
jnh1	92	92	92	92
jnh7	89	89	89	89
jnh12	94	94	94	94
jnh17	95	95	95	95
ssa7552-038	—	1452	1448	1448

Table 4: Computed upper bounds

assignments.

As the results given in the previous section clearly show, the proposed algorithms, *min-prime* and *bsolo*, are by far the most competitive for the set of benchmarks considered. Thus, for these classes of benchmarks either *min-prime* or *bsolo* would be the option of choice. Moreover, as the experimental results show, the branch and bound SAT-based ILP algorithm *bsolo* is in general more efficient than the SAT-based linear search ILP algorithm.

Despite the promising results given in the previous section, more work needs to be done before an ILP algorithm can be used for solving instances of the minimum-size prime implicant problem which have a practical impact. Furthermore, observe that even though both the minimum size prime implicant problem and the binate covering problem have similar formulations, the algorithms for solving each problem should have different organizations, as the experimental results of *min-prime* [14], *bsolo* and *scherzo* [4] clearly suggest.

As mentioned earlier, one key bottleneck of the proposed *min-prime* algorithmic solution is the ILP layer around GRASP. The repeated addition of large clauses to the CNF formula reduces the ability of GRASP for backtracking non-chronologically and in several cases causes GRASP to always backtrack chronologically. In contrast, with *bsolo*, since no large clauses are created, non-chronological backtracking can in general be exercised.

One other improvement to the proposed algorithmic framework consists of iterating the computation of prime implicants by increasing size. This requires implementing

the procedure described in Section 4.3. Its usefulness depends on the target number of prime implicants. If all prime implicants are required, this approach clearly becomes impractical when the total number of prime implicants is exponential on the number of variables.

Finally, other realizations of the SAT-based branch and bound ILP algorithm can be envisioned and used for computing minimum-size prime implicants. These other realizations utilize different bounding procedures. We are currently experimenting with bounding procedures based on LP-relaxations [3, 7] and on Lagrangian relaxations [1, 7].

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.

[2] P. Barth, "A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization," Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, January 1995.

[3] M. R. C. M. Berkelaar, UNIX^{TM} Manual Page of lp-solve. Eindhoven University of Technology, Design Automation Section, 1992.

[4] O. Coudert, "On Solving Covering Problems," in *Proceedings of the Design Automation Conference*, June 1996.

[5] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of the Association for Computing Machinery*, vol. 7, pp. 201-215, 1960.

[6] D. S. Johnson and M. A. Trick (eds.), *Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1993. DIMACS benchmarks available in ftp://Dimacs.Rutgers.EDU/pub/challenge/sat/benchmarks/cnf.

[7] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, 1988.

[8] T. Ngair, "A New Algorithm for Incremental Prime Implicant Generation," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1993.

[9] C. Pizzuti, "Computing Prime Implicants by Integer Programming," in *Proceedings of International Conference on Tools with Artificial Intelligence*, November 1996.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 1994.

[11] T. Schiex and G. Verfaillie, "Nogood Recording for Static and Dynamic Constraint Satisfaction Problems," in *Proceedings of the International Conference on Tools with Artificial Intelligence*, pp. 48-55, 1993.

[12] J. P. M. Silva and K. A. Sakallah, "Conflict Analysis in Search Algorithms for Propositional Satisfiability," in *Proceedings of the International Conference on Tools with Artificial Intelligence*, November 1996.

[13] J. P. M. Silva and A. L. Oliveira, "Improving Satisfiability Algorithms with Dominance and Partitioning," in *International Workshop on Logic Synthesis*, May 1997.

[14] J. P. M. Silva, "On Computing Minimum Size Prime Implicants," in *International Workshop on Logic Synthesis*, May 1997.

[15] J. P. M. Silva, GRASP source code, available from ftp://algos.inesc.pt/pub/users/jpms/soft/grasp/.

[16] R. Zabih and D. A. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 155-160, 1988.