Aligning CNF- and Equivalence-Reasoning

Marijn Heule[*] and Hans van Maaren

Department of Information Systems and Algorithms,
Faculty of Electrical Engineering, Mathematics and Computer Sciences,
Delft University of Technology
marijn@heule.nl, h.vanmaaren@its.tudelft.nl

Abstract. Structural logical formulas sometimes yield a substantial fraction of so called equivalence clauses after translation to CNF. Probably the best known example of this is the `parity`-family. Large instances of such CNF formulas cannot be solved in reasonable time if no detection of, and extra reasoning with, these clauses is incorporated. That is, in solving these formulas, there is a more or less separate algorithmic device dealing with the equivalence clauses, called equivalence reasoning, and another dealing with the remaining clauses. In this paper we propose a way to align these two reasoning devices by introducing parameters for which we establish optimal values over a variety of existing benchmarks. We obtain a truly convincing speed-up in solving such formulas with respect to the best solving methods existing so far.

1 Introduction

The notorious `parity-32` benchmarks [3] remained unsolved by general purpose SAT solvers for a considerable time. In [12] a method was proposed which, for the first time, could solve these instances in a few minutes. The key to this method was to detect the clauses that represented so called *equivalences* $l_1 \leftrightarrow l_2 \leftrightarrow \cdots \leftrightarrow l_n$ (where the l_i are literals, or their negations, appearing in the formula at hand) and to pre-process the set of these equivalences in such a way that dependent and independent variables became visible. The remaining clauses then were tackled with a rather straightforward DPLL procedure but in such a way that kept track of the role of these dependent and independent variables. It was developed as a two-phase method, where the equivalence part was established and transformed in a pre-processing phase.

The next important step was made by Li [6]. He incorporated a form of equivalence reasoning in every node of an emerging search tree. His approach did not incorporate a pre-processing phase (at least not regarding the equivalence clauses) and thus he established the first one-phase SAT solver `eqsatz` which could tackle these instances in reasonable time.

[*] Supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.306.

A disadvantage of his method is the fact that he uses a full look-ahead approach: all variables enter the look-ahead phase, contrary to partial look-ahead, which runs only on a pre-selected number of variables. Full look-ahead is costly for larger size formulas. In addition, his look-ahead evaluation function to measure the reduction of the formula during the look-ahead phase is - in our opinion - not optimal. Also, his equivalence reasoning is restricted to equivalences of length at most three.

Some years later Ostrowski *et al.* [10] extended the above pre-processing ideas from [12] to logical gates other than equivalences, resulting in the lsat solver. However, their DPLL approach to deal with the remaining CNF-part uses a Jeroslow-Wang branching rule and they do not perform a look-ahead phase, which is - again in our opinion - not an optimal alignment.

In this paper we propose an alignment of equivalence reasoning and DPLL reasoning which does not assume a *full* look-ahead approach. This will enforce us to introduce adequate pre-selection heuristics for selecting those variables which are allowed to enter an Iterative Unit Propagation phase. Further, we will evaluate the progress in enrolling the formula at hand in a more detailed manner as was done in eqsatz. We are forced to introduce parameters to guide this search. These parameters are needed to aggregate the reduction of the equivalence part of the formula and that of the remaining CNF part. Further, our method is able to deal with equivalences of arbitrary size. This in turn leads us to an investigation of the relative importance of equivalences of different size. Surprisingly, this relative importance turns out to be rather differently measured as would be expected when taking similar relative importance of ordinary clause-lengths as a guiding principle.

We optimise the various parameters to ensure a convincing speed-up in solving formulas with a substantial equivalence part, both with respect to the various alternative solvers available and with respect to a variety of benchmarks known of this type.

2 Equivalence Reasoning in Pre-processor

After initialisation, the first goal of the pre-processor is to simplify the formula as much as possible. This is achieved by iterative propagation of all unary clauses and binary equivalences. After this procedure, the equivalence clauses are detected using a simple syntactical search procedure, extracted from the formula and placed in a separate data-structure. We refer to this data-structure as the Conjunction of Equivalences (CoE). The aim of this equivalence reasoning enhanced pre-processor is to solve the extracted CoE sub-formula.

A solution is obtained by performing the first phase of the algorithm by Warners and Van Maaren [12]: We initialise set $\mathrm{I} = \{x_1, \cdots, x_m\}$, the set of *independent* variables, with m referring to the initial number of variables. We loop through the equivalency clauses once, selecting variable x_i in each one to eliminate from all other equivalence clauses. Subsequently we remove x_i from I, and call it a *dependent* variable. Thus we end up with a set of equivalence clauses for which all satisfiable assignments can be constructed by assigning all possible combinations of truth values to the independent variables. The values of the dependent variables are uniquely determined by an assignment of the independent variables. Note that during the elimination process a contradiction might be derived, which implies unsatisfiability of the original problem.

Numerous of such independent sets could be obtained by this algorithm. The performance of a solver might vary significantly under different choices of the independent set, as we have observed using the march solver (developed by Joris van Zwieten, Mark Dufour, Marijn Heule and Hans van Maaren; it participated in the Sat 2002 [7], Sat 2003 [8], and Sat 2004 [9] competitions). Therefore, two enhancements are added to the original algorithm to find an independent set that would result in relatively fast performance: the first addition involves an explicit prescription for the selection of the variables to eliminate: for every equivalence clause the variable is selected that occurs (in a weighted fashion) least frequently in the CNF. Occurrences in binary clauses are counted twice as important as occurrences in n-ary clauses.

The motivation for selecting the least occurring variable is twofold: first, if the selected variable x_i does not occur in the CNF at all, the equivalence clause in which x_i occurs, becomes a *tautological clause* after elimination, because x_i could always be taken as such to satisfy it. Neglecting tautological clauses during the solving phase could result in a considerable speed-up. Second, faster reduction of the formula is expected when the independent variables occur frequently in the CNF-part: independent variables will be forced earlier to a certain truth value by constraints from both the CoE- and the CNF-part.

The second addition is a procedure that reduces the sum of the lengths of all non-tautological equivalences in the CoE. This procedure consists of two steps: the first searches for pairs of equivalence clauses that could be combined to created a binary equivalence. Note that binary equivalence clauses can always be made tautological, since one of its literals could be removed from the CNF by replacing it by the other. The second step loops through all equivalence clauses and checks whether a different choice for the dependent variable in that clause would result in a smaller sum of lenghts of non-tautological equivalences. Both methods are iteratively repeated until both yield no further reduction.

Several benchmark families in the Sat 2003, Sat 2002 and Dimacs benchmark suites[1] can be solved by merely applying the pre-processing presented above. One of these families is `xor-chain` which contains the smallest unsolved unsatisfiable instances from the Sat 2002 competition. Table 1 shows the required time to solve these families for various solvers. Notice that march uses the proposed pre-processing. In the table, the numbers behind the family names refer to the number of instances in a family. The last five columns show the total time required to solve a family. In these columns, numbers between braces express the number of benchmarks that could not be solved within a 120 seconds time limit. Judging from the data in the table, lsat is the only solver which can compete with the march pre-processor since it solves all but one families in comparable time.

[1] All three suites are available at www.satlib.org

Table 1. Performances of the solvers march, eqsatz, satzoo, lsat and zchaff in seconds on several families that could be solved by merely pre-processing

family		contributer	suite	march	eqsatz	satzoo	lsat	zchaff
bevhcube	(4)	Bevan	SAT '03	0.02	2.64 (2)	2.74 (2)	0.02	0.01 (3)
dodecahedron	(1)	Bevan	SAT '03	0.01	0.01	0.08	0.01	0.01
hcb	(4)	Bevan	SAT '03	0.16	0.01 (3)	0.01 (3)	2.53	0.01 (3)
hypercube	(4)	Bevan	SAT '03	0.09	0.40 (3)	0.08 (3)	0.33	2.56 (3)
icos	(2)	Bevan	SAT '03	0.01	2.29 (1)	4.12 (1)	0.02	—— (2)
marg	(17)	Bevan	SAT '03	0.12	195.83 (5)	52.93 (5)	0.13	0.08 (11)
urqh	(26)	Bevan	SAT '03	0.19	102.58 (20)	16.76 (20)	0.47	1.19 (22)
hardmn	(18)	Moore	SAT '03	0.87	0.75	—— (18)	0.30	—— (18)
genurq	(10)	Ostrowski	SAT '03	0.95	0.07 (7)	0.68 (1)	0.57	0.39 (6)
Urquhart	(30)	Simon	SAT '02	0.27	—— (30)	58.09 (25)	0.01	0.13 (29)
urquhart	(6)	Chu Min Li	SAT '02	0.03	0.29 (4)	95.78 (3)	0.01	0.04 (5)
xor-chain	(27)	Zhang-Lintao	SAT '02	0.16	0.17 (25)	0.77 (25)	0.02	2.25 (24)
barrel	(9)	Biere	DIMACS	2.27	2.95	99.50 (1)	2.13 (4)	46.10 (1)
dubois	(13)	Dubois	DIMACS	0.02	0.1	0.75	0.01	0.06
pret	(8)	Pretolani	DIMACS	0.03	0.08	20.81	0.01	—— (8)

3 Combined Look-Ahead Evaluation

Look-ahead appears to be a powerful technique to solve a wide range of problems. The pseudo-code of an elementary look-ahead procedure is presented in Algorithm 1. The look-ahead procedure in march closely approximates this elementary procedure. Notice that it does not perform any equivalence reasoning during this phase.

Algorithm 1. LOOK-AHEAD( )

```
Let $\mathcal{F}'$ and $\mathcal{F}''$ be two copies of $\mathcal{F}$
for each variable $x_i$ in $\mathcal{P}$ do
  $\mathcal{F}'$ := ITERATIVEUNITPROPAGATION( $\mathcal{F} \cup \{x_i\}$ )
  $\mathcal{F}''$ := ITERATIVEUNITPROPAGATION( $\mathcal{F} \cup \{\neg x_i\}$ )
  if empty clause $\in \mathcal{F}'$ and empty clause $\in \mathcal{F}''$ then
    return "unsatisfiable"
  else if empty clause $\in \mathcal{F}'$ then
    $\mathcal{F} := \mathcal{F}''$
  else if empty clause $\in \mathcal{F}''$ then
    $\mathcal{F} := \mathcal{F}'$
  else
    $\mathrm{H}(x_i) = 1024 \times \mathrm{DIFF}(\mathcal{F}, \mathcal{F}') \times \mathrm{DIFF}(\mathcal{F}, \mathcal{F}'') + \mathrm{DIFF}(\mathcal{F}, \mathcal{F}') + \mathrm{DIFF}(\mathcal{F}, \mathcal{F}'')$
  end if
end for
return $x_i$ with greatest $\mathrm{H}(x_i)$ to branch on
```

An effective look-ahead evaluation function (DIFF in short) is critical to the effectiveness of the branching variable the look-ahead returns. Experiments on random 3-SAT instances showed that using a DIFF that counts newly created binary clauses results in fast performances on these benchmarks and many other families. Addition of new clauses of length > 2 to the DIFF requires weights that express the relative importance of clauses of various length. Weights that result in optimal performance on random k-SAT formulas could be described by linear regression: e.g. Kullmann [5] uses weights in his OKsolver that could be approximated by 0.22^{n-2}. In this equation n refers to the length of a clause, with $n \geq 2$.

Little is known about effective evaluation functions to measure the importance of a new equivalence clause. In eqsatz by Li [6] only new binary equivalences are counted. These are weighed twice as important as a new binary clause. The importance of the new equivalence clauses of various length could be obtained by measuring the reduction of its translation into CNF. Applying the approximation of the weights by Kullmann [5] results in a weight function of $2^{n-1} \times 0.22^{n-2} \approx 10.33 \times 0.44^n$ for a new equivalence of length n. However, this reference should be labelled as vague, since the weights are optimised with respect to random formulas.

Although we have indications that other models might be more appropriate when equivalence clauses are involved, we take this regression model as a first start. Performances were measured for various parameter settings of equation (1). In this equation, n refers to the reduced length of an equivalence clause. Parameter q_{base} denotes the factor that describes the decreasing importance of equivalence clauses of various length and parameter q_{const} expresses the relation between the reduction of the CNF-clauses and the equivalence clauses. Since march uses a 3-SAT translator, only new binary clauses are created. The evaluation of the look-ahead is calibrated by defining the importance of a new binary clause to value 1. The result of eq_n then defines the relative importance of a new equivalence clause of length n in relation to a new binary clause.

$$eq_n = q_{\text{const}} \times {q_{\text{base}}}^n \tag{1}$$

Wide scale experiments were troubled by the lack of useful benchmarks: many benchmark families that contain a significant part of equivalence clauses are easily solved with mere pre-processing procedures: either the solving procedure for the CoE results in a contradiction, or the propagation of the unary clauses and the binary equivalences found during pre-processing are sufficient to solve the formula. Many benchmarks families with a significant CoE-part that require a sophisticated solving procedure after pre-processing are neither useful for these experiments, because most or all of their equivalence clauses have length 3. For comparison: The SAT 2003 [8] competition suite consisted of 11 families which are solved in pre-processing while only five needed further search. Of those five only two had a large number of long equivalences after the pre-processing.

These two families are the `parity32` and the `hwb`. The first family consists of the SAT-encoding of minimal disagreement parity problems contributed by Crawford *et al.* [3]. The second consists of equivalence checking problems that arise by combining two circuits computed by the hidden weighted bit function. These latter are contributed by Stanion [8]. Both families have been used to determine

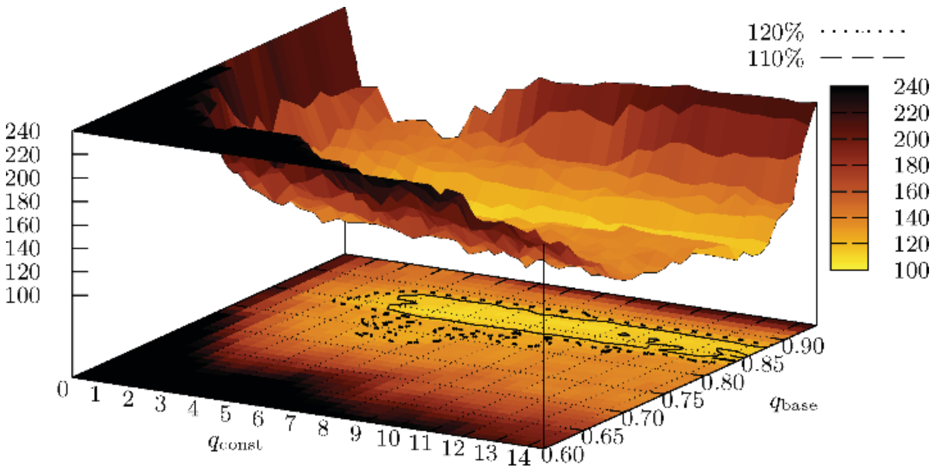

Fig. 1. Performances achieved by `march` on various settings of q_{base} and q_{const}. The values on the z-axis are the cumulated performances on the whole `parity-32` and `hwb-n20` families in seconds. Contour lines are drawn at 110% and 120% of optimal performance

Table 2. Weights to measure the reduction of equivalence clauses of various lengths

Reduced length (n):	2	3	4	5	6	7	8	9	10
CNF-reference:	2.00	0.88	0.39	0.17	0.07	0.03	0.01	0.01	0.00
Found optimum:	3.97	3.38	2.87	2.44	2.07	1.76	1.50	1.27	1.08

the parameter setting for equation (1) that results in optimal performance. The results of these experiments are shown in Fig. 1. During our experiments, the values $q_{\text{const}} = 5.5$ and $q_{\text{base}} = 0.85$ appeared optimal. Two conclusions can be derived regarding the results: (1) parameter q_{base} has a much larger influence on the performance than q_{const}. (2) Using optimal settings, the reduction of equivalences is considered far more important than the reduction of the equivalent CNF-translations would suggest: table 2 shows the weights used for both settings.

4 Pre-selection Heuristics

Although look-ahead is a powerful technique, it pays off to restrict the number of variables which enter this procedure. In Algorithm 1 this partial behaviour is achieved by performing only look-ahead on variables in set $\mathcal{P}$. At the beginning of each node, this set is filled by pre-selection heuristics that are based on an approximation function of the combined evaluation of the look-ahead (Ace). The ranking of variable x is calculated by multiplying $\text{Ace}(x)$ and $\text{Ace}(\neg x)$.

$\mathcal{E}(x)$, used to obtain $\textsc{Ace}(x)$, refers to the set of all equivalence clauses in which x occurs and $occ_3(x)$ refers to the number of occurrences of x in ternary clauses.

$$\textsc{Ace}(x) = occ_3(\neg x) + \sum_{Q_i \epsilon \mathcal{E}(x)} eq_{|Q_i|-1} + \sum_{\neg x \vee y \epsilon \mathcal{F}} \left(occ_3(\neg y) + \sum_{Q_i \epsilon \mathcal{E}(y)} eq_{|Q_i|-1} \right) \tag{2}$$

In the versions of march without equivalence reasoning fast, performance is achieved on average by performing look-ahead only on the “best” 10 % of the variables. This constant percentage is not always optimal. It is not even optimal for the benchmarks used in this paper, but since it provided optimal performance on a wide scale of experiments, we restricted ourselves to this 10 %. To illustrate the diversity of partial look-ahead optima, march requires 1120 secondes to solve a benchmark provided by Philips using the 10 % setting (see table 4), while it requires only 761 seconds at the optimal setting of 8 ‰. [4] provides more insight in the behaviour of march using different percentages of variables entering the look-ahead phase.

5 Additional Equivalence Reasoning

Various additional forms of equivalence reasoning are tested. These include:

– Removal of equivalence clauses that have became tautological during the solving phase. This results in a speed-up due to faster propagation.
– Propagation of binary equivalences in the CoE: replacing one of its literals by the other. This increases the chance that a variable occurs twice in an equivalence clause, so both could be removed.
– Prevention of equivalent variables to enter the look-ahead procedure, since equivalent variables will yield an equivalent $\textsc{Diff}$.

Only the last adjustment realised a noticeable speed-up of about 10 %. The gain that other procedures accomplished were comparable to their cost, resulting in a status quo in terms of time.

6 Results

Six solvers are used to compare the results of march: eqsatz[2], lsat[3], satzoo[4], zchaff[5], limmat[6], and OKsolver[7]. The choice for eqsatz and lsat is obvious since

[2] version 2.0, available at http://www.laria.u-picardie.fr/∼cli/EnglishPage.html
[3] version 1.1, provided by authors
[4] version 1.02, available at http://www.math.chalmers.se/∼een/Satzoo/
[5] version 2003.07.01, available at http://www.ee.princeton.edu/∼chaff/zchaff.php
[6] version 1.3, available at http://www2.inf.ethz.ch/personal/biere/projects/limmat/
[7] version 1.2, available at http://cs-svr1.swan.ac.uk/ csoliver/OKsolver.html

they are the only other SAT solvers performing equivalence reasoning. Since equivalence clauses merely occur in handmade and industrial problems, we added some solvers that are considered state-of-the-art in these categories: satzoo and zchaff, respectively. For extended reference we added two winners of the SAT 2002 competition: limmat and OKsolver. The last is also a look-ahead SAT solver.

All solvers were tested on an AMD 2000+ with 128Mb memory running on Mandrake 9.1. Besides the parity32 and the hwb benchmarks, we experimented on the longmult family that arises from bounded model checking [2], five unsolved benchmarks (pyhala-braun-x (pb-x in short) and lisa21-99-a) from the SAT 2002 competition contributed by Pyhala and Aloul, respectively [7], and three factoring problems (2000009987x) contributed by Purdom [11]. Except from both bounded model checking families and the benchmark provided by Philips, all benchmarks were used in the SAT 2003 competition. To enable a comparison with the SAT 2003 results[8], we used the shuffled benchmarks generated for this competition during our experiments. However, these shuffled benchmarks caused a slowdown in performance of eqsatz: e.g. eqsatz solves most original parity32 benchmarks within the 2000 seconds time limit.

Two versions of our solver are used to evaluate performance: the first, march$^\circ$ uses the equation $eq_n = 5.5 \times 0.85^n$ to measure the reduction of the CoE during the look-ahead, and applies it to the calculation of ACE. The second variant, march* does not use the CoE-part during the look-ahead but operates using the original CNF instead. Both march variants use a 10% partial look-ahead.

In table 3, six properties of experimented benchmarks are presented:

#Cls refers to the initial number of clauses
#Var refers to the initial number of variables
#Ind refers to the number of variables in the independent set
#Eq refers to the number of detected equivalence clauses.
#Nt refers to the number of non-tautological equivalences after pre-processing.
|Nt| refers to the average length of the non-tautological equivalences after the pre-processing.

Table 4 shows the performances of the various solvers during our experiments. Of all properties listed above, the average length (|Nt|) appears to be the most useful indicator for when to use march$^\circ$ instead of march*: both families that profit clearly from the equivalence reasoning have a high average length. The slowdown on the longmult family could be explained by the small number of equivalence clauses compared to the number of independent variables: the relatively costly equivalence reasoning is performed during the look-ahead while the differences in the branch decision between march$^\circ$ and march* are small.

[8] results of the SAT 2003 competition are available at www.lri.fr/~simon/contest03/results/

Table 3. Properties of several benchmarks containing equivalence clauses

instance	#Cls	#Var	#Ind	#Eq	#Nt	|Nt|
par32-1	10227	3176	157	1158	218	7.79
par32-2	10253	3176	157	1146	218	7.83
par32-3	10297	3176	157	1168	218	7.84
par32-4	10313	3176	157	1176	218	8.12
par32-5	10325	3176	157	1182	218	7.91
par32-1-c	5254	1315	157	1158	218	7.49
par32-2-c	5206	1303	157	1146	218	8.02
par32-3-c	5294	1325	157	1168	218	7.69
par32-4-c	5326	1333	157	1176	218	7.73
par32-5-c	5350	1339	157	1182	218	8.13
hwb-n20-1	630	134	96	36	35	4.91
hwb-n20-2	630	134	96	36	35	4.88
hwb-n20-3	630	134	96	36	35	4.80
hwb-n22-1	688	144	104	38	37	4.84
hwb-n22-2	688	144	104	38	37	4.56
hwb-n22-3	688	144	104	38	37	4.62
hwb-n24-1	774	162	116	44	43	5.83
hwb-n24-2	774	162	116	44	43	4.86
hwb-n24-3	774	162	116	44	43	4.86
hwb-n26-1	832	172	124	46	45	5.04
hwb-n26-2	832	172	124	46	45	5.24
hwb-n26-3	832	172	124	46	45	5.56
longmult-6	8853	2848	1037	174	90	3.93
longmult-7	10335	3319	1276	203	105	3.93
longmult-8	11877	3810	1534	232	120	3.93
longmult-9	13479	4321	1762	261	135	3.93
longmult-10	15141	4852	2014	290	150	3.93
longmult-11	16863	5403	2310	319	165	3.93
longmult-12	18645	5974	2620	348	180	3.93
longmult-13	20487	6565	2598	377	195	3.93
longmult-14	22389	7176	2761	406	210	3.93
longmult-15	24351	7807	2784	435	225	3.93
pb-s-40-4-03	31795	9638	2860	3002	3001	3.00
pb-s-40-4-04	31795	9638	2860	2936	2935	3.00
pb-u-35-4-03	24320	7383	2132	2220	2219	3.00
pb-u-35-4-04	24320	7383	2131	2277	2276	3.00
lisa21-99-a	7967	1453	1310	460	459	3.87
2000009987fw	12719	3214	1615	1358	1319	3.54
2000009987nc	10516	2710	1303	1286	1262	3.46
2000009987nw	11191	2827	1342	1322	1299	3.38
philips	4456	3642	1005	342	224	3.50

Table 4. Performances of the solvers march, eqsatz, satzoo, lsat, zchaff, limmat, and OKsolver in seconds on various benchmarks with equivalence clauses

instance	march$^\circ$	march*	eqsatz	satzoo	lsat	zchaff	limmat	OKsolver
par32-1	**0.55**	>2000	568.31	>2000	90.85	>2000	>2000	>2000
par32-2	**0.3**	>2000	>2000	>2000	88.43	>2000	>2000	>2000
par32-3	**1.08**	>2000	>2000	>2000	7.54	>2000	>2000	>2000
par32-4	**7.93**	>2000	>2000	>2000	79.87	>2000	>2000	>2000
par32-5	**8.82**	>2000	>2000	>2000	34.41	>2000	>2000	>2000
par32-1-c	**0.47**	>2000	>2000	>2000	3.91	>2000	>2000	>2000
par32-2-c	7.82	>2000	>2000	>2000	**4.45**	>2000	>2000	>2000
par32-3-c	**5.06**	>2000	>2000	>2000	33.59	>2000	>2000	>2000
par32-4-c	**0.39**	>2000	>2000	>2000	52.39	>2000	>2000	>2000
par32-5-c	**6.77**	>2000	>2000	>2000	71.98	>2000	>2000	>2000
hwb-n20-1	**18.05**	39.43	78.51	47.04	771.48	300.72	>2000	80.88
hwb-n20-2	**23.24**	50.82	83.25	73.38	738.16	461.49	>2000	69.58
hwb-n20-3	**16.16**	38.56	75.09	24.37	564.81	257.07	>2000	67.57
hwb-n22-1	**68.27**	164.65	299.45	108.	>2000	785.89	>2000	286.38
hwb-n22-2	**53.67**	145.51	297.3	85.79	>2000	1097.33	>2000	280.53
hwb-n22-3	**58.6**	148.29	306.71	60.6	>2000	1710.10	>2000	318.75
hwb-n24-1	**556.25**	796.56	>2000	624.17	>2000	>2000	>2000	>2000
hwb-n24-2	**463.46**	832.48	>2000	862.86	>2000	>2000	>2000	>2000
hwb-n24-3	**332**.	670.21	>2000	471.73	>2000	>2000	>2000	>2000
hwb-n26-1	**1203.99**	>2000	>2000	>2000	>2000	>2000	>2000	>2000
hwb-n26-2	**1777.78**	>2000	>2000	>2000	>2000	>2000	>2000	>2000
hwb-n26-3	**1703.12**	>2000	>2000	>2000	>2000	>2000	>2000	>2000
longmult-6	7.13	3.98	11.96	5.1	66.32	**1.59**	2.20	12.38
longmult-7	35.79	21.37	55.15	21.74	109.19	**13.32**	28.32	63.17
longmult-8	100.92	70.8	185.85	**66.22**	192.56	68.17	178.42	260.88
longmult-9	202.03	**130.46**	347.75	138.8	289.53	131.46	465.87	526.72
longmult-10	260.06	**168.21**	520.1	232.96	404.27	252.95	764.78	898.33
longmult-11	226.35	**151.36**	662.19	307.62	542.81	344.64	784.88	1167.59
longmult-12	128.57	**89.37**	741.19	338.48	709.15	305.07	784.72	1256.35
longmult-13	67.92	**52.69**	855.14	272.67	901.13	255.96	644.33	1516.62
longmult-14	44.33	**31.78**	985.37	383.08	1112.49	266.54	685.48	1830.4
longmult-15	25.26	**24.54**	1108.80	215.12	1236.6	207.72	630.25	>2000
pb-s-40-4-03	>2000	510.51	>2000	**184.21**	>2000	357.98	>2000	>2000
pb-s-40-4-04	>2000	**600.41**	>2000	>2000	>2000	>2000	>2000	>2000
pb-u-35-4-03	771.24	**698.22**	>2000	>2000	>2000	>2000	>2000	>2000
pb-u-35-4-04	821.43	**736.31**	>2000	>2000	>2000	>2000	>2000	>2000
lisa21-99-a	**21.26**	1170.12	>2000	>2000	>2000	>2000	>2000	>2000
2000009987fw	175.68	**115.89**	521.47	267.81	181.86	116.4	257.1	649.59
2000009987nc	137.41	**84.16**	197.27	167.4	159.55	94.61	206.75	610.84
2000009987nw	135.25	**87.9**	157.03	218.09	166.55	104.84	228.38	486.86
philips	1120.61	**1032.82**	3390.59	2114.64	1277.68	>3600	>3600	1589.59

7 Conclusions

In this paper, we presented a new alignment between equivalence reasoning and look-ahead in a DPLL Sat solver. The resulted solver outperforms existing techniques on benchmarks that contain a significant part of equivalence clauses. Two main features appeared sufficient for effective equivalence reasoning:

– an effective solving procedure for the CoE during the pre-processing.
– an effective evaluation function to measure the reduction of equivalence clauses during the look-ahead procedure.

Additional features of integration may further increase the performance, but substantial gains have not been noticed yet. However, two procedures are worth mentioning: first, integration of the effective evaluation function (Ace) into the pre-selection heuristics of the look-ahead, resulted in a speed-up up to 30%. Second, a small performance gain on practically all benchmarks, with and without equivalence clauses, was achieved by preventing equivalent variables to enter the look-ahead phase.

We conclude that aligning Equivalence- and CNF- reasoning as carried out pays off convincingly. Although some instances are *not* solved without incorporating the CoE reductions during the look-ahead phase (march$^\circ$), others suffer from this additional overhead and are easier solved by updating and investigating the CoE part at the chosen path only (march*).

References

1. D. Le Berre and L. Simon, *The essentials of the SAT'03 Competition.* Springer Verlag, Lecture Notes in Comput. Sci. **2919** (2004), 452–467.
2. A. Biere, A. Cimatti, E.M. Clarke, Y. Zhu, *Symbolic model checking without BDDs.* in Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, Springer Verlag, Lecture Notes in Comput. Sci. **1579** (1999), 193–207.
3. J.M. Crawford, M.J. Kearns, R.E. Schapire, *The Minimal Disagreement parity problem as a hard satisfiability problem.* Draft version (1995).
4. M.J.H. Heule, J.E. van Zwieten, M. Dufour and H. van Maaren, *March_eq, Implementing Efficiency and Additional Reasoning in a Look-ahead SAT Solver.* Appearing in the same volume.
5. O. Kullmann, *Investigating the behaviour of a SAT solver on random formulas.* Submitted to Annals of Mathematics and Artificial Intelligence (2002).
6. C.M. Li, *Equivalent literal propagation in the DLL procedure.* The Renesse issue on satisfiability (2000). Discrete Appl. Math. **130** (2003), no. 2, 251–276.
7. L. Simon, D. Le Berre, and E. Hirsch, *The SAT 2002 competition.* Accepted for publication in Annals of Mathematics and Artificial Intelligence (AMAI) **43** (2005), 343–378.
8. L. Simon, *Sat'03 competition homepage.* http://www.lri.fr/~simon/contest03/results/
9. L. Simon, *Sat'04 competition homepage.* http://www.lri.fr/~simon/contest/results/

10. R. Ostrowski, E. Gregoire, B. Mazure, L. Sais, *Recovering and exploiting structural knowledge from CNF formulas*, in Proc. of the Eighth International Conference on Principles and Practice of Constraint Programming, Springer Verlag, Lecture Notes in Comput. Sci. **2470** (2002), 185–199.
11. P. Purdom and A. Sabry, *CNF Generator for Factoring Problems.* http://www.cs.indiana.edu/cgi-pub/sabry/cnf.htm
12. J.P. Warners, H. van Maaren, *A two phase algorithm for solving a class of hard satisfiability problems.* Oper. Res. Lett. **23** (1998), no. 3-5, 81–88.