Early Conflict Detection Based BCP for SAT Solving

Matthew D. T. Lewis, Tobias Schubert, and Bernd W. Becker

Chair of Computer Architecture,
Albert-Ludwigs-University of Freiburg, Germany
{lewis,schubert,becker}@informatik.uni-freiburg.de

Abstract. This paper describes a new BCP algorithm that improves the performance of Chaff class solvers by reducing the total number of clauses the BCP procedure must evaluate. This is done by: detecting conflicts earlier; evaluating clauses better; and by increasing the controllability of the conflicts which the BCP procedure finds. Solvers like Limmat [10] include a simple *Early Conflict Detection BCP* (ECDB), however we introduce a new aggressive ECDB procedure and the MIRA solver that efficiently incorporates it while easily facilitating comparisons between ECDB modes. With the full ECDB procedure enabled, MIRA was able to reduce the number of evaluated clauses by 59% on average compared to the disabled ECDB version. This new procedure and other speedup techniques discussed here allow MIRA to solve problems 3.7 times faster on average than zChaff. Lastly, this paper shows how significant speedup can be attained relatively easily by incorporating a certain level of ECDB into other solvers.

1 Introduction

SAT solvers have advanced substantially over the last few years making them an evermore-increasing part of the EDA domain. While most solvers are still based on the DPLL [7] algorithm, they contain significant enhancements such as: conflict analysis with non-chronological backtracking; efficient Boolean Constraint Propagation (BCP) procedures; and better decision-making heuristics. Conflict analysis was first introduced in Grasp [5], and is currently the backbone of most new solvers. Secondly, Chaff [2–4] advanced the BCP procedure by introducing watched literal lists. Furthermore, Chaff provided a simple but highly optimized BCP implementation. In this paper we introduce a more complex, but more efficient *Early Conflict Detection BCP* (ECDB) procedure that is able to achieve speedup by significantly reducing the total amount of work that the BCP procedure must perform. Multiple versions of the ECDB procedure will be discussed, varying in complexity and performance. The ability to integrate some of these ideas in current solvers will also be highlighted.

The remainder of this paper starts by providing an overview of the problem and terminology followed by a brief description of current SAT solvers. Section 3 and 4 describe the MIRA solver and ECDB with Section 5 explaining how ECDB is realized in the MIRA solver. While ECDB is the main focus of this paper, section 6 through 8 will discuss other important parts of the MIRA solver that aid ECDB such as MIRA's Decision Heuristics (for decision making and ECDB) and Conflict Analysis Procedure that can easily cope with a re-ordered implication queue. Lastly, this paper assumes the reader has a thorough knowledge of the inner workings of a modern SAT solver. It therefore provides only a preliminary overview of how current Chaff based SAT solvers work. However, for an excellent and complete overview of a modern SAT solver, refer to [8].

2 Overview and Terminology

Boolean Satisfiability (SAT) problems are usually specified in conjunctive normal form (CNF). A CNF file consists of a list of clauses that are logically ANDed together. Each clause then consists of the logical ORing of one or more literals (Boolean variables or their complements, shown by $\neg$). A simple example is shown below.

$$F(x_1,\ldots,x_n) = (\neg x_1 + x_2) \bullet (\neg x_1 + \neg x_2 + x_3) \bullet (\neg x_1 + \neg x_2 + \neg x_3) \bullet \ldots \qquad (1)$$

A Chaff type solver starts by assigning a value to a literal based on a decision heuristic. This literal is called the current decision literal. The solver then invokes the BCP procedure which detects all implications and conflicts associated with this decision. Any implications that are found are added to the implication queue. This queue contains the consequences of assigning the current decision literal to its current value. Implications in this queue can also trigger other implications. The BCP procedure must process all these implications before the solver is free to make another decision. If in the process of evaluating this queue it runs into a conflict, in which two clauses force a single literal to opposite values, a conflict signal is generated. A conflict analysis procedure is then invoked to find the reason and the origin of the conflict. Based on this information, the assignment stack (chronologically ordered list of all defined literals) is corrected to resolve the conflict. The BCP procedure is then rerun to check the implications of this correction. The decision-making, BCP, and conflict analysis procedures continue until either a solution is found or the problem is proven unsatisfiable.

When evaluating clauses during the BCP stage, the implication queue can become quite large (100's of entries in the queue are normal). The larger it becomes, the more information about the current problem it contains; and the better the probability of a conflict existing within the queue. The idea behind the ECDB procedure is to utilize this information to generate unit clauses and subsequently conflict clauses sooner. An example of how this is done will be shown in section 4. Secondly, the implication queue can be used to better evaluate clauses. This better evaluation of clauses leads to better choices for watched literals, reducing the chance that a clause needs to be re-evaluated in the future. This better utilization of the implication queue is the major improvement when compared to other solvers.

3 The MIRA Solver and ECDB

MIRA was created because Full ECDB could not be efficiently implemented in Chaff. MIRA is still based on Chaff inheriting such features as conflict analysis and fast BCP using watched literals. MIRA, however, gains considerable speedup over Chaff through the use of a new ECDB procedure that has three modes of operation: *Full ECDB; Partial ECDB; and No ECDB.* Note that all three versions of MIRA share the same Decision, Conflict Analysis, and Clause Management procedures.

3.1 No ECDB

The *No ECDB* mode contains no improvements to detect conflicts early. This makes it more like a Chaff type solver for comparison.

3.2 Partial ECDB

The first improvement of the *Partial ECDB* mode when compared to Chaff is how unit clauses are handled. Given an implication queue, Chaff would examine all entries in chronological order ignoring conflicting implications (e.g. two separate clauses that force a single literal but to opposite values). Before Chaff detects this conflict, it must process the entire implication queue up until the point where the first of the conflicting implications is located (assuming no other conflicts exist). The implication queue can be quite large, forcing Chaff's BCP procedure to process many clauses that are not required. MIRA in contrast, checks the implication queue for conflicts before adding implications. If a conflict exists, the conflict resolution procedure is immediately invoked. This is similar to what is done in Limmat [10].

This partial version also goes beyond Limmat allowing for fast clause evaluation when the other watched literal is correctly defined, even if that literal is in the implication queue and has not yet been processed by the BCP procedure. This is done as the implication queue and assignment stack are already available together because of the full ECDB part of MIRA. This also allows for a better utilization of the data structure MIRA uses to store clauses. Note that when MIRA is in Partial ECDB mode the clause data structure is the same as the one presented in Figure 1 below, however the CV variable is not used. The data structure is therefore similar to the one presented in [9] where the authors compared different WL schemes.

3.3 Full ECDB

The *Full ECDB* mode of MIRA goes even further. Here, the ECDB procedure is still similar to Chaff when it traverses the watched literal list of the current decision variable. However, unlike Chaff which checks literals in the order that they were added to the implication queue, MIRA first checks the implications that are most likely to cause a conflict. The heuristic for this is described in section 6. Secondly, once a literal has been added to the implication queue, it becomes immediately defined throughout the entire BCP procedure. Consequently, the BCP procedure uses both the assignment stack and implication queue when evaluating a particular clause for a free or properly assigned literal (a literal assignment that satisfies the clause). This ensures that it does not choose a new watched literal that is improperly assigned in the implication queue and thus reduces the probability that the clause will be re-evaluated. Moreover, evaluating clauses in this manner allows the BCP procedure to generate unit and conflict clauses sooner. An example of how powerful Early Conflict Detection is will be discussed below in section 4. Also, since Chaff's conflict analysis procedure as is cannot handle re-ordered assignment stacks, MIRA's procedure will be discussed in section 7.

4 Full ECDB Example

Using the sample CNF from section 2 for this example, Chaff would start by assigning $x_1 = 1$. It would then check all the clauses (assuming Chaff is currently watching the first two literals of every clause). While checking the clauses it would find the unit clause $(\neg x_1 + x_2)$ and assign $x_2 = 1$ in the implication queue. Once done checking all the clauses in the watched literal list for $x_1 = 1$, it would then move onto the next literal in the implication queue, forcing $x_2 = 1$. It must then look at clauses that contain $\neg x_2$ as a watched literal (in this case the last two clauses). Again, while going through the clauses it could find two more unit clauses and it would add x_3 to the implication queue. Partial ECDB would find the conflict here as two clauses force x_3 to opposite values. Chaff though, would continue with $x_2 = 1$ until all clauses are checked. Then it would check clauses for x_3, and only then will it find the conflict.

In Full ECDB mode, the conflict is generated on the first pass. If $x_1 = 1$ is the first decision the ECDB procedure will immediately add $x_2 = 1$ to the assignment stack as soon as it evaluates the first clause. The ECDB procedure will then evaluate the second clause. It will see that since $x_1 = 1$ and $x_2 = 1$, $x_3 = 1$ is the only possible solution (unit clause). Again it will add $x_3 = 1$ to the assignment stack. Then, when it evaluates the third clause a conflict will be immediately generated.

By utilizing more of the information available at each step of the ECDB procedure, the unit clause stack grows faster than Chaff's, producing unit clauses faster, and then either arriving at an answer or a conflict sooner. This significantly reduces the number of clauses the BCP procedure evaluates. Also, by allowing the BCP procedure to make better choices when exchanging the watched literals, it can further reduce future BCP operations. An example of this is when Chaff evaluates a clause, it might replace the current watched literal with a literal that is wrongly assigned in the implication queue. In this way MIRA avoids the future re-evaluation of the clause that Chaff would be required to do.

5 Realization of the ECDB Procedure

The BCP procedure of any SAT Solver normally represents over 80% of the processing time [2], so it is crucial that it is efficiently implemented. To optimize the BCP procedure in MIRA, the implication queue was combined with the assignment stack. A separate *Literal Value Array* (LVA) was created for the BCP procedure when evaluating clauses. The LVA contains the value for all literals that are defined (decisions and implications). This allows the ECDB to easily check clauses as it only uses the one array. If it finds a unit clause, it must update both the assignment stack and the LVA. The same is true when backtracking. Lastly, there is a pointer that points to the end of the assignment stack and another that points to the current position of the ECDB procedure within this stack. The second pointer is required because the ECDB procedure must still check all implications that are associated with the current decision to ensure correctness. In this way all clauses that are in each decision's or implication's watched literal clause list are eventually checked if no conflict is found. Note that this stack will routinely be re-ordered to ensure the *best* implications are checked first and the heuristic for this is described in section 6.

Fig. 1. Clause Data Structure

WL0	WL1	CV	x_1	x_2	$\dots$	x_n

The ECDB procedure also arranges clauses into a special data structure that allows it to work efficiently. In Chaff's watched literal scheme the pointers for the watched literals float, meaning their position in the clause is not fixed. This is nice when replacing watched literals because no literal swapping is required, however because of this, Chaff does not know the exact position of the both watched literals within a clause. In MIRA, each clause follows the data structure in Figure 1. In the diagram WL0 and WL1 are the two watched literals of that clause. The watched literals of a clause are always located in the first two positions. This structure is similar to the one presented in [10] but contains the CV variable which is an important modification and will be discussed later. This structure is used in all three ECDB versions of MIRA and allows the ECDB procedure to easily check the value of the other watched literal. The other or second watched literal can have three possible values: properly defined; undefined; or improperly defined.

If the second watched literal is *properly defined*, which on the benchmarks shown in section 9 is on average about 60% of the time, the ECDB procedure can move onto the next clause without evaluating the rest of the current clause. This results in a significant increase in BCP clause throughput (the number of clauses checked by the BCP procedure in a set period of time). For this very reason and to further improve the chance that it does not need to evaluate the entire clause, the Partial ECDB version incorporates the implication queue when examining the other watched literal. Also, since both watched literals are in sequential memory addresses, it is likely that they are both loaded into the same cache line. This reduces accesses to main memory, increasing the effectiveness of the cache.

In the second most common case, where the other watched literal is *undefined*, the procedure is the same as Chaff's, and MIRA looks for a new literal to replace the current one. If this fails, a unit clause is reported and a variable is added to the implication queue.

When using the Full ECDB procedure there is also a third and more complicated case when the second watched literal is *incorrectly defined*. In Chaff, this case would constitute a conflict, but in MIRA implications can be defined but not yet processed. This means that Full ECDB procedure does not immediately know if the current clause is a conflicting clause and it must therefor search the clause for a free or properly defined literal. If during the search a free or properly defined literal is not found, a conflict will be signalled and the conflict resolution procedure will be run. However, if a free or properly defined literal is found, MIRA will replace the current WL with it. The algorithm will then continue searching the clause for a second undefined or properly defined literal. If it does not find one then the current clause is a unit clause and MIRA will define the literal that needs to be defined. However, if MIRA finds another free literal, it will be swapped with the cache variable (called the CV, and is the third variable in a clause) and not with the other watched literal. This is because it is difficult for MIRA to find (and then remove) the reference to the current clause from the other watched literal's clause list (the list of all clauses that have this literal as a watched literal). The CV is there to reduce the future work the BCP must do when it tries to replace the other watched literal, as there is a good chance the third literal is free or properly defined. This avoids researching the entire clause. The use of the CV further reduces the effect of the second search because of its physical location which is normally on the same cache line as the watched literals. This further increases the cache friendliness as each clause has a relatively small effective cache foot print.

To summarize, the efficiency is increased for the first case because the positions of both WLs are known, allowing MIRA to skip the search for this variable in the clause. In the second case the efficiency is comparable to Chaff. The third case is a bit more complicated, but we only loose performance when this third case results in a conflict as MIRA, unlike Chaff, must search the entire clause. Normally this is the case for less then 0.1% of all evaluated clauses therefor having only a negligible effect on perfomance.

Lastly, there are some clauses that cannot fit into this data structure, namely clauses with only one literal. If MIRA finds such a clause (through initialization or conflict analysis), the literal is forced to decision level 0 with all other single literal clauses. MIRA then restarts the search from decision level 0. Note that this is similar to random restarts used in many solvers but here it reduces the complexity of the ECDB procedure as it can always assume that every clause has two watched literals.

6 Decision Heuristic

The next area where MIRA differs from that of Chaff is in the decision-making. Chaff uses the most common literal scheme which is referred to as the Variable State Independent Decaying Sum (VSIDS) heuristic. This heuristic uses a counter for every variable that is incremented every time a variable is added to a clause. These counters are also frequently reduced (divided by 2) after some set period or time. This strategy is the one used in zChaff. MIRA improves on this strategy slightly by incrementing the counts for each variable the conflict analysis looked at, even if the variable does not appear in the newly generated conflict clause. This idea was taken from BerkMin [1] because, while certain literals do not appear in conflict clauses, they can still be a focal point of many conflicts. MIRA lastly adds an addition point for the UIP and conflicting variable. This part of the decision strategy is the same for all three versions of MIRA in order to insure a more equal comparison between ECDB modes.

As stated earlier, when MIRA is in *Full ECDB* mode, the ECDB procedure also uses a second decision heuristic when deciding in which order to check the implications contained in the implication queue. In MIRA the ECDB procedure chooses an implication with the highest chance of creating a conflict. This is accomplished by using the same counters as the normal decision heuristic, however the ECDB procedure does not choose the literal with the highest count, rather it chooses the literal whose complement has the highest count. This strategy not only finds conflicts faster but also accelerates the growth of implication queue, further aiding the ECDB procedure in evaluating clauses. This decision heuristic also has the added benefit of focusing the BCP procedure, allowing it to find conflicts that are more related to the current search space (and not a random conflict that is one of many unfound conflicts that normally exist in the implication queue). Also, the addition of a decision strategy in the ECDB procedure can increase the performance of all three versions of MIRA (full, partial, and no ECDB), although our experience has shown that Full ECDB can take better advantage of it.

7 Conflict Analysis

The conflict analysis algorithm that is used in MIRA is implemented differently than that of Chaff's in order to handle MIRA's implication queue, but the results are the same. Both algorithms generate conflict clauses based on the first UIP. The first UIP conflict analysis scheme traces conflicts back to the first decision or implication that dominates the path towards both values of the conflicting literal. For a more in-depth look at first UIP conflicts and other conflict analysis schemes, refer to [3].

In Chaff, to generate a first UIP conflict clause, its conflict analysis runs by going chronologically backwards through the list of implications associated with the most recent decision. Chaff starts by marking all literals that are contained inside this list and in the conflicting clause. If a literal in the conflicting clause is not in this list (i.e. it was set at an earlier decision level), Chaff adds it to the new conflict clause. Chaff then continues to go chronologically backward through this list. If a literal was marked by a previous iteration of the loop its forcing clause is analyzed in the same way as the first clause. Otherwise, if the literal is not marked it is skipped. Chaff continues in this manner until all literals in the list have been looked at. The last marked literal in the list becomes the UIP and is used for backtracking.

Chaff's conflict analysis is simple and efficient. However, since MIRA's list of implications on the current decision level are not always in the proper order, due to BCP re-ordering it for example, many modifications to Chaff's conflict analysis procedure would be required, reducing the simplicity and efficiency of the algorithm. Instead the conflict analysis procedure in MIRA runs using a stack. MIRA starts by examining each literal in the conflicting clause. If the literal was set at the current decision level it is added to stack and marked so it cannot be added again. If it was set before the current decision level then it is added to the conflict clause and also marked so it cannot be added twice. Once finished with the conflicting clause, it takes the most recently implied implication variable out of the stack and repeats the procedure above with the variables respective forcing clause until only one variable is left in the stack. The last variable is then added to the conflicting clause and becomes the UIP literal for backtracking. This stack implementation of the conflict analysis procedure allows MIRA to more easily generate conflict clauses in the absence of a chronologically ordered of implication queue.

8 Clause Management

Modern solvers must manage the large number of conflict clauses they generate. If no clauses are ever deleted, the BCP procedure will become overwhelmed by the sheer number of clauses it must evaluate. To control the growth of the clause database MIRA employs a slightly more complicated clause-cleaning (clause-deleting) algorithm than Chaff. When MIRA decides to delete clauses, it has two criteria it chooses between. If a clause is unsatisfied, MIRA will only delete it if it has lots of undefined literals. This prevents MIRA from deleting long undefined clauses that only have a few undefined literals, as these clauses can still be very useful. Secondly, MIRA deletes any defined clause that is over a certain length. Furthermore, MIRA protects the most recently added clauses from deletion. Lastly, this Clause Management scheme remains constant throughout all three versions of MIRA to insure a more equal comparison between ECDB modes.

9 Results

MIRA was written in C++ using GNU C++ compiler version 2.95.4. The version of zChaff used was 2003.7.22. All comparisons were run on an AMD Athlon XP 1700+ with 1.5GB of RAM. All benchmark were taken from [11].

First, tables 1 and 2 compare MIRA's different modes of operation. These two tables were constructed from only the problems that all three versions solved. Table 1 shows how many clauses on average the BCP procedure evaluated before a conflict was found. This table clearly shows the reduction in BCP operations as the Full ECDB procedure can find conflicts by only searching half the number of clauses as the No ECDB version. Note that on industrial and hand made problems the performance of ECDB is far better than on random problems. Random problems don't seem to have as much clause interdependence which limits implication queue growth thereby reducing ECDB's effectiveness. Lastly, even partial ECDB performed well.

Table 2 shows the total number of BCP operations performed and the total run time for each of the solvers. Again the Full ECDB version reduced the total BCP work to 40% of a regular BCP procedure, reducing run time to 44% of the No ECDB version. This reduction in BCP operations is not only due to finding conflicts sooner, but also due to choosing better watched literals and focusing the search. The Partial ECDB version again performed well here scaling almost linearly with the number of BCP operations. The excellent run time scaling was due to the addition of the implication queue when evaluating if the second watched literal was properly defined.

The next results are shown to prove that even though MIRA's actual code is not as optimized as zChaffs, it still provides significant speedup due to the better BCP algorithm. Table 3 shows a comparison on a variety of Industrial, Hand Made, and Random in which each solver was given 1800 seconds to solve the problem. The time column for each solver has three times associated with it. The first is the time required for problems zChaff was able to prove. Since MIRA solved every problem that zChaff did, this was done to indicate the speedup of MIRA over zChaff. The second is the time required for both solvers to solve instances that were solved by at least one solver. The last time shown is the total time given to the solver. As can be seen, significant speedup was obtained and MIRA was consistently faster on all test problem sets excluding UNSAT random ones. The average speedup was 3.7 times but on certain problem sets it was close to 20 times. MIRA was also able to solve all instances that Chaff solved, plus 4 instances that zChaff could not, making it more robust than zChaff.

10 Conclusion

A new BCP procedure was presented that introduces novel ways to utilize more of the information available to the solver, in particular the implication queue, in order to gain speedup. This new aggressive *Early Conflict Detection BCP* procedure (ECDB) that allows MIRA to better evaluate clauses while detecting unit and conflict clauses sooner, is the main contribution of this paper. The efficient implementation and incorporation of the ECDB procedure into MIRA and further improvements were discussed. MIRA's three modes of operation were compared, and performance increases were highlighted. Speedup was demonstrated for both levels of ECDB when compared to the case without. MIRA was then compared against zChaff to show that these improvements are practical as MIRA is able to compete with a modern SAT solver. Lastly, our results and experiences suggest that certain levels of ECDB can not only accelerate the performance of future SAT solvers, but can be easily integrated into most existing modern SAT solvers, allowing them to realize ECDB speedup.

References

1. Goldberg, E., Novikov, Y.: BerkMin: a Fast and Robust Sat-Solver. DATE. Paris. (2002) 142–149.
2. Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S.: Engineering an Effecient SAT Solver. DAC, (2001)
3. Zhang, L., Madigan, C. F., Moskewicz, M. H., Malik, S.: Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. ICCAD. (2001)
4. Zhang, L., Malik, S.: Cache Performance of SAT Solvers: A Case Study for Efficient Implementation of Algorithms. SAT. (2003)
5. Marques-Silva, J. P., Sakallah, K. A.:GRASP: A Search Algorithm for Propositional Satisfiability. IEEE Transactions on Computers. Vol. 48 (1999) 506–521
6. Lynce, I., Marques-Silva, J.:Efficient Data Structures for Fast SAT Solvers. (2001)
7. Davis, S., Putnam, M.:A Computing Procedure for Quantification Theory. Journal of the ACM. Vol. 7 (1960) 201–215
8. Sörensson, N., Eén, N.:An Extensible SAT-solver. SAT. (2003)
9. Van Gelder, A.:Generalizations of Watched Literals for Backtracking Search. (2001)
10. Biere, A.:Limmat Solver. `http://www.inf.ethz.ch/personal/biere/projects/limmat/`
11. SAT2003 and SAT2004 benchmarks from SATLIB: `http://www.satlib.org`

Table 1. ECDB Enabled Work Reduction

	Full ECDB Mode	Partial ECDB Mode	No ECDB Mode
Maris	635.04 (47,5%)	1001.28 (74,9%)	1336.72 (100%)
DinPhil	1194.13 (56,0%)	1673.27 (78,5%)	2132.78 (100%)
GoldBerg randnet	1243.18 (73,0%)	1493.70 (87,7%)	1702.87 (100%)

Table 2. BCP and Run Time Comparison

	Num. Evaluated Clauses	Total Run Time
No ECDB	2,048,787,866 (100%)	1139.92 (100%)
Partial ECDB	1,165,544,067 (56,9%)	649.92 (57,0%)
Full ECDB	831,466,209 (40,6%)	502.59 (44,1%)

Table 3. MIRA vs zChaff

Problem Set	Number of Instances	Full ECDB		zChaff		Speedup
		$Time^1(sec.)$ $Time^2(sec.)$ $Time^3(sec.)$	Number of Aborts	$Time^1(sec.)$ $Time^2(sec.)$ $Time^3(sec.)$	Number of Aborts	
Maris	24 Total (17 S, 7 U)	73,71 73,71 21673,71	12	653,61 653,61 22253,61	12	8,87 8,87
Bejing	16 Total (15 S, 1 U)	100,12 100,12 3700,12	2	313,73 313,73 3913,73	2	3,03 3,03
DinPhil (SAT)	11 Total (11 S, 0 U)	60,41 83,37 83,37	0	1116,66 2916,66 2916,66	1	18,48 34,98
DinPhil (UNSAT)	11 Total (0 S, 11 U)	178,06 1038,43 4638,43	2	1050,42 2850,42 6450,42	3	5,90 2,74
Moore Hidden K3 S2 (Random)	24 Total (18 S, 0 U)	759,52 867,72 36867,72	20	1769,58 3569,58 39569,58	21	2,33 4,11
Goldberg (UNSAT)	24 Total (0 S, 24 U)	190,41 1821,74 28821,74	15	134,72 1934,72 28934,72	16	0,71 1,06
Total	113 Total (70 S, 43 U)	1362,11 3985,09 95785,09	51	5038,72 12238,72 104038,72	55	3,70 3,07

[1]Time required to solve problems zChaff solved. [2]Time required to solve problems either solver solved. [3]Total time used by the solver.