Full CNF Encoding: The Counting Constraints Case

Olivier Bailleux[1] and Yacine Boufkhad[2]

[1] LERSIA, Université de Bourgogne
Avenue Alain Savary, BP 47870 21078 Dijon Cedex
olivier.bailleux@u-bourgogne.fr
[2] LIAFA, Université Paris 7
Case 7014 - 2, place Jussieu F-75251 Paris Cedex 05
Yacine.Boufkhad@liafa.jussieu.fr

Abstract. Many problems are naturally expressed using CNF clauses and boolean cardinality constraints. It is generally believed that solving such problems through pure CNF encoding is inefficient, so many authors has proposed specialized algorithms : the pseudo-boolean solvers. In this paper we show that an appropriate pure CNF encoding can be competitive with these specialized methods. In conjunction with our encoding, we propose a slight modification of the DLL procedure that allows any DLL-based SAT solver to solve boolean cardinality optimization problems. We show experimentally that our encoding allows zchaff to be competitive with pseudo-boolean solvers on some decision and optimization problems.

1 Introduction

The increasing efficiency of SAT solvers and their highly optimized implementations make them good candidates for solving NP-complete problems using CNF representation. The idea of encoding NP-Complete problems in CNF follows directly from Cook's theorem. This approach requires much less development time than implementing a dedicated solver that reaches the high performances of modern SAT solvers. It also benefits from future improvements of SAT solvers with no additional cost. Moreover, problems that mix CNF clauses and other types of constraints, such as arithmetic constraints, can be expressed through encoding to a full CNF representation allowing modern SAT techniques (like clause learning) to be fully functional without need of adapting them within a mixed ad-hoc solver. However, the practical interest of such an approach can be limited both by the size and the difficulty of the resulting CNF formula.
The present study focuses on the problems that can be expressed by propositional clauses and/or counting constraints, i.e., contraints that impose upper and lower bounds to the number of variables fixed to 1 in a given set of Boolean variables.
Various methods can be used for solving these problems, such as 0-1 integer linear programming and pseudo-Boolean solvers using SAT solving techniques with non clausal representation of linear constraints [2]. All theses methods solve problems that can be expressed in the form of linear inequalities of the type : $a_1x_1 + a_2x_2 ..... + a_nx_n \leq b$. In this paper we deal only with the restriction to the counting constraint, namely the case where $a_i = 1$.
A method for encoding in CNF linear integer inequalities was given by Warner in [7]. In [1], Warner's CNF encoding of linear integer inequalities is compared with the PBS pseudo Boolean solver on global routing problems. Despite the conciseness of the encoding and the use of the state-of-the-art zchaff SAT solver, PBS clearly outperforms the full CNF approach based on Warner's encoding. Opposingly, [3] shows that an appropriate CNF encoding of counting constraints that preserves generalized arc consistency through unit propagation, can be competitive with a commercial constraint programming system, and even with a dedicated solver, on discrete tomography problems. These results show that it is somewhat hard to claim a ground rule about the full CNF approach of solving counting constraints.
This paper brings two contributions.
First, we show that thanks to the efficient CNF encoding presented in [3], using a full CNF approach can be competitive with PBS on problems specified with propositional clauses and/or counting constraints and solves instances that PBS can not solve. On some benches, this encoding also clearly outperforms Warner's linear-space encoding, in spite of the greater size of the generated formulae.
Second, we present a DLL based optimization procedure that maximizes the number of variables fixed to 1 in a given subset of the variables belonging to a CNF formula, while satisfying the formula. This

optimization procedure consists in a slight modification of the DLL procedure that works in conjunction with the CNF encoding of counting constraints presented in [3]. This transformation can be easily applied to any DLL based SAT solver, including the ones that use clause learning. It allows problems like MAX-ONEs or MAX-SAT, that were not previously included in the field of SAT solver applications, to be handled by slightly modified existing SAT solvers. Experimental results show that zchaffopt, the modified version of zchaff can be competitive with the PBS optimizer and 0-1 ILP solver OPBDP on the MAX-ONEs and MAX-SAT problems.
These results attest that, using an efficient CNF encoding allowing unit propagation of DLL based solvers to restore the generalized arc consistency, the full CNF approach is a serious alternative to dedicated solvers for handling counting constraints. They also suggest that the full CNF approach could be successfully used for many other problems based on global constraints, subject to the design of optimized encoding schemes.

2 Encoding cardinality constraint

2.1 Warner's encoding applied to counting constraints

In [7], a linear-time transformation of linear inequalities into CNF is presented. The linear inequalities are of this type $a_1x_1 + a_2x_2 + ... + a_mx_m < b$ where a_i is an integer and x_i is a boolean variable. The counting constraint is the particular case where every coefficient $a_i = 1$. The principle of this encoding can be defined inductively by $v_{i+1} = v_i + x_i$ and $v_0 = 0$. Every integer variable v_i is represented by some boolean variables as its binary representation. The relations $v_{i+1} = v_i + x_i$ are then encoded in a standard manner into clauses between boolean variables in the binary representations of v_i, v_{i+1} and x_i. The problem with this encoding is that unit propagation does not maintain the generalized arc consistency in all cases. In the next subsection, we introduce a CNF encoding that maintains the generalized arc consistency whatever is the order of instantiation of variables.

2.2 The UT-MGAC encoding

For sake of readability, the CNF encoding of Boolean cardinality constraints introduced in [3] will be called UT-MGAC (for Unit Totalizer Maintaining Generalized Arc Consistency). Let us recall that this encoding uses a unary representation of integer intervals. Namely, a set of Boolean variables $x_0, \ldots, x_{\max}$ can represent any interval $\mu..\lambda$ in the range $0..\max$ by setting $x_0, \ldots, x_\mu$ to 1 and $x_{\lambda+1}, \ldots, x_{\max}$ to 0.
An adder based on this representation can deduce the interval where an integer c falls, given the intervals of integers a and b such that $c = a + b$. The resulting CNF formula allows unit propagation to derive all the consequences of every assignment with respect to generalized arc consistency. The encoding of a cardinality constraint then consists in a totalizer structured as a pyramidal adder network, extended by unary clauses that restrict the possible output values.
The encoding clauses and additional encoding variables can be generated according to the figure 1, where each adder with inputs $a_1, \ldots, a_{m_1}$, $b_1, \ldots, b_{m_2}$, and output $r_1, \ldots, r_m$ is encoded as

$$\bigwedge_{\substack{0\leq\alpha\leq m_1\\ 0\leq\beta\leq m_2\\ 0\leq\sigma\leq m\\ \alpha+\beta=\sigma}} (C_1(\alpha,\beta,\sigma) \wedge C_2(\alpha,\beta,\sigma)) \tag{1}$$

using the following notations:

$$a_0 = b_0 = r_0 = 1, a_{m_1+1} = b_{m_2+1} = r_{m+1} = 0$$

$$C_1(\alpha,\beta,\sigma) = (\overline{a_\alpha} \vee \overline{b_\beta} \vee r_\sigma),\ C_2(\alpha,\beta,\sigma) = (a_{\alpha+1} \vee b_{\beta+1} \vee \overline{r_{\sigma+1}})$$

In the unary representation of an integer a, when the bit a_α is equal to 1, the integer $a \geq \alpha$ and conversely if $a_\alpha = 0$ then $a < \alpha$. Given this, the clause $(\overline{a_\alpha} \vee \overline{b_\beta} \vee r_\sigma)$, which will be called of type C_1, ensures that the three following inequalities can not be false at the same time : $a < \alpha$, $b < \beta$, $r \geq \alpha + \beta$ and the clause $(a_{\alpha+1} \vee b_{\beta+1} \vee \overline{r_{\sigma+1}})$, which will be called of type C_2, ensures that at least one of the three following inequalities is true : $a > \alpha$, $b > \beta$, $r \leq \alpha + \beta$.
Now let n be the number of input and output variables of the totalizer, and Q be the cardinality constraint $\mu \leq N \leq \lambda$, where N is the number of input variables of the totalizer that can be fixed to one according to

Q. Q will be achieved by additional unit clauses enforcing the output variables to match the interval $\mu..\lambda$. Namely,

$$\bigwedge_{1\leq i\leq \mu} (s_i) \quad \bigwedge_{\lambda+1\leq j\leq n} (\overline{s_j}) \tag{2}$$

As proved in [3], given any partial truth assignment of the input variables, the C_1 and C_2 clauses allow unit propagation to restore the generalized arc consistency of Q. This encoding requires $\Omega(n^2)$ clauses and $\Omega(nlogn)$ additional variables. Without loss of the correctness and filtering properties, its effective size can be optimized by bounding to $\lambda+1$ the number of output variables of any adder in the totalizer.

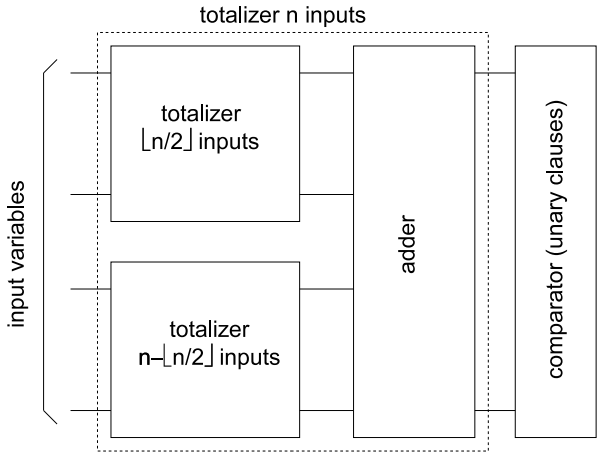

Fig. 1. UT-MGAG encoding scheme for Boolean cardinality constraints

2.3 Comparison of encodings

We compared the UT-MGAC encoding with warners [7] encoding and with PBS [1, 2], which includes counting constraints without encoding them at all. We have done this comparison on the tomography problems described in [3] which have many counting constraints. The encoded problems have been solved using zchaff. Table 1 lists the CPU times on a Pentium IV 1.3 GHz computer and shows clearly that this UT-MGAC enconding outperforms warner's and the PBS solver on the tomography problems.

Name	Zchaff UT-MGAC	Zchaff Warners	PBS
mouse20	0.21	74	0.07
mouse22	111	–	–
letter18	22.7	70	8.12
letter20	220	–	–
rand20-1	6	69	–
rand20-2	6.3	0.21	–

Table 1. Comparison between UT-MGAC encoding, Warners encoding and PBS on tomography instances, 1000s is set as time limit

3 Optimization DLL

At this point, we present, a slight modification to the DLL procedure that allows it, in conjunction with the encoding of cardinality constraint presented in section 2.2, to optimize the number of ones in some objective set of variables. The recursive procedure is presented in Algorithm 1. It is applied to a set of variables including the CNF encoding of the totalizer described in section 2.2. It output is denoted by the tuple $(u_1, u_2, ..., u_m)$. The goal is to find a solution such that the it maximizes the integer having unary representation $u_1, u_2, ..., u_n$.

Example : MAXONES problem

Given a CNF formula Φ built upon a set of variables V, find an assignment to the variables of Φ that satisfies it and that have the maximum number of ones.

We use the encoding of the totalizer described in section 2.2. Let $\mathbf{Tot}(V)$ *be the CNF encoding of the totalizer that have V as input. The output is the tuple $(u_1, u_2, ..., u_m)$. The final formula that is to be solved by the subsequent algorithm is then $\Phi \wedge \mathbf{Tot}(V)$. Note that any solution of $\Phi \wedge \mathbf{Tot}(V)$, restricted to the variables in V, is a solution of Φ and $u_1 u_2 ... u_m$ is the unary representation of the sum of ones in that solution restricted to the variables in V.*

The procedure presented in Algorithm 1 is called with the empty set as an argument. I is a consistent set of literals i.e. it does not contain a literal and its negation. I represents the partial truth assignment obtained by assigning true to every literal in it. $F|_I$ represents the formula obtained from F by applying the usual filtering techniques (unit propagation and eventually other rules) after assigning the value true to every literal in I. In contrast with usual DLL procedure, **DLLopt** does not stop when it finds a solution but continues the search to find new solutions. The optimization is here performed by simulating a branch and bound method that consists in adding some unit clauses each time a new solution is found. The variable *max* records the maximum number of ones found so far plus one. It represents the goal required for the next solution.

Algorithm 1 A DLL that optimizes the number of 1's in some set of variables. C is the set of clauses the tuple $(u_1, u_2, ..., u_n)$ represents the output of the objective function. *max* is initialized to 0 and $I = \emptyset$

```
 1: DLLopt(I)
 2: if F|_I is a contradiction then
 3:    return;
 4: else
 5:    if every variable of F is assigned in I (i.e. I is a solution) then
 6:       for every literal u_i ∈ I such that i > max do
 7:          F ← F ∧ u_i;
 8:          Increment max;
 9:       end for
10:       Increment max;
11:       F ← F ∧ u_max;
12:       return;
13:    else
14:       select a new variable v such that v ∉ I and v̄ ∉ I
15:       DLLopt(I ∪ {v});
16:       DLLopt(I ∪ {v̄});
17:    end if
18: end if
```

3.1 Experimental results on optimization problems

In this section, we report the results on two sets of benchmarks MAX-ONES and MAX-SAT. The instances are built from representative benchmarks from the SATLIB database and from random instances. We have modified zchaff [6] according to the **DLLopt** algorithm. Then we have compared the resulting program zchaffopt against the pseudo-boolean solver PBS [2] and the 0-1 ILP solver OPBDP [5]. We used the

default settings for these programs(-D 1 option for PBS). The experiments were conducted on a Pentium 1.5GHz computer running under linux and with 1GByte RAM. Each run was allowed a maximum of 5000s.

MAX-ONES : Table 2 lists the results of MAX-ONEs experiment. The size of the formulas generated by using the UT-MGAC encoding is reported for each instance. For each group of benchmarks, only those that at least one of the the programs solves are reported. For those that have a large number of instances like graph coloring ones, only the first three instances are reported. The first remark is that zchaffopt solve many instances that the one or both of the other solvers does not within the time limit. On the instances that are easy for all the solvers, zchaffopt is slower because it needs to handle a large formula. The random instances are generated at the hardest point of DISTANCE-SAT random instance [4].

MAX-SAT : MAX-SAT is encoded by adding a negated new variable to each clause and by maximizing the sum of the additional variables for zchaffopt and OPBDP. For PBS it is encoded as described in [1]. As in [1], a preprocessing is made using WalkSat to compute a lower bound on the number of satisfied clauses. In the cases where only one clause is unsatisfied, the problem amounts to prove that the instance is unsatisfiable which is done very efficiently with zchaff. We choose not to report the result in theses cases. Conversely, the difficulty of the problem increases with the number of unsatisfied clauses. In all cases the bound given by WalkSat is optimal and then the problem amounts to prove the optimality of this bound. Table 3 lists the results of MAX-SAT experiment.zchaffopt and OPBDP have almost the same performances and both are faster than PBS on these instances.

4 Conclusion

We gave the experimental evidence that full CNF encoding is a competitive option for solving counting constraints. We have shown that an appropriate encoding and slight modifications to DLL can make zchaff to solve the problems usually reserved to pseudo-boolean solvers or 0-1 ILP solvers.

Acknowledgments : We thank Fadi Aloul and Igor Markov for providing us with PBS code.

References

1. F. ALOUL, A. RAMANI, I. MARKOV, , AND K. SAKALLAH, *Generic ilp versus specialized 0-1 ilp: an update*, in International Conference on Computer Design (ICCD), 2002, pp. 108–122.
2. F. ALOUL, A. RAMANI, I. MARKOV, AND K. SAKALLAH, *Pbs: A backtrack search pseudo-boolean solver*, in Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002), 2002, pp. 346–353.
3. O. BAILLEUX AND Y. BOUFKHAD, *Efficient cnf encoding of boolean cardinality constraints*, in Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming, CP 2003, vol. 2833, LNCS.
4. O. BAILLEUX AND P. MARQUIS, *DISTANCE-SAT: Complexity and algorithms*, in Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99), Orlando, Florida, 1999, pp. 642–647.
5. P. BARTH, *A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization, technical report mpi-i-95-2-003*, tech. report, Max-Planck-Institut Fr Informatik, 1995.
6. M. MOSKEWICZ, C. MADIGAN, Y. ZHAO, L. ZHANG, AND S. MALIK, *Chaff: Engineering an efficient sat solver*, in 39th Design Automation Conference, June 2001.
7. J. P. WARNERS, *A linear-time transformation of linear inequalities into conjunctive normal form*, Information Processing Letters, 68 (1998), pp. 63–69.

		Original formula		After encoding		CPU time		
Group	**Name**	**#var**	**#cla**	**#var**	**#cla**	**zchaffopt**	**PBS**	**OPBDP**
Inductive Inference	**ii8a1**	66	186	466	5276	0.03	0.04	0.01
	ii8a2	180	800	1544	35748	179	780	55
	ii8a3	264	1552	2392	75240	4413	–	2800
Graph 3-Coloring	**flat200-1**	600	2237	6176	372789	510	–	–
	flat200-2	600	2237	6176	372789	39	–	–
	flat150-1	450	1680	4438	211706	4	–	180
	flat150-2	450	1680	4438	211706	6	–	–
	flat75-1	225	840	1994	54778	0.2	2	0.23
	flat75-2	225	840	1994	54778	0.2	612	6.2
Graph 5-Coloring 100 vertices	**sw100_8_0_1**	500	3100	4988	261576	9	–	–
	sw100_8_0_2	500	3100	4988	261576	10	–	–
	sw100_8_4_1	500	3100	4988	261576	1568	–	–
	sw100_8_4_2	500	3100	4988	261576	421	–	–
	sw100_8_8_1	500	3100	4988	261576	2	4.2	–
	sw100_8_8_2	500	3100	4988	261576	2.5	0.19	–
Quasigroups	**qg5-11**	1331	64054	15255	1862132	10	0.6	–
	qg6-09	729	21844	7724	566546	2	0.13	–
	qg7-09	729	22060	7724	566762	1.5	0.1	–
Planning	**3blocks**	283	9690	2601	94132	1	0.6	483
	4blocks	758	47820	8072	636254	2095	291	–
	4blocksb	410	24758	3998	199624	1	0.8	4712
Bejing	**bw.a**	459	4675	4537	223053	0.3	0.03	0.2
	bw.b	1087	13772	12083	1216246	5	0.3	2.3
	bw.c	3016	50457	38128	9213921	112	4	–
Random instances	mean	50	100	336	3122	0.01(100)	0.04(100)*	0.01(100)*
	on 100 3CNF	75	150	547	6644	0.7(100)	12(100)*	0.3(100)*
	instances	100	200	772	11444	10.3(100)	62(18)*	3.5(100)*

* : Between () the number of solved instances. The mean CPU is computed only over solved instances
- : signifies not solved after 5000s or out of memory

Table 2. CPU time for MAX-ONEs on some benchmarks and randomly generated instances

	Original formula		After encoding*		CPU time		
Name	**#var**	**#cla**	**#var**	**#cla**	**zchaffopt**	**PBS**	**OPBDP**
jnh08	100	850	7406	181516	0.32	4	0.74
jnh09	100	850	7406	181516	0.43	4.4	0.49
jnh13	100	850	7406	181516	0.57	8	0.62
jnh14	100	850	7406	181516	0.35	4	0.62
jnh15	100	850	7406	181516	0.6	5.8	0.98
jnh19	100	850	7406	181516	0.89	10.8	0.53
jnh211	100	800	6990	170268	0.28	3	0.34
jnh302	100	900	7806	192716	63	–	29
jnh303	100	900	7806	192716	56	3032	15
jnh304	100	900	7806	192716	7.5	41	3.25
jnh305	100	900	7806	192716	14	412	5.9
jnh307	100	900	7806	192716	4.1	58	4.6
jnh308	100	900	7806	192716	0.8	12	0.54
jnh309	100	900	7806	192716	0.4	4	0.17
jnh310	100	900	7806	192716	10.1	224	2.8

*: size of the formula using UT-MGAC for solving with zchaffopt

Table 3. CPU time on MAX-SAT for JNH unsatisfiable instances from the DIMACS database.